

The screenshot shows the DOM tree of a web page. At the top, there's a link to a Bootstrap CSS file. Below it, the main content starts with a navigation bar (navbar) containing a form (todo-form). The form includes a heading (h2) labeled 'Tasks' and a list group (list-group) for todos. Each todo item is represented by a list item (li) with an ID like 'task-1517176192142'. The DOM tree also shows various style elements and comments throughout the code.

```
<todo-application>
  #shadow-root (open)
    <link rel="stylesheet" type="text/css" href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css">
    <style>...</style>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark">...</nav>
    <main class="container">
      <todo-form>
        <style>...</style>
        <div class="card todo-form">...</div>
      </todo-form>
      <hr>
      <todo-list ref="list">
        <style>...</style>
        <h2>Tasks:</h2>
        <ul ref="todos" class="list-group">
          <todo-task ref="task-1517176192142" id="task-1517176192142">...</todo-task> == $0
          <todo-task ref="task-1517176320397" id="task-1517176320397">...</todo-task>
          <todo-task ref="task-1517176320397" id="task-1517176320397">...</todo-task>
        </ul>
      </todo-list>
    ...
```

DOM (material complementario)

Document Object Model

El Modelo de Objetos del Documento (DOM) es una estructura de objetos generada por el navegador, la cual representa la página HTML actual. Con JavaScript la empleamos para acceder y modificar de forma dinámica elementos de la interfaz.

Es decir que, por ejemplo, desde JS podemos modificar el texto contenido de una etiqueta <h1>.

¿Cómo funciona?

- La estructura de un documento HTML son las etiquetas.
- En el Modelo de Objetos del Documento (DOM), cada etiqueta HTML es un objeto, al que podemos llamar nodo. Las etiquetas anidadas son llamadas “nodos hijos” de la etiqueta “nodo padre” que las contiene.
- Todos estos objetos son accesibles empleando JavaScript mediante el objeto global document
- Por ejemplo, document.body es el nodo que representa la etiqueta <body>

Estructura

Cada etiqueta HTML 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 HTML se transforma en un nodo que deriva del correspondiente a su "etiqueta padre".

La transformación de las etiquetas HTML 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.

Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque los más usados son:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- 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 HTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta HTML.
- Comment, representa los comentarios incluidos en la página HTML.

Para acceder a los nodos

Aa Selectores	☰ Definición
<code>document.getElementById("mild")</code>	Este selector nos retorna el primer elemento que coincide con ese id
<code>document.getElementsByClassName("miClase")</code>	Este selector nos retorna una lista de elementos que coinciden con la clase brindada
<code>document.querySelector(".unaClase")</code>	Este selector es probablemente el que más utilizas, este nos permite buscar un elemento utilizando la sintaxis de CSS. Esto quiere decir que podemos buscar una clase utilizando ".clase", un id "#id" o un elemento en sí "div"
<code>document.getElementsByTagName("div")</code>	Nos devuelve todas las etiquetas que coincidan con el tipo dado.

Eventos

JavaScript es un lenguaje de programación basado en eventos. Antes de continuar, es necesario que demos un viaje rápido por encima de los "eventos", la madre de las interacciones modernas en JS. Los eventos, son señales, que se disparan cuando ocurre algo en particular en nuestro DOM, nosotros con JS podemos escuchar estos eventos, utilizando "listeners", los cuales depositamos en aquellos elementos que nos interesa saber qué es lo que está ocurriendo. Entre los eventos que más utilizaremos (cabe aclarar que hay una lista gigantesca), están:

Para escuchar eventos

Aa Eventos	☰ Definición
<code>element.addEventListener("click")</code>	Este evento, apunta a cuando el elemento es oprimido.
<code>element.addEventListener("dblclick")</code>	Este evento se dispara cuando el elemento recibe dos clicks seguidos.
<code>element.addEventListener("mouseover")</code>	Este evento se dispara cuando el mouse entra en su área.
<code>element.addEventListener("mouseout")</code>	Este evento se dispara cuando el mouse sale de su área.

Callback

Si asignas una función como parámetro a otra función, hablaremos de un *callback*.

Los **callbacks** aseguran que una función no se va a ejecutar antes de que se complete una tarea, sino que se ejecutará justo después de que la tarea se haya completado. Nos ayuda a desarrollar código JavaScript asíncrono y nos mantiene a salvo de problemas y errores.

<https://www.freecodecamp.org/espanol/news/funciones-callback-en-javascript-que-son-los-callback-en-js-y-como-usarlos/>

Métodos

Aa Nombre del método	☰ Descripción
<code>innerHTML</code>	Reemplaza el contenido de un elemento. Devuelve la sintaxis HTML de un elemento. Se puede agregar etiquetas y texto
<code>textContent</code>	representa el contenido de texto de un nodo y sus descendientes. Se puede usar para agregar o editar texto, no se deben agregar etiquetas.
<code>childNodes</code>	muestra los nodos hijos.
<code>parentNode</code>	es el padre del nodo actual. El padre de un elemento es un nodo del tipo Element , un nodo Document , o un nodo DocumentFragment
<code>createElement</code>	crea un elemento HTML especificado por el nombre de la etiqueta.
<code>appendChild</code>	El método appendChild inserta un nuevo nodo dentro de la estructura DOM de un documento
<code>getAttribute</code>	El método <code>getAttribute</code> devuelve el valor del atributo especificado en el elemento. Si el atributo especificado no existe, el valor retornado puede ser tanto null como una cadena vacía https://www.todo-argentina.net/cursos/dhtml/pagina19.php

Aa Nombre del método	☰ Descripción
<u>setAttribute</u>	Establece el valor de un atributo en la etiqueta indicada. Si el atributo ya existe, el valor es actualizado, en caso contrario, el nuevo atributo es añadido con el nombre y valor indicado
<u>style</u>	permite modificar los estilos de un nodo, agregando propiedades CSS. vinculo.style.background="green"; vinculo.style.color="white";
<u>classList</u>	accedemos a las clases de un nodo

▼ classList (accedemos a las clases)

- .add("string") agrega una clase
- .remove(("string")) elimina una clase
- .contains(("string")) Comprueba si la clase indicada existe en el atributo de clase del elemento.
- .replace(oldClass, newClass) Reemplaza una clase existente por una nueva.
- .toggle(("string")) alterna un elemento entre dos posibilidades, ocultar y mostrar.
- textCss permite agregar varios estilos en una sola linea
parrafos[1].style.cssText = "background: yellow; font-size: 32px;"

Fragment

Investigar que problema resuelve Fragment
<https://www.youtube.com/watch?v=kUpx6ovPILc>

Diferencias entre textContent, innerText y innerHTML

Las diferencias entre `textContent`, `innerText` e `innerHTML` se refieren a cómo se accede y manipula el contenido de un elemento en JavaScript. Aquí tienes una explicación de cada uno de ellos:

`textContent`: El atributo `textContent` devuelve el texto contenido en un elemento, sin tener en cuenta las etiquetas HTML. Esto significa que si hay etiquetas HTML dentro del elemento, `textContent` devolverá solo el texto plano, sin interpretar las etiquetas.

Además, `textContent` es de solo lectura, lo que significa que solo puedes obtener el texto, pero no puedes establecerlo. Ejemplo:

```
var element = document.getElementById("miElemento");
var texto = element.textContent; // Obtiene el texto sin etiquetas HTML
```

`innerText`: El atributo `innerText` también devuelve el texto contenido en un elemento, al igual que `textContent`. Sin embargo, a diferencia de `textContent`, `innerText` tiene en cuenta la representación visual del texto en el navegador, lo que significa que respeta las reglas de estilo y oculta el contenido si está oculto por CSS. También es de solo lectura. Ejemplo:

```
var element = document.getElementById("miElemento");
var texto = element.innerText; // Obtiene el texto considerando el estilo y la visibilidad
```

`innerHTML`: La propiedad `innerHTML` devuelve o establece el contenido HTML completo de un elemento, incluidas las etiquetas HTML. Esto significa que puedes obtener o establecer tanto el texto como las etiquetas HTML dentro de un elemento. Al utilizar `innerHTML` para establecer contenido, debes tener cuidado para evitar ataques de inserción de código malicioso (XSS) si el contenido es proporcionado por los usuarios. Ejemplo:

```
var element = document.getElementById("miElemento");
var contenido = element.innerHTML; // Obtiene el contenido HTML del elemento
element.innerHTML = "<p>Nuevo contenido</p>"; // Establece el contenido HTML del elemento
```

En resumen, `textContent` devuelve solo el texto sin etiquetas HTML, `innerText` devuelve el texto considerando el estilo y la visibilidad, y `innerHTML` devuelve o establece el contenido HTML completo de un elemento, incluidas las etiquetas.