

Guía de Trabajo: DOM en JavaScript

Objetivo

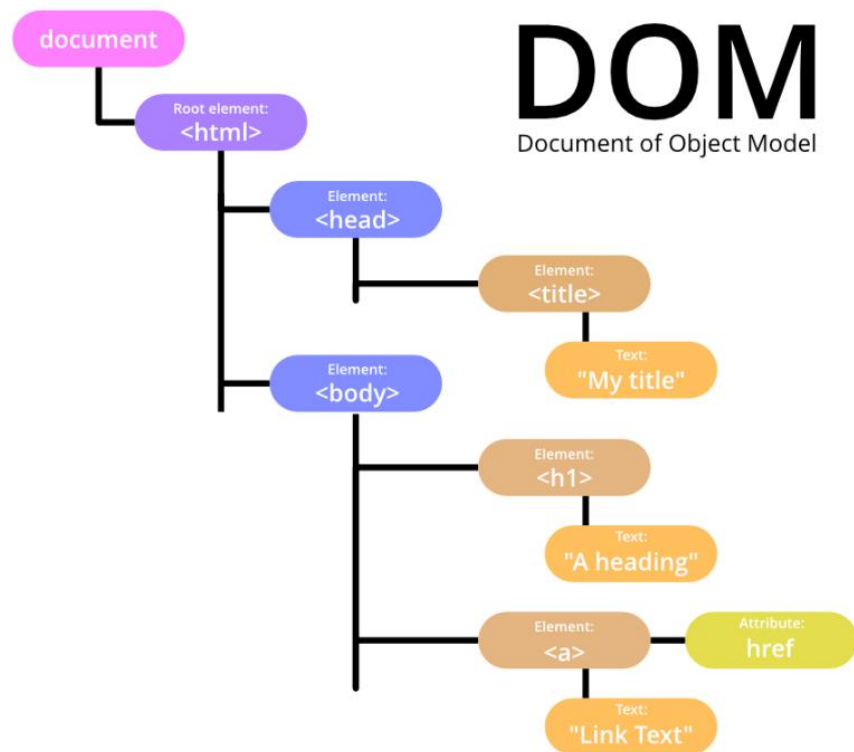
El objetivo del DOM es proporcionar un modelo estructurado de los documentos para que los lenguajes de programación puedan acceder, modificar y actualizar la estructura y el contenido de una página web en tiempo real.

Introducción al DOM

El DOM (Document Object Model) es una interfaz de programación que permite manipular documentos HTML y XML. Cuando un navegador carga una página web, crea una representación del documento en forma de árbol jerárquico, donde cada elemento, atributo o contenido de texto es un nodo.

1. Qué es el DOM

Básicamente, el DOM es una estructura jerárquica en forma de árbol que representa todos los elementos de una página web.



El DOM en Javascript se utiliza para interactuar con los elementos de una página web de manera dinámica. Es decir, que los desarrolladores pueden acceder a los elementos HTML, modificar su contenido, estilo y atributos, pero también agregar o eliminar elementos cuando sea necesario en la maquetación web

2. Componentes clave del DOM

Concepto	Descripción
Nodo	Representa cualquier parte del documento: etiquetas, texto, atributos, etc.
Elemento	Nodo que representa una etiqueta HTML (<code><div></code> , <code><p></code> , <code><h1></code> , etc.).
Propiedad	Información asociada al nodo, como <code>id</code> , <code>class</code> , o <code>innerHTML</code> .
Método	Acción que se puede realizar en un nodo, como <code>getElementById()</code> o <code>appendChild()</code> .

3. El DOM permite:

- Leer, modificar o eliminar elementos HTML.
- Cambiar el estilo de los elementos.
- Responder a eventos como clics y teclas presionadas.

4. Seleccionar elementos en el DOM


Los métodos de selección permiten acceder a nodos específicos del árbol DOM. A continuación, una tabla de los métodos más comunes con sus respectivos ejemplos.

Método/Propiedad	Acción	Ejemplo
<code>getElementById()</code>	Selecciona un elemento por su atributo <code>id</code> .	<pre>const elemento = document.getElementById('miDiv');</pre>
<code>getElementsByClassName()</code>	Selecciona todos los elementos con una clase específica (devuelve una HTMLCollection).	<pre>const elementos = document.getElementsByClassName('miClase');</pre>
<code>getElementsByTagName()</code>	Selecciona todos los elementos de un tipo de etiqueta específica.	<pre>const parrafos = document.getElementsByTagName('p');</pre>
<code>querySelector()</code>	Selecciona el primer elemento que coincide con un selector CSS.	<pre>const elemento = document.querySelector('.miClase');</pre>
<code>querySelectorAll()</code>	Selecciona todos los elementos que coinciden con un selector CSS (devuelve una NodeList).	<pre>const elementos = document.querySelectorAll('.miClase');</pre>

- **getElementById()**

Sintaxis:


javascript

 Copiar código

```
document.getElementById('id');
```

Ejemplo:

html


 Copiar código

```
<div id="miDiv">Hola, soy un div</div>
<script>
  const elemento = document.getElementById('miDiv');
  console.log(elemento.textContent); // Output: Hola, soy un div
</script>
```



- **getElementsByClassName()**


javascript

 Copiar código

```
document.getElementsByClassName('className');
```

Ejemplo:

html


 Copiar código

```
<div class="miClase">Div 1</div>
<div class="miClase">Div 2</div>
<script>
  const elementos = document.getElementsByClassName('miClase');
  console.log(elementos[0].textContent); // Output: Div 1
</script>
```

- **getElementsByTagName()**

Sintaxis:


javascript

 Copiar código

```
document.getElementsByTagName('tagName');
```

Ejemplo:


html

 Copiar código

```
<p>Parrafo 1</p>
<p>Parrafo 2</p>
<script>
  const parrafos = document.getElementsByTagName('p');
  console.log(parrafos[1].textContent); // Output: Parrafo 2
</script>
```

- **querySelector()**


javascript

 Copiar código

```
document.querySelector('selectorCSS');
```

Ejemplo:

html

 Copiar código


```
<div class="miClase">Div único</div>
<div class="miClase">Div ignorado</div>
<script>
  const elemento = document.querySelector('.miClase');
  console.log(elemento.textContent); // Output: Div único
</script>
```



- **querySelectorAll()**

Sintaxis:


javascript

 Copiar código

```
document.querySelectorAll('selectorCSS');
```

Ejemplo:

html

 Copiar código

```
<div class="miClase">Div 1</div>
<div class="miClase">Div 2</div>
<script>
  const elementos = document.querySelectorAll('.miClase');
  elementos.forEach((el) => console.log(el.textContent));
  // Output:
  // Div 1
  // Div 2
</script>
```



5. Principales diferencias

Método	Selector Aceptado	Devuelve	Soporta Selectores CSS Complejos
querySelector	ID, clase, etiqueta, combinaciones	Nodo único	Sí
querySelectorAll	ID, clase, etiqueta, combinaciones	NodeList (similar a arreglo)	Sí
getElementById	ID	Nodo único	No
getElementsByName	Clase	HTMLCollection	No
getElementsByTagName	Etiqueta	HTMLCollection	No

6. ¿Cuál usar y Cuándo?

Método	¿Qué selecciona?	¿Qué devuelve?	¿Cuándo usarlo?
<code>querySelector</code>	El primer elemento que coincida	Un único elemento	Para selecciones flexibles o avanzadas (con un solo resultado)
<code>querySelectorAll</code>	Todos los elementos que coincidan	NodeList (lista de nodos)	Para selecciones flexibles o avanzadas (con varios resultados)
<code>getElementById</code>	Un elemento con un ID específico	Un único elemento	Cuando ya sabes el ID único del elemento
<code>getElementsByClassName</code>	Todos los elementos con una clase	HTMLCollection (dinámica)	Para seleccionar varios elementos por clase
<code>getElementsByTagName</code>	Todos los elementos de una etiqueta	HTMLCollection (dinámica)	Para seleccionar todos los elementos de un tipo específico

7. Manipulación del DOM

Los métodos de manipulación permiten cambiar el contenido, estilo o estructura de un documento.

Método/Propiedad	Acción	Ejemplo
<code>createElement()</code>	Crea un nuevo elemento HTML en el documento.	<pre>const nuevoParrafo = document.createElement('p'); nuevoParrafo.textContent = 'Texto';</pre>
<code>appendChild()</code>	Agrega un nodo como hijo al final de un elemento existente.	<pre>const div = document.getElementById('miDiv'); div.appendChild(nuevoParrafo);</pre>
<code>innerHTML</code>	Establece/obtiene contenido HTML dentro de un elemento.	<pre>const div = document.getElementById('miDiv'); div.innerHTML = '<p>Hola</p>';</pre>
<code>textContent</code>	Establece/obtiene el contenido de texto (sin formato HTML) de un elemento.	<pre>const div = document.getElementById('miDiv'); div.textContent = 'Texto plano';</pre>
<code>style</code>	Manipula los estilos en línea de un elemento.	<pre>const div = document.getElementById('miDiv'); div.style.color = 'blue'; div.style.fontSize = '20px';</pre>
<code>removeChild()</code>	Elimina un nodo hijo de un elemento existente.	<pre>const lista = document.getElementById('miLista'); const item = lista.firstElementChild; lista.removeChild(item);</pre>
<code>setAttribute()</code>	Agrega o modifica un atributo de un elemento.	<pre>const img = document.getElementById('miImg'); img.setAttribute('src', 'imagen.jpg'); img.setAttribute('alt', 'Descripción de imagen');</pre>



4. Eventos en el DOM

Teoría:

Los eventos son acciones realizadas por el usuario o el navegador (clics, teclas presionadas, etc.). A través de JavaScript, podemos responder a estos eventos usando `addEventListener`.

Evento	Descripción
<code>click</code>	Ocurre cuando se hace clic en un elemento.
<code>mouseover</code>	Ocurre cuando el puntero pasa sobre un elemento.
<code>keydown</code>	Ocurre cuando se presiona una tecla.
<code>submit</code>	Ocurre cuando se envía un formulario.

Ejercicio practico

Requerimientos del ejercicio: Tienda en línea

Descripción general

1. Crear una tienda en línea básica con **tres productos** (cada uno con una imagen, nombre, precio y botón "Agregar al carrito").
2. Implementar la funcionalidad de un **carrito de compras**:
 - Los productos pueden agregarse al carrito al presionar el botón correspondiente.
 - Los productos en el carrito deben poder eliminarse si el usuario se equivoca.
3. Cuando se agregue o elimine un producto debe notificarle al usuario que fue agregado o eliminado con éxito.
4. Sumar el valor de cada elemento agregado.
5. Utilizar métodos y eventos del DOM para implementar toda la lógica.

Estructura del HTML

- Crear un contenedor para los productos:
 - Cada producto debe incluir:
 - Una imagen.
 - Un título o descripción.
 - Un precio.
 - Un botón "Agregar al carrito".
- Crear un área para mostrar el carrito:
 - Debe mostrar los productos seleccionados.
 - Cada producto debe incluir un botón "Eliminar".

Manipulación del DOM:

- **createElement**: para crear los elementos del carrito dinámicamente.
- **appendChild**: para agregar productos al carrito.
- **removeChild**: para eliminar productos.

Selección de elementos:

- **querySelector / querySelectorAll.**
- **getElementById**
- **getElementsByClassName()**

Eventos:

- **addEventListener('click', ...)** para manejar los botones de agregar y eliminar.
- (Opcional) **keydown** para manipular con el teclado.

Conclusión

En resumen, el DOM es una herramienta poderosa que permite a los desarrolladores web transformar documentos estáticos en experiencias dinámicas e interactivas. Su conocimiento es fundamental para construir aplicaciones modernas y optimizadas.