

Modelo de objetos del documento (DOM)

Antonio, Daniel, Doryan, Jason, Marcos, Mario, Xing

Tabla de contenidos

Modelo de objetos del documento (DOM).....	2
Árbol de nodos.....	2
Tipos de nodos	4
Propiedades de un nodo	5
Acceso al documento desde código.....	5
Acceso desde propiedades del document.....	5
Acceso a nodo desde métodos del document.....	16
Acceso a nodo utilizando selectores CSS	17
Acceso a nodos a partir de otros	18
Gestión de nodos.....	19
Crear nodo.....	19
Añadir nodo a la página	19
Eliminar nodo.....	20
Modificar el DOM con childNode	20
Ejemplos	21
Gestión de atributos.....	23
Atributos de clase	23
Extras	25
DOM Virtual	25
Bibliografía.....	26

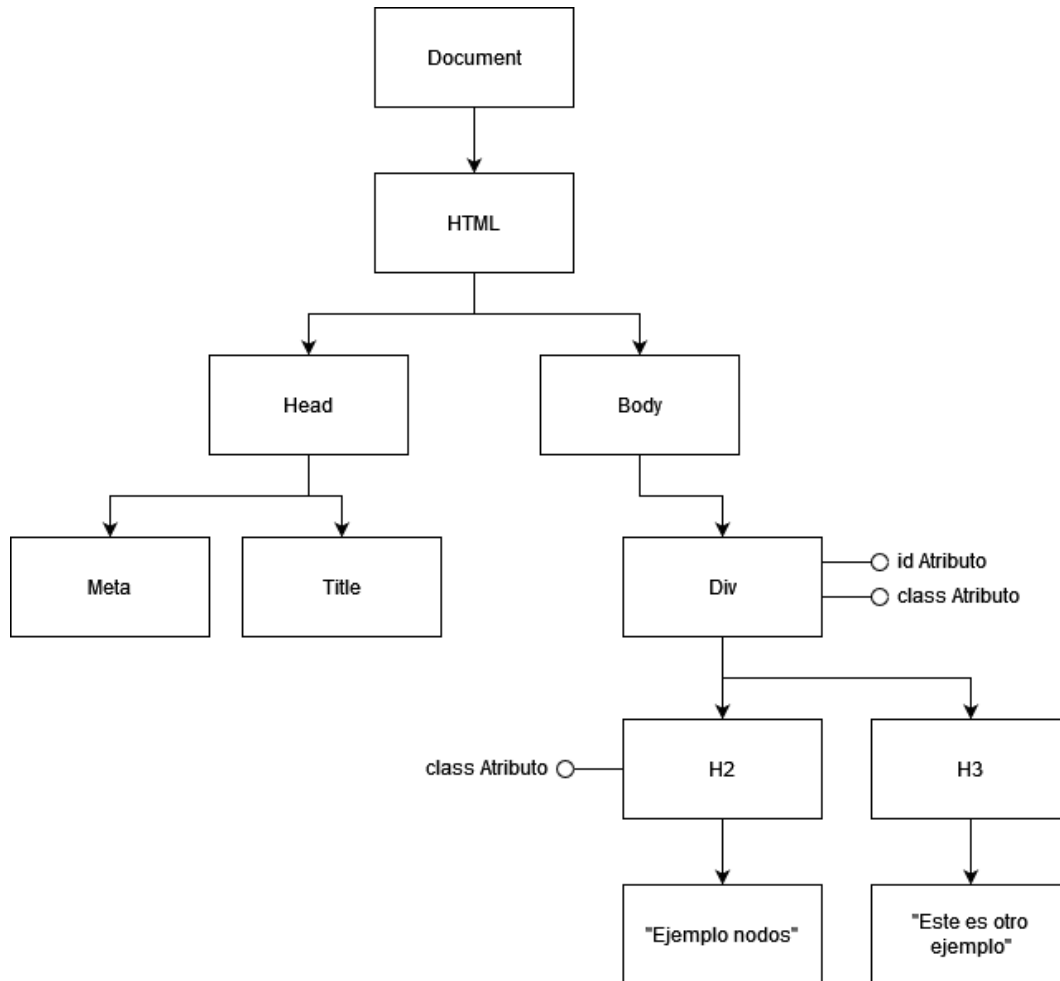
Modelo de objetos del documento (DOM)

DOM (Document Object Model) es una API definida para representar e interactuar con cualquier documento HTML.

El DOM representa el documento como un árbol de nodos, donde cada nodo representa una parte del mismo.

Árbol de nodos

Un ejemplo de árbol de nodos sería algo como esto:



Que en HTML sería algo como:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplos</title>
</head>
<body>
  <div id="contenedor" class="contenedor">
    <h2 class="titulo-ejemplo">Ejemplo nodos</h2>
    <h3>Este es otro ejemplo</h3>
  </div>
</body>
</html>
```

Tipos de nodos

Hay diferentes tipos de nodos:

Tipo	Descripción	Número
Element	Nodos definidos por etiquetas HTML, puede contener atributos y de él pueden derivar otros nodos	1
Attr	Obsoleto nodo que representa cada uno de los atributos de las etiquetas HTML. Aunque con JS no lo vemos como nodos sino como información asociada al nodo de tipo <i>element</i>	2
Text	El texto dentro de un nodo <i>element</i> se considera un nodo de tipo <i>text</i>	3
CDataSection	Obsoleto Representa una sección CDATA en un documento	4
EntityReference	Obsoleto Representa una referencia a entidades XML	5
Entity	Obsoleto Representa una entidad en un documento XML	6
ProcessingInstruction	Representa una instrucción de procesamiento de un documento XML	7
Comment	Representa los comentarios incluidos en el documento HTML	8
Document	Nodo raíz del que derivan los demás nodos del árbol	9
DocumentType	El comentario inicial del documento HTML <!DOCTYPE html>	10
DocumentFragment	Representa un objeto tipo <i>Document</i> que no tiene padre	11
Notation	Obsoleto	12

Propiedades de un nodo

Los nodos tienen propiedades generales que comparten.

Propiedad	Descripción
nodeName	Nombre del nodo
nodeValue	Valor del nodo
nodeType	Número que representa el tipo de nodo (del 1 al 12)
ownerDocument	Documento al que pertenece el nodo
firstChild	Primer hijo del nodo
lastChild	Último hijo del nodo
childNodes	Lista de hijos de un nodo
previousSibling	Hermano anterior al nodo
nextSibling	Hermano siguiente al nodo
hasChildNodes	Indica si el nodo tiene hijos o no
attributes	Lista de los atributos del nodo

Acceso al documento desde código

Acceso desde propiedades del document

Una vez que está construido el DOM ya podemos comenzar a acceder a cualquier nodo del árbol y manipularlo de forma sencilla.

Se puede acceder al código mediante su **id** (`getElementById`), mediante el nombre de su **etiqueta** (`getElementsByTagName`), mediante su **nombre** (`getElementsByName`), mediante el nombre de su **clase** (`getElementsByClassName`), o mediante un **selector CSS** (`querySelectorAll`), pero en este apartado nos centraremos en las formas de acceder las **propiedades del DOM** que no son tan conocidas.

document.anchors

Actualmente **obsoleto**, aunque algunos navegadores aun lo soportan.

Es una propiedad de solo lectura del DOM que retorna una lista con todas las anclas del documento.

Ejemplo de código que genera una tabla de contenidos con todas las anclas del documento.

- HTML:

```
<h1>Titulo</h1>
<a name="contenidos"><h2>Contenidos</h2></a>
<ul id="tabla"></ul>
```

```
<a name="plantas"><h2>Plantas</h2></a>
<ol>
  <li>Naranjas</li>
  <li>Manzanas</li>
  <li>Peras</li>
  <li>Plátanos</li>
</ol>
```

```
<a name="verduras"><h2>Verduras</h2></a>
<ol>
  <li>Zanahoria</li>
  <li>Puerro</li>
  <li>Remolacha</li>
</ol>
```

- JavaScript:

```
window.onload = function() {
  var tabla = document.getElementById("tabla");
  var i, li, nuevaAncla;
  for (i = 0; i < document.anchors.length; i++) {
    li = document.createElement("li");
    nuevaAncla = document.createElement("a");
    nuevaAncla.href = "#" + document.anchors[i].name;
    nuevaAncla.innerHTML = document.anchors[i].text;
    li.appendChild(nuevaAncla);
  }
};
```

- Resultado:

Titulo

Contenidos

- [Contenidos](#)
- [Plantas](#)
- [Verduras](#)

Plantas

1. Naranjas
2. Manzanas
3. Peras
4. Platanos

Verduras

1. Zanahoria
2. Puerro
3. Remolacha

document.body

Retorna el nodo del **<body>** del documento.

- En consola:

```
> document.body
< <body class>
  <script>window.fontTest();</script>
  <span class="font-test" style></span>
  <div class="page-wrapper page-wrapper_sidebar_on"></div>
  <div class="page-footer"></div> (fix)
  <progress class="tutorial-progress sticky" data-sticky value="3" max="32" data-tooltip="Lección 3 de 32" style="position: fixed; top:
0px; left: 0px; z-index: 101; background: white; margin: 0px; width: 1903px;"></progress>
</body>
```

- Ejemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
  <p>Párrafo ejemplo</p>
```

```
  <script>
```

```
    document.body.innerHTML += "Texto añadido usando document.body":
```

```
  </script>
```

```
</body>
```

```
</html>
```


- Resultado:

Parrafo ejemplo

Texto añadido usando document.body

document.documentElement

Es una propiedad de solo lectura que sirve para obtener el elemento raíz del documento, por ejemplo, el elemento **<html>** de un documento HTML.

- Ejemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
  <p>Al hacer click en el botón se mostrará  
  el nombre del nodo del documento usando  
  documentElement.</p>
```

```
  <button onclick="function()">Haz click</button>
```

```
  <p id="ejemplo"></p>
```

```
  <script>
```

```
    function funcion() {
```

```
      var raiz = document.documentElement.nodeName;
```

```
      document.getElementById("ejemplo").innerHTML = raiz;
```

```
    }
```

```
  </script>
```

```
</body>
```

```
</html>
```

- Resultado:

Al hacer click en el botón se mostrará el nombre del nodo del documento usando documentElement.

Haz click

HTML

document.embeds

Esta propiedad retorna un array con todos los elementos embebidos en el documento, es decir, usando la etiqueta **<embed>**.

- En la consola:

Al hacer click en el botón se mostrará el nombre del nodo del documento usando documentElement.

Haz click

HTML

- Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <style> embed{width: 200px;} </style>
</head>
<body>
  <p>Número de elementos embebidos del documento:</p>
  <p id="ejemplo"></p>

  <embed type="image/jpg" src="pic_1.jpg"/>
  <embed type="image/jpg" src="pic_2.jpg"/>

  <script>
    let num = document.embeds.length;
    document.getElementById("ejemplo").innerHTML = num;
  </script>
</body>
</html>
```

- Resultado:

Número de elementos embebidos del documento:

2



document.forms

Es una propiedad de solo lectura que retorna un array con todos los elementos **<form>** del documento.

Método	Descripción
[índice]	Retorna el elemento en la posición indicada. (Empieza en 0)
namedItem(id)	Retorna el elemento con el id indicado.

- Ejemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Ejemplo de document.form</title>
```

```
</head>
```

```
<body>
```

```
  <form id="daniel"></form>
```

```
  <form id="mario"></form>
```

```
  <form id="jason"></form>
```

```
  <input type="button" onclick="console.log(document.forms)" value="Todos los formularios" />
```

```
  <input type="button" onclick="console.log(document.forms[1])" value="Segundo formulario" />
```

```
  <input type="button" onclick="console.log(document.forms.namedItem('daniel'))" value="Formulario con el id Daniel" />
```

```
</body>
```

```
</html>
```

- Resultado:

```
HTMLCollection(3) [form#daniel, form#mario, form#jason, daniel: form#daniel, mario: form#mario, jason: form#jason]
  <form id="mario"></form>
  <form id="daniel"></form>
```

document.head

Es una propiedad que retorna el nodo **<head>**, del documento. Si por alguna razón existiesen dos o más **<head>**, esta propiedad devolvería la primera.

- Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo head</title>
</head>
<body>
  <script>
    console.log(document.head);
  </script>
</body>
</html>
```

- Resultado:

```
▼ <head>
  <title>Ejemplo head</title>
  </head>
```

document.images

Es una propiedad de solo lectura que retorna un array de todas las etiquetas **** del documento. - Ejemplo:

```
<!DOCTYPE html>
<html>
<body>
  
  
  

  <p>Numero de imágenes en el documento:</p>
  <p id="ejemplo"></p>

  <script>
    let num = document.images.length;
    document.getElementById("ejemplo").innerHTML = num;
    console.log(document.images);
  </script>
</body>
</html>
```

- Resultado:



Numero de imagenes en el documento:

3

- Consola:

```
▼ HTMLCollection(3) ⓘ VM24:4
  ▶ 0: img
  ▶ 1: img
  ▶ 2: img
  length: 3
  ▶ [[Prototype]]: HTMLCollection
```

document.links

Esta propiedad de solo lectura retorna un array con todas las etiquetas `<a>` del documento. - Ejemplo:

```
<!DOCTYPE html>
<html>
<body>
  <p><a href="https://www.google.es/">Enlace 1</a></p>
  <p><a href="https://confugiradores.es/">Enlace 2</a></p>

  <p>La URL del segundo enlace es:</p>
  <p id="ejemplo"></p>

  <script>
    let url = document.links.item(1).href;
    document.getElementById("ejemplo").innerHTML = url;
    console.log(document.links);
  </script>
</body>
</html>
```

- Resultado:

[Enlace 1](https://www.google.es/)

[Enlace 2](https://confugiradores.es/)

La URL del segundo enlace es:

<https://confugiradores.es/>

- Consola:

```
▼ HTMLCollection(2)  VM28:4
  ▶ 0: a
  ▶ 1: a
    length: 2
  ▶ [[Prototype]]: HTMLCollection
```

document.scripts

Es una propiedad de solo lectura que retorna un array con todos los `<script>` del documento.

Método	Descripción
[índice] e item(índice)	Retorna el elemento en la posición indicada. (Empieza en 0)

- Ejemplo:

```

<!DOCTYPE html>
<html>
<body>
  <script>
    document.write("ejemplo de document.scripts");
  </script>
  <script>
    document.write("Ejemplo 2");
  </script>
  <script>
    document.write("Ejemplo 3");
  </script>

  <p>El contenido del primer script es:</p>
  <p id="ejemplo"></p>

  <script>
    let ejemplo = document.getElementById("ejemplo");

    ejemplo.innerHTML += document.scripts[0].text;

    ejemplo.innerHTML += document.scripts.item(1).text;

    console.log(document.scripts);
  </script>
</body>
</html>

```

- Resultado:

Ejemplo de document.scripts Ejemplo 2 Ejemplo 3

El contenido del primer script es:

```

document.write("Ejemplo de document.scripts");
document.write("Ejemplo 2");

```

- Consola:

```

▼ HTMLCollection(4) ⓘ VM34:8
  ▶ 0: script
  ▶ 1: script
  ▶ 2: script
  ▶ 3: script
    length: 4
  ▶ [[Prototype]]: HTMLCollection

```

document.title

Es una propiedad que establece o retorna el título del documento. - Ejemplo:

```

<!DOCTYPE html>
<html>
<head>
  <title>Título de ejemplo</title>
</head>
<body>
  <p>El título del documento es: </p>
  <p id="ejemplo"></p>

  <script>
    document.getElementById("ejemplo").innerHTML = `Antes de modificar:
    ${document.title}.`;

    document.title = "Titulo modificado";

    document.getElementById("ejemplo").innerHTML += `<br>Después de modificar:
    ${document.title}.`;
  </script>
</body>
</html>

```

- Resultado:

El título del documento es:

Antes de modificar: Título de ejemplo.

Después de modificar: Título modificado.

Acceso a nodo desde métodos del document

- **getElementById ()**: este método sirve para buscar el atributo id del elemento HTML, en principio un documento HTML bien construido no debe existir más de un elemento con mismo id, por lo tanto, este método siempre devuelve una sola elemento, formato:

```
const nombreDeVariable = document.getElementById("atributo id de elemento HTML");
```

- **getElementsByClassName()**: este método sirve para buscar elementos que contiene atributo class, es importante darse cuenta es getElements es plural porque puede tener varios elementos con mismo class (al contrario de atributo id) por lo tanto devuelve un array con elemento encontrado, si no encuentra nada se devuelve un array vacío, formato:

```
const nombreDeVariable = document.getElementsByClassName("atributo class de elementos HTML");
```

- **getElementsByTagName()/getElementsByName()**: son exactamente igual los dos métodos, la diferencia es uno busca por atributo name que está dentro de etiqueta HTML y otro se busca por nombre de etiqueta, formato:

```
// el método devuelve un tipo de dato HTMLCollection o NodeList, parecido a array, pero no es array,  
// así que la mejor manera de obtener información de variable es usar el método  
forEach().
```

```
const nombreDeVariable = document.getElementsByTagName("atributo name de etiqueta HTML");
```

```
// el método devuelve un dato de tipo array, no se recomienda usar el método  
forEach().
```

```
const nombreDeVariable = document.getElementsByTagName("nombre de etiqueta");
```

Acceso a nodo utilizando selectores CSS

querySelector(sel)/querySelectorAll(sel): los dos métodos tienen la misma función de buscar elemento que coincide con el selector css sel, la diferencia es:

- querySelector(sel): solo puede seleccionar una sola elemento y si no tiene nada se devuelve null.
- querySelectorAll(sel): puede seleccionar varios elementos y si no tienes nada devuelve un corchete [] (array vacío).

querySelector tiene una función muy parecida a getElementById la única diferencia es que querySelector(sel) tiene indicado sel como parámetro y siempre es una string, ejemplo:

//con # se indica que es de tipo id.

```
const caja = document.querySelector("#caja");
```

//no necesita poner # para indicar que es de tipo id.

```
const caja = document.getElementById("caja");
```

querySelectorAll puede seleccionar varias clases o varios atributos que lleva mismo nombre como atributo name.

```
const clases = document.querySelectorAll(".clase1 , .clase2");
```

```
const variable = document.querySelectorAll("[name = nombre]");
```

Acceso a nodos a partir de otros

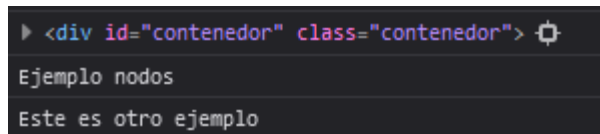
Para acceder usaremos las funciones como: `document.getElementById`, pero en vez de poner *document* pondremos el nombre del nodo que hayamos almacenado anteriormente en una variable como en el siguiente ejemplo:

```
<div id="contenedor" class="contenedor">
  <h2 class="titulo-ejemplo">Ejemplo nodos</h2>
  <h3>Este es otro ejemplo</h3>
</div>

<script>
  const contenedor = document.getElementById('contenedor');
  const title1 = contenedor.querySelector('.titulo-ejemplo');
  const title2 = title1.nextElementSibling;

  console.log(contenedor);
  console.log(title1.textContent);
  console.log(title2.textContent);
</script>
```

Ese pequeño script nos devolvería lo siguiente, demostrando que hemos podido acceder a los nodos de forma correcta:



```
▶ <div id="contenedor" class="contenedor">
Ejemplo nodos
Este es otro ejemplo
```

Gestión de nodos

El DOM tiene métodos para gestionar los nodos.

Crear nodo

- **createElement(etiqueta):** Crea un elemento con el nombre de la etiqueta que se le pasa por parámetro
- **createTextNode(contenido):** Crea un nodo de texto con el *string* indicado por parámetro

```
const li = document.createElement("li");
const text = document.createTextNode("Esto es una lista");
```

Añadir nodo a la página

- **nodoPadre.appendChild(nodoHijo):** Añade un elemento al final de la lista a un elemento padre

// Suponiendo que ya existiera una lista en el código HTML

```
const ul = document.getElementById("lista");
```

```
ul.appendChild(li); // Se añade el <li> al <ul>
```

```
li.appendChild(text); // Se añade el texto al <li>
```

El resultado de la ejecución del código de arriba es el siguiente:

```
<body>
  <ul id="lista">
    <!-- Se ha añadido un elemento a la lista -->
    <li>Esto es una lista</li>
  </ul>
</body>
```

Eliminar nodo

- **nodoPadre.removeChild(nodoHijo):** Elimina el nodo pasado por parámetro
// Se elimina el elemento de la lista que habíamos añadido anteriormente
`ul.removeChild(li);`

```
<body>
  <ul id="lista">
    <!-- La lista se vuelve a quedar vacía -->
  </ul>
</body>
```

Modificar el DOM con childNode

- **element.childNodes:** Devuelve una lista de tipo NodeList en la que se guardan todos los nodos del elemento indicado. Al igual que en un *array*, se pueden acceder a sus posiciones mediante índices

Partiendo de la siguiente estructura HTML:

```
<body>
  <ul id="lista">
    <li id="n1" class="elemento-lista">Uno</li>
    <li id="n2" class="elemento-lista">Dos</li>
    <li id="n3" class="elemento-lista">Tres</li>
  </ul>
</body>
```

Al llamar al método *childNodes* sobre la lista:

```
const nodes = lista.childNodes;
```

```
console.table(nodes);
```

El resultado en la consola sería el siguiente:

Índice	Valor
0	li#n1.elemento-lista
1	li#n2.elemento-lista
2	li#n3.elemento-lista

Ejemplos

// Crear nodo de tipo Element

```
var parrafo = document.createElement("p");
```

// Crear nodo de tipo Text

```
var contenido = document.createTextNode("Hola Mundo!");
```

// Añadir el nodo Element como hijo de la página

```
parrafo.appendChild(contenido);
```

```
document.body.appendChild(parrafo);
```

// Eliminar nodos

```
var parrafo = document.getElementById("provisional");
```

```
parrafo.parentNode.removeChild(parrafo);
```

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

// Ejemplo en detalle

<p title="Texto de un párrafo" id="parrafito">Esto es un ejemplo de ****texto HTML**
** que puedes tener**** en tu documento.**</p>**

```
var nuevoParrafo = document.createElement('p');
```

```
var nuevoTexto = document.createTextNode('Contenido añadido al párrafo.');
```

// Creamos tres elementos nuevos: p, b, br

```
var elementoP = document.createElement('p');
```

```
var elementoB = document.createElement('b');
```

```
var elementoBR = document.createElement('br');
```

// Le asignamos un nuevo atributo title al elementoP que hemos creado.

```
elementoP.setAttribute('title', 'Párrafo creado desde JavaScript');
```

// Preparamos los nodos de texto

```
var texto1 = document.createTextNode('Con JavaScript se ');
```

```
var texto2 = document.createTextNode('pueden realizar ');
```

```
var texto3 = document.createTextNode('un montón');
```

```
var texto4 = document.createTextNode(' de cosas sobre el documento.');
```

```
nuevoParrafo.appendChild(nuevoTexto);
```

```
document.getElementById('parrafito').appendChild(nuevoParrafo);
```

// Añadimos al elemento B los nodos de texto2, elementoBR y texto3.

```
elementoB.appendChild(texto2);
elementoB.appendChild(elementoBR);
elementoB.appendChild(texto3);
```

// Añadimos al elemento P los nodos de texto1, elementoB y texto4.

```
elementoP.appendChild(texto1);
elementoP.appendChild(elementoB);
elementoP.appendChild(texto4);
```

// Insertamos el nuevo párrafo como un nuevo hijo de nuestro párrafo.

```
document.getElementById('parrafito').appendChild(elementoP);
```

```
var nodelist = elementNodeReference.childNodes;
```

```
const parg = document.createElement('p');
```

// Primero comprueba que el elemento tiene nodos hijos.

```
if (parg.hasChildNodes()) {
    var hijo = parg.childNodes;

    for (var i = 0; i < hijo.length; i++) {
        // se hace algo con cada hijo como hijo[i]
        // NOTA: Añadir o eliminar hijos cambiará la lista automáticamente.
    }
}
```

// Esto es una forma de eliminar todos los hijos de un nodo.

// box hace referencia a un elemento.

```
while(box.firstChild) {
    //La lista se reindexará automáticamente

    box.removeChild(box.firstChild);
}
```

Gestión de atributos

El DOM cuenta con métodos para poder gestionar los atributos de cada nodo:

Método	Descripción	Ejemplo
setAttribute(nombre, valor)	Establece el valor de un atributo a un nodo	element.setAttribute('id', 'ejemplo')
getAttribute(nombre)	Recupera el valor del atributo indicado como parámetro	element.getAttribute('id') // devuelve 'ejemplo'
hasAttribute(nombre)	Devuelve verdadero o falso dependiendo de si el nodo tiene el atributo indicado	element.hasAttribute('class') // devuelve false
removeAttribute(nombre)	Elimina el atributo indicado por parámetro del nodo	element.removeAttribute('id')

Atributos de clase

En JavaScript, el Modelo de objetos de documento (DOM) permite acceder y manipular elementos en un documento HTML o XML. El DOM se representa como una estructura en forma de árbol, donde cada nodo representa un elemento o atributo en el documento.

Aquí hay algunos ejemplos de cómo puede usar atributos de clase para acceder y manipular elementos en el DOM usando JavaScript:

1. Obtenga todos los elementos con una clase específica:

```
var elements = document.getElementsByClassName("my-class");
```

2. Añadir una clase a un elemento:

```
var element = document.getElementById("my-element");
element.classList.add("new-class");
```

3. Eliminar una clase de un elemento:

```
var element = document.getElementById("my-element");
element.classList.remove("old-class");
```

4. Comprobar si un elemento tiene una clase específica:

```
var element = document.getElementById("my-element");
if (element.classList.contains("my-class")) {
    // Do something
}
```


5. Alternar una clase en un elemento:

```
var element = document.getElementById("my-element");
element.classList.toggle("highlight");
```

6. Reemplazar una clase por otra:

```
var element = document.getElementById("my-element");
element.classList.replace("old-class", "new-class");
```

7. Acceda al atributo classList:

```
var element = document.getElementById("my-element");
var classList = element.classList;
```

Estos son solo algunos ejemplos de cómo puede usar atributos de clase para acceder y manipular elementos en el DOM usando JavaScript. El DOM proporciona muchos otros métodos y propiedades que puede utilizar para recorrer, acceder y modificar los elementos de un documento.

- [Modificación de clases.](#)

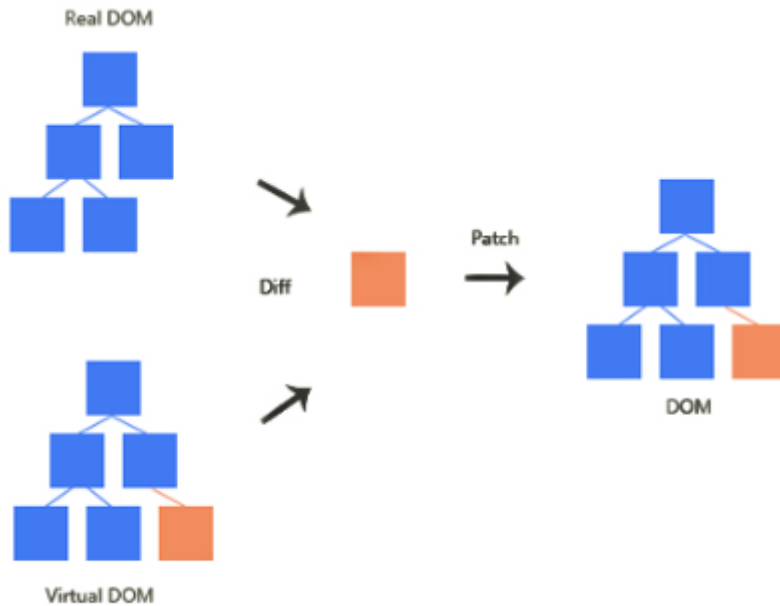
Método/Propiedad	Descripción	Ejemplo
className	Obtiene o establece un valor de clase.	element.className;
classList.add()	Agrega uno o más valores de clase.	element.classList.add('active');
classList.toggle()	Activa o desactiva una clase.	element.classList.toggle('active');
classList.contains()	Comprueba si el valor de clase existe.	element.classList.contains('active');
classList.replace()	Sustituye un valor de clase existente por uno nuevo.	element.classList.replace('old', 'new');
classList.remove()	Elimina un valor de clase.	element.classList.remove('active');

- [Explicación más detallada y ejemplos.](#)

Extras

DOM Virtual

El DOM virtual es una copia idéntica del DOM real que implementan varios frameworks de JavaScript como puede ser React. Su propósito es comparar los cambios que se produzcan en el DOM real para encontrar la manera más eficiente de implementarlos.



Las ventajas de utilizar un DOM virtual a la par del DOM real son que, a la hora de renderizarse el DOM real, sólo se renderiza de nuevo los elementos del DOM real que hayan cambiado, en lugar de renderizar todos los elementos de nuevo. Esto mejora el rendimiento de carga de la página y hace que sea más fluida.

- [Vídeo explicativo](#)

Bibliografía

- [DOM 1](#)
- [DOM MDN 2](#)
- [DOM Lenguaje JS](#)
- [Introducción DOM MDN](#)
- [NODOS](#)
- [TIPOS DE NODOS 1](#)
- [TIPOS DE NODOS 2](#)
- [TIPOS DE NODOS 3](#)
- [TIPOS DE NODOS 4](#)
- [Tipos de nodos y propiedades de los nodos](#)
- [Acceso a nodo desde metodos del document](#)
- [Propiedades y metodos del dom](#)
- [Acceso a nodo utilizando selectores css](#)
- [Gestión de nodos referencia 1](#)
- [Gestión de nodos referencia 2](#)
- [Gestión de nodos referencia 3](#)
- [Gestión de nodos referencia 4](#)
- [Gestión de atributos](#)
- [Modificación de clases](#)
- [Explicación más detallada y ejemplos.](#)
- [dom-virtual](#)