# Contenido



Sobre todo si te encuentras en fase de aprendizaje, lo normal suele ser crear código HTML desde un fichero HTML. Sin embargo, y sobre todo con el auge de las páginas **SPA** (*Single Page Application**) y los frameworks Javascript, esto ha cambiado bastante y es bastante frecuente **crear código HTML desde Javascript** de forma dinámica.

Esto tiene sus ventajas y sus desventajas. Un fichero **.html** siempre será más sencillo, más «estático» y más directo, ya que lo primero que analiza un navegador web es un fichero de marcado HTML. Por otro lado, un fichero **.js** es más complejo y menos directo, pero mucho más potente, «dinámico» y flexible, con menos limitaciones.

En este artículo vamos a ver como podemos **crear elementos HTML desde Javascript** y aprovecharnos de la potencia de Javascript para hacer cosas que desde HTML, sin ayuda de Javascript, no podríamos realizar o costaría mucho más.

# 1 CREAR ELEMENTOS HTML

Existen una serie de métodos para **crear de forma eficiente** diferentes elementos HTML o nodos, y que nos pueden convertir en una tarea muy sencilla el crear estructuras dinámicas, mediante bucles o estructuras definidas:

| Métodos | Descripción |
|---|---|
| **.createElement(tag, options)** | Crea y devuelve el elemento HTML definido por el **STRING** **tag**. |
| **.createComment(text)** | Crea y devuelve un **nodo** de comentarios HTML **<!-- text -->**. |
| **.createTextNode(text)** | Crea y devuelve un **nodo** HTML con el texto **text**. |
| **.cloneNode(deep)** | Clona el nodo HTML y devuelve una copia. **deep** es **false** por defecto. |
| **.isConnected** | Indica si el nodo HTML está insertado en el documento HTML. |

Para empezar, nos centraremos principalmente en la primera, que es la que utilizamos para **crear elementos HTML** en el DOM.

### 1.1 EL MÉTODO createElement()

Mediante el método **.createElement()** podemos crear un **ELEMENT** HTML **en memoria** (*¡no estará insertado aún en nuestro documento HTML!*). Con dicho elemento almacenado en una variable, podremos modificar sus características o contenido, para **posteriormente** insertarlo en una posición determinada del DOM o documento HTML.

Vamos a centrarnos en el proceso de **creación del elemento**, y en el próximo capítulo veremos el apartado de insertarlo en el DOM. El funcionamiento de **.createElement()** es muy sencillo: se trata de pasarle el nombre de la etiqueta **tag** a utilizar.

```
const div = document.createElement("div");                  // Creamos un <div></div>
const span = document.createElement("span");            // Creamos un <span></span>
const img = document.createElement("img");     // Creamos un <img>
```

De la misma forma, podemos crear comentarios HTML con **createComment()** o nodos de texto sin etiqueta HTML con **createTextNode()**, pasándole a ambos un **STRING** con el texto en cuestión. En ambos, se devuelve un **NODE** que podremos utilizar luego para insertar en el documento HTML:

```
const comment = document.createComment("Comentario");   //   <!--Comentario-->
const text = document.createTextNode("Hola");       // Nodo de texto: 'hola'
```

El método **createElement()** tiene un parámetro opcional denominado **options**. Si se indica, será un objeto con una propiedad **is** para definir un **elemento personalizado** en una modalidad menos utilizada. Se verá más adelante en el apartado de **Web Components**.

Ten presente que en los ejemplos que hemos visto estamos creando los elementos en una constante, pero de momento **no están añadiéndose al documento HTML**, por lo que no aparecerían visualmente. Más adelante veremos como añadirlos.

## 2 EL MÉTODO cloneNode()

Hay que tener mucho cuidado al crear y **duplicar** elementos HTML. Un error muy común es asignar un elemento que tenemos en otra variable, pensando que estamos creando una copia (*cuando no es así*):

```
const div = document.createElement("div");
div.textContent = "Elemento 1";

const div2 = div;  // NO se está haciendo una copia
div2.textContent = "Elemento 2";
```

```
div.textContent;  // 'Elemento 2'
```

Con esto, quizás pueda parecer que estamos duplicando un elemento para crearlo a imagen y semejanza del original. Sin embargo, lo que se hace es una **referencia** al elemento original, de modo que si se modifica el **div2**, también se modifica el elemento original. Para evitar esto, lo ideal es utilizar el método **cloneNode()**:

```
const div = document.createElement("div");
div.textContent = "Elemento 1";

const div2 = div.cloneNode();   // Ahora SÍ estamos clonando
div2.textContent = "Elemento 2";

div.textContent;  // 'Elemento 1'
```

El método **cloneNode(deep)** acepta un parámetro **BOOLEAN deep** opcional, a **false** por defecto, para indicar el tipo de clonación que se realizará:

- Si es **true**, clonará también sus hijos, conocido como una **clonación profunda** (*Deep Clone*).
- Si es **false**, no clonará sus hijos, conocido como una **clonación superficial** (*Shallow Clone*).

## 3 LA PROPIEDAD isConnected

Por último, la propiedad **isConnected** nos indica si el nodo en cuestión está conectado al DOM, es decir, si está insertado en el documento HTML:

- Si es **true**, significa que el elemento está conectado al DOM.
- Si es **false**, significa que el elemento no está conectado al DOM.

Hasta ahora, hemos creado elementos que no lo están (*permanecen sólo en memoria*). En el capítulo Insertar elementos en el DOM veremos como insertarlos en el documento HTML para que aparezca visualmente en la página.

## 4 MANIPULACIÓN DE ATRIBUTOS

Hasta ahora, hemos visto como crear elementos HTML con Javascript, pero no hemos visto como modificar los atributos HTML de dichas etiquetas creadas. En general, una vez tenemos un elemento sobre el que vamos a crear algunos atributos, lo más sencillo es **asignarle valores como propiedades** de objetos:

```
const div = document.createElement("div"); // <div></div>
div.id = "page";        // <div id="page"></div>
```

```
div.className = "data";   // <div id="page" class="data"></div>
div.style = "color: red"; // <div id="page" class="data" style="color: red"></div>
```

Sin embargo, en algunos casos esto se puede complicar (*como se ve en uno de los casos del ejemplo anterior*). Por ejemplo, la palabra **class** (*para crear clases*) o la palabra **for** (*para bucles*) son palabras reservadas de Javascript y no se podrían utilizar para crear atributos. Por ejemplo, si queremos establecer una clase, se debe utilizar la propiedad **className**.

Es posible asignar a la propiedad **className** varias clases en un **STRING** separadas por espacio. De esta forma se asignarán múltiples clases. Aún así, recomendamos utilizar la propiedad **classList** que explicaremos más adelante en el capítulo manipulación de clases CSS.

Aunque la forma anterior es la más rápida, tenemos algunos métodos para utilizar en un elemento HTML y añadir, modificar o eliminar sus atributos:

| Métodos | Descripción |
|---|---|
| **hasAttributes()** | Indica si el elemento tiene atributos HTML. |
| **hasAttribute(attr)** | Indica si el elemento tiene el atributo HTML **attr**. |
| **getAttributeNames()** | Devuelve un **ARRAY** con los atributos del elemento. |
| **getAttribute(attr)** | Devuelve el valor del atributo **attr** del elemento o **NULL** si no existe. |
| **removeAttribute(attr)** | Elimina el atributo **attr** del elemento. |
| **setAttribute(attr, value)** | Añade o cambia el atributo **attr** al valor **value**. |
| **getAttributeNode(attr)** | Idem a **getAttribute()** pero devuelve el atributo como **nodo**. |
| **removeAttributeNode(attr)** | Idem a **removeAttribute()** pero devuelve el atributo como **nodo**. |
| **setAttributeNode(attr, value)** | Idem a **setAttribute()** pero devuelve el atributo como **nodo**. |

Estos métodos son bastante autoexplicativos y fáciles de entender, aún así, vamos a ver unos ejemplos de uso donde podemos ver como funcionan:

```
// Obtenemos <div id="page" class="info data dark" data-number="5"></div>
const div = document.querySelector("#page");

div.hasAttribute("data-number");  // true (data-number existe)
div.hasAttributes();          // true (tiene 3 atributos)

div.getAttributeNames();      // ["id", "data-number", "class"]
div.getAttribute("id");       // "page"
```

```
div.removeAttribute("id");      // class="info data dark" y data-number="5"
div.setAttribute("id", "page"); // Vuelve a añadir id="page"
```

Los tres últimos métodos mencionados: **getAttributeNode()**, **removeAttributeNode()** y **setAttributeNode()** son versiones idénticas a sus homónimos, sólo que devuelven el afectado, útil si queremos guardarlo en una variable y seguir trabajando con él.

Recuerda que hasta ahora hemos visto como crear elementos y cambiar sus atributos, pero **no los hemos insertado en el DOM** o documento HTML, por lo que no los veremos visualmente en la página. En el siguiente capítulo abordaremos ese tema.

## 5 USANDO FRAGMENTOS

En algunas ocasiones, nos puede resultar muy interesante utilizar **fragmentos**. Los fragmentos son una especie de documento paralelo, aislado de la página con la que estamos trabajando, que tiene varias características:

- No tiene elemento padre. Está aislado de la página o documento.
- Es mucho más simple y ligero (*mejor rendimiento*).
- Si necesitamos hacer cambios consecutivos, no afecta al **reflow** (*repintado de un documento*).

De esta forma, es una estrategia muy útil para usarlo de *documento temporal* y no realizar cambios consecutivos, con su impacto de rendimiento. Para crearlos, necesitaremos utilizar la siguiente función:

| Métodos | Descripción |
|---|---|
| **document.createDocumentFragment()** | Crea un fragmento aislado (*sin padre*). |

Así pues, el **OBJECT** que devuelve el método **document.createDocumentFragment()** es un **fragmento** que podremos utilizar para almacenar en su interior un pequeño DOM temporal, que luego añadiremos en nuestro DOM principal.

```
const fragment = document.createDocumentFragment();

for (let i = 0; i < 5000; i++) {
  const div = document.createElement("div");
  div.textContent = `Item número ${i}`;
  fragment.appendChild(div);
}

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

Como se puede ver, utilizamos el fragmento **fragment** generado como ubicación temporal donde hacer todos los cambios del DOM que necesitemos, sin que afecten al **reflow** del documento de forma independiente. Una vez terminemos nuestra lógica y tengamos el DOM definitivo, lo insertamos como hacemos siempre, por ejemplo, con un **appendChild** (*ver más adelante*).

Es entonces cuando se traslada todo el DOM del fragmento al lugar donde hemos indicado en el **appendChild** (*en nuestro ejemplo, a la etiqueta **<body>**), dejando nuevamente el fragmento vacío.

Los elementos **<template>** utilizan **fragmentos** para crear un DOM desconectado del documento principal. Puedes ver más en Plantillas y DOM en WebComponents.