Contenido

INSERTAR ELEMENTOS EN EL DOM

En el capítulo anterior hemos visto como crear elementos en el DOM, pero dichos elementos **se creaban en memoria** y los almacenábamos en una variable o constante. No se conectaban al DOM o documento HTML de forma automática, sino que debemos hacerlo manualmente, que es justo lo que veremos en este artículo: como insertar elementos en el DOM, así como eliminarlos.

En este artículo vamos a centrarnos en tres categorías:

- **Reemplazar** contenido de elementos en el DOM
- **Insertar** elementos en el DOM
- **Eliminar** elementos del DOM

Reemplazar contenido

Comenzaremos por la familia de propiedades siguientes, que enmarcamos dentro de la categoría de **reemplazar contenido** de elementos HTML. Se trata de una vía rápida con la cuál podemos añadir (*o más bien, reemplazar*) el contenido de una etiqueta HTML.

Las propiedades son las siguientes:

Propiedades	Descripción
STRING **.nodeName**	Devuelve el nombre del nodo (etiqueta si es un elemento HTML). Sólo lectura.
STRING **.textContent**	Devuelve el contenido de texto del elemento. Se puede asignar para modificar.
STRING **.innerHTML**	Devuelve el contenido HTML del elemento. Se puede usar asignar para modificar.

Propiedades	Descripción
STRING **.outerHTML**	Idem a **.innerHTML** pero incluyendo el HTML del propio elemento HTML.

La propiedad **nodeName** nos devuelve el nombre del todo, que en elementos HTML es interesante puesto que nos devuelve el nombre de la etiqueta **en mayúsculas**. Se trata de una propiedad de sólo lectura, por lo cuál no podemos modificarla, sólo acceder a ella.

La propiedad textContent

La propiedad **.textContent** nos devuelve el **contenido de texto** de un elemento HTML. Es útil para obtener (*o modificar*) **sólo el texto** dentro de un elemento, obviando el etiquetado HTML:

```
const div = document.querySelector("div");  // <div></div>

div.textContent = "Hola a todos"; // <div>Hola a todos</div>
div.textContent;  // "Hola a todos"
```

Observa que también podemos utilizarlo para **reemplazar el contenido de texto**, asignándolo como si fuera una variable o constante. En el caso de que el elemento tenga anidadas varias etiquetas HTML una dentro de otra, la propiedad **.textContent** se quedará sólo con el contenido textual completo, como se puede ver en el siguiente ejemplo:

```
// Obtenemos <div class="info">Hola <strong>amigos</strong></div>
const div = document.querySelector(".info");

div.textContent;  // "Hola amigos"
```

La propiedad innerHTML

Por otro lado, la propiedad **.innerHTML** nos permite hacer lo mismo, pero interpretando el código HTML indicado y renderizando sus elementos:

```
const div = document.querySelector(".info"); // <div class="info"></div>

div.innerHTML = "<strong>Importante</strong>"; // Interpreta el HTML
div.innerHTML;   // "<strong>Importante</strong>"
div.textContent;  // "Importante"

div.textContent = "<strong>Importante</strong>";  // No interpreta el HTML
```

Observa que la diferencia principal entre **.innerHTML** y **.textContent** es que el primero renderiza e interpreta el marcado HTML, mientras que el segundo lo inserta como contenido de texto literalmente.

Ten en cuenta que la propiedad **.innerHTML** comprueba y parsea el marcado HTML escrito (*corrigiendo si hay errores*) antes de realizar la asignación. Por ejemplo, si en el ejemplo anterior nos olvidamos de escribir el cierre **** de la etiqueta, **.innerHTML** automáticamente lo cerrará. Esto puede provocar algunas incongruencias si el código es incorrecto o una disminución de rendimiento en textos muy grandes que hay que preprocesar.

Por otro lado, la propiedad **.outerHTML** es muy similar a **.innerHTML**. Mientras que esta última devuelve el código HTML del interior de un elemento HTML, **.outerHTML** devuelve también el código HTML del propio elemento en cuestión. Esto puede ser muy útil para reemplazar un elemento HTML combinándolo con **.innerHTML**:

```
const data = document.querySelector(".data");
data.innerHTML = "<h1>Tema 1</h1>";

data.textContent;   // "Tema 1"
data.innerHTML;    // "<h1>Tema 1</h1>"
data.outerHTML;    // "<div class="data"><h1>Tema 1</h1></div>"
```

En este ejemplo se pueden observar las diferencias entre las propiedades **.textContent** (*contenido de texto*), **.innerHTML** (*contenido HTML*) y **.outerHTML** (*contenido y contenedor HTML*).

Insertar elementos

A pesar de que los métodos anteriores son suficientes para crear elementos y estructuras HTML complejas, sólo son aconsejables para pequeños fragmentos de

código o texto, ya que en estructuras muy complejas (*con muchos elementos HTML*) la **legibilidad del código** sería menor y además, el rendimiento podría resentirse.

Hemos aprendido a crear elementos HTML y sus atributos, pero aún no hemos visto como añadirlos al documento HTML actual (*conectarlos al DOM*), operación que se puede realizar de diferentes formas mediante los siguientes métodos disponibles:

Métodos	Descripción
NODE **.appendChild(node)**	Añade como hijo el nodo **node**. Devuelve el nodo insertado.
ELEMENT **.insertAdjacentElement(pos, elem)**	Inserta el elemento **elem** en la posición **pos**. Si falla, NULL.
UNDEFINED **.insertAdjacentHTML(pos, str)**	Inserta el código HTML **str** en la posición **pos**.
UNDEFINED **.insertAdjacentText(pos, text)**	Inserta el texto **text** en la posición **pos**.
NODE **.insertBefore(new, node)**	Inserta el nodo **new** antes de **node** y como hijo del nodo actual.

De ellos, probablemente el más extendido es **.appendChild()**, no obstante, la familia de métodos **.insertAdjacent*()** también tiene buen soporte en navegadores y puede usarse de forma segura en la actualidad.

El método appendChild()

Uno de los métodos más comunes para añadir un elemento HTML creado con Javascript es **appendChild()**. Como su propio nombre indica, este método realiza un **«append»**, es decir, inserta el elemento como un hijo al final de todos los elementos hijos que existan.

Es importante tener clara esta particularidad, porque aunque es lo más común, no siempre querremos insertar el elemento en esa posición:

```
const img = document.createElement("img");
img.src = "https://lenguajejs.com/assets/logo.svg";
img.alt = "Logo Javascript";

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

En este ejemplo podemos ver como creamos un elemento **<img>** que aún no está conectado al DOM. Posteriormente, añadimos los atributos **src** y **alt**, obligatorios en una etiqueta de imagen. Por último, conectamos al DOM el elemento, utilizando el método **.appendChild()** sobre **document.body** que no es más que una referencia a la etiqueta **<body>** del documento HTML.

Veamos otro ejemplo:

```
const div = document.createElement("div");
div.textContent = "Esto es un div insertado con JS.";

const app = document.createElement("div"); // <div></div>
app.id = "app";      // <div id="app"></div>
app.appendChild(div); // <div id="app"><div>Esto es un div insertado con JS</div></div>
```

En este ejemplo, estamos creando dos elementos, e insertando uno dentro de otro. Sin embargo, a diferencia del anterior, el elemento **app** no está conectado aún al DOM, sino que lo tenemos aislado en esa variable, sin insertar en el documento. Esto ocurre porque **app** lo acabamos de crear, y en el ejemplo anterior usabamos **document.body** que es una referencia a un elemento que ya existe en el documento.

Los métodos insertAdjacent*()

Los métodos de la familia **insertAdjacent** son bastante más versátiles que **.appendChild()**, ya que permiten muchas más posibilidades. Tenemos tres versiones diferentes:

- **.insertAdjacentElement()** donde insertamos un objeto ELEMENT
- **.insertAdjacentHTML()** donde insertamos **código HTML** directamente (*similar a innerHTML*)
- **.insertAdjacentText()** donde no insertamos elementos HTML, sino un NODE con texto

En las tres versiones, debemos indicar por parámetro un STRING **pos** como primer parámetro para indicar en que posición vamos a insertar el contenido. Hay 4 opciones posibles:

- **beforebegin**: El elemento se inserta **antes** de la etiqueta HTML de apertura.
- **afterbegin**: El elemento se inserta **dentro** de la etiqueta HTML, **antes de su primer hijo**.
- **beforeend**: El elemento se inserta **dentro** de la etiqueta HTML, **después de su último hijo**. Es el equivalente a usar el método **.appendChild()**.
- **afterend**: El elemento se inserta **después** de la etiqueta HTML de cierre.

Veamos algunos ejemplo aplicando cada uno de ellos con el método **.insertAdjacentElement()**:

```
const div = document.createElement("div");  // <div></div>
div.textContent = "Ejemplo";                // <div>Ejemplo</div>

const app = document.querySelector("#app"); // <div id="app">App</div>

app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentElement("afterbegin", div);
// Opción 2: <div id="app"> <div>Ejemplo</div> App</div>

app.insertAdjacentElement("beforeend", div);
// Opción 3: <div id="app">App <div>Ejemplo</div> </div>

app.insertAdjacentElement("afterend", div);
// Opción 4: <div id="app">App</div> <div>Ejemplo</div>
```

Ten en cuenta que en el ejemplo muestro **varias opciones alternativas**, no lo que ocurriría tras ejecutar las cuatro opciones una detrás de otra.

Por otro lado, notar que tenemos **tres versiones** en esta familia de métodos, una que actúa sobre elementos HTML (*la que hemos visto*), pero otras dos que actúan sobre código HTML y sobre nodos de texto. Veamos un ejemplo de cada una:

```
app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentHTML("beforebegin", '<p>Hola</p>');
// Opción 2: <p>Hola</p> <div id="app">App</div>

app.insertAdjacentText("beforebegin", "Hola a todos");
// Opción 3: Hola a todos <div id="app">App</div>
```

El método insertBefore()

Por último, el método **insertBefore(newnode, node)** es un método más específico y menos utilizado en el que se puede especificar exactamente el lugar a insertar un nodo. El parámetro **newnode** es el nodo a insertar, mientras que **node** puede ser:

- NULL; insertando **newnode** después del último nodo hijo. Equivalente a **.appendChild()**.
- NODE o ELEMENT; insertando **newnode** antes de dicho **node** de referencia.

Eliminar elementos

Al igual que podemos insertar o reemplazar elementos, también podemos eliminarlos. Ten en cuenta que al «eliminar» un nodo o elemento HTML, lo que hacemos realmente no es borrarlo, sino **desconectarlo del DOM o documento HTML**, de modo que no están conectados, pero siguen existiendo.

El método remove()

Probablemente, la forma más sencilla de eliminar nodos o elementos HTML es utilizando el método **.remove()** sobre el nodo o etiqueta a eliminar:

```
const div = document.querySelector(".deleteme");

div.isConnected;   // true
div.remove();
div.isConnected;   // false
```

En este caso, lo que hemos hecho es buscar el elemento HTML ** ** en el documento HTML y desconectarlo de su elemento padre, de forma que dicho elemento pasa a no pertenecer al documento HTML.

Sin embargo, existen algunos métodos más para eliminar o reemplazar elementos:

Métodos	Descripción
UNDEFINED **.remove()**	Elimina el propio nodo de su elemento padre.
NODE **.removeChild(node)**	Elimina y devuelve el nodo hijo **node**.
NODE **.replaceChild(new, old)**	Reemplaza el nodo hijo **old** por **new**. Devuelve **old**.

El método **.remove()** se encarga de desconectarse del DOM a sí mismo, mientras que el segundo método, **.removeChild()**, desconecta el nodo o elemento HTML proporcionado. Por último, con el método **.replaceChild()** se nos permite cambiar un nodo por otro.

El método removeChild()

En algunos casos, nos puede interesar eliminar un nodo hijo de un elemento. Para esas situaciones, podemos utilizar el método **.removeChild(node)** donde **node** es el nodo hijo que queremos eliminar:

```
const div = document.querySelector(".item:nth-child(2)");   // <div class="item">2</div>

document.body.removeChild(div); // Desconecta el segundo .item
```

El método replaceChild()

De la misma forma, el método **replaceChild(new, old)** nos permite cambiar un nodo hijo **old** por un nuevo nodo hijo **new**. En ambos casos, el método nos devuelve el nodo reemplazado:

```
const div = document.querySelector(".item:nth-child(2)");

const newnode = document.createElement("div");
newnode.textContent = "DOS";

document.body.replaceChild(newnode, div);
```