3.1. DOM y eventos

LENGUAJES DE PROGRAMACIÓN NIVEL 5. UNIDAD 3

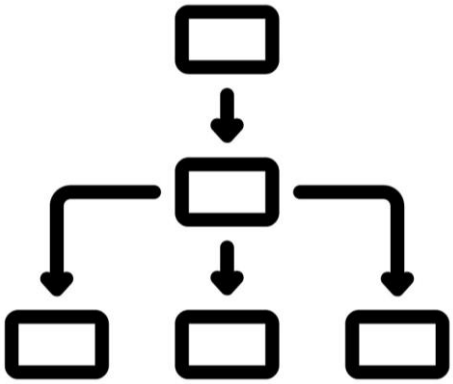
JAVASCRIPT - DESAROLLO FRONTEND AVANZADO

Financiado por la Unión Europea

Plan de Recuperación, Transformación y Resiliencia

Contenido

DOM



El Document Object Model (DOM) es una representación estructurada y jerárquica de un documento HTML o XML que proporciona una interfaz programática para interactuar con los elementos de una página web.

En esencia, el DOM convierte cada elemento HTML en un objeto que puede ser manipulado y modificado mediante código JavaScript.

Definición y conceptos básicos

El DOM consta de un conjunto de nodos que representan cada elemento del documento, como etiquetas, atributos y texto. Estos nodos forman una estructura de árbol donde cada elemento es un nodo y puede tener nodos secundarios (hijos) y nodos padres (ancestros).

Elementos que componen el DOM

El DOM (Document Object Model) está compuesto por una serie de elementos que representan la estructura de un documento HTML o XML.

Cada uno de esos elementos es un nodo que compone esa estructura.

Estos elementos incluyen:

- **Document:** Es el nodo raíz del árbol DOM y representa todo el documento HTML. Es el punto de entrada para acceder y manipular el resto de los nodos del documento.
- **Element:** Representa cada etiqueta HTML en el documento, como , <p>, , etc. Los elementos pueden contener otros elementos, atributos y texto.

- **Atributo:** Son los atributos de los elementos HTML que definen sus características o comportamiento, como id, class, src, href, etc.
- **Texto:** Representa el contenido de texto dentro de un elemento HTML, como el texto dentro de un <p> o un .
- **Comentario:** Son comentarios dentro del código HTML que no se muestran en la página pero están presentes en el DOM.

Estos son los elementos principales que componen un DOM. Cada uno de ellos forma parte de la estructura jerárquica del árbol DOM, que es una representación visual de la estructura del documento HTML. Los desarrolladores web utilizan el DOM para acceder y manipular estos elementos mediante JavaScript, lo que permite crear interactividad y dinamismo en las páginas web.

Relaciones entre elementos

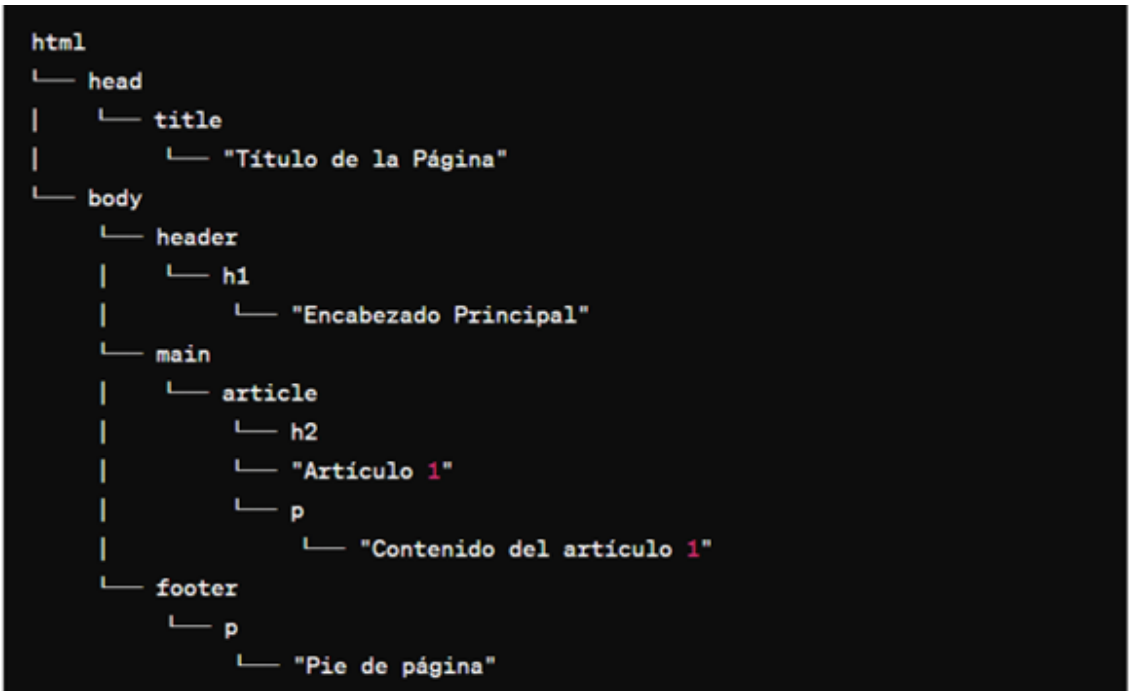
En el DOM (Document Object Model), los elementos están organizados en una estructura jerárquica de árbol, lo que significa que cada elemento tiene una relación específica con otros elementos en el documento.

Las principales relaciones comunes entre los elementos del DOM incluyen:

- **Padre e Hijo:** Un elemento puede ser el padre de otro elemento si contiene directamente a ese elemento dentro de su estructura. El elemento contenido se denomina hijo y el elemento que lo contiene se denomina padre.
- **Hermano:** Dos elementos que tienen el mismo padre se consideran hermanos. Estos elementos son hijos del mismo elemento padre y comparten el mismo nivel de jerarquía en el árbol DOM.
- **Descendiente y Ancestro:** Un elemento es descendiente de otro elemento si está contenido dentro de él, ya sea directa o indirectamente. Por otro lado, un elemento es un ancestro de otro si contiene directa o indirectamente a ese elemento.
- **Elemento Raíz:** En el árbol DOM, hay un solo elemento que actúa como el nodo raíz del documento completo. Este elemento, comúnmente llamado <html>, contiene todos los demás elementos en la página.

Ejemplo

Este es un ejemplo sencillo de una estructura de árbol DOM:



1

En este ejemplo:

- El elemento <html> es el nodo raíz del árbol DOM.
- Tiene dos hijos principales: <head> y <body>.
- El <head> contiene un <title> como hijo.

---

[1] Accesibilidad
html
head
title
“Título de página”
body
header
h1
“Encabezado principal
main
article
h2
“Articulo 1”
p
“Contenido del Articulo 1”
footer
p
“Pie de página”

- El <body> contiene tres elementos principales: <header>, <main> y <footer>.
- <header> contiene un encabezado (<h1>).
- <main> contiene un artículo (<article>) con un título (<h2>) y un párrafo (<p>).
- <footer> contiene un párrafo (<p>).

Esta estructura representa una página web simple con un encabezado, un área de contenido principal y un pie de página. Cada elemento está anidado dentro de otros elementos según su relación jerárquica en el árbol DOM.

Importancia del DOM en el desarrollo web

El DOM es fundamental en el desarrollo web porque permite la interacción dinámica con los elementos de una página. Algunas de las razones por las que el DOM es importante son:

- **Interactividad:** Permite a los desarrolladores crear páginas web dinámicas y interactivas mediante la manipulación de elementos y la respuesta a eventos del usuario.
- **Accesibilidad:** Facilita el acceso y la modificación de los elementos de una página, lo que es crucial para la creación de aplicaciones web accesibles.
- **Compatibilidad:** Proporciona una interfaz estándar para interactuar con los elementos de una página, lo que garantiza la compatibilidad entre diferentes navegadores y plataformas.

En resumen, el DOM es una parte esencial del desarrollo web moderno, ya que permite a los desarrolladores crear experiencias interactivas y dinámicas para los usuarios, mejorando la usabilidad y la accesibilidad de las aplicaciones web.

Operaciones con el DOM

Con el DOM (Document Object Model), puedes realizar una variedad de operaciones para interactuar con la estructura y el contenido de un documento HTML.

Algunas de las operaciones más comunes que se pueden realizar con el DOM incluyen:

- **Acceder a elementos:** Puedes acceder a elementos específicos del documento HTML utilizando métodos como getElementById, getElementsByClassName, getElementsByTagName, querySelector y querySelectorAll.
- **Modificar contenido:** Puedes cambiar el contenido de los elementos HTML, como el texto dentro de un párrafo o el valor de un atributo. Esto se puede hacer utilizando propiedades como innerHTML, textContent y métodos como setAttribute.
- **Añadir y eliminar elementos:** Puedes crear nuevos elementos HTML dinámicamente y añadirlos al documento utilizando el método createElement, y luego añadiendo estos elementos al DOM con métodos como appendChild o insertBefore. También puedes eliminar elementos existentes utilizando el método removeChild.
- **Modificar estilos CSS:** Puedes cambiar los estilos CSS de los elementos, como su color, tamaño o posición, utilizando propiedades como style.
- **Manejar eventos:** Puedes añadir event listeners a los elementos para detectar y responder a eventos del usuario, como clics de ratón, pulsaciones de teclas, cambios de entrada, etc. Esto se puede hacer utilizando métodos como addEventListener.
- **Traversing DOM:** Puedes navegar por la estructura del DOM para acceder a nodos secundarios, padres, hermanos, hijos, etc. Esto se puede hacer utilizando propiedades como parentNode, childNodes, nextSibling, previousSibling, firstChild y lastChild.
- **Manipulación de clases:** Puedes añadir, eliminar y cambiar clases CSS en elementos utilizando métodos como classList.add, classList.remove y classList.toggle.
- **Crear y manipular atributos:** Puedes añadir, modificar y eliminar atributos de elementos utilizando métodos como setAttribute, getAttribute y removeAttribute.

Casos de uso típicos

Las operaciones con el DOM son fundamentales en el desarrollo web y se utilizan en una amplia variedad de casos. Algunos de los casos de uso típicos incluyen:

- **Actualización dinámica de contenido:** Cambiar el contenido de una página web en respuesta a acciones del usuario o eventos específicos, como cargar nuevos datos desde un servidor, mostrar mensajes de error o actualizar listas de productos.
- **Validación de formularios:** Validar los datos introducidos por el usuario en formularios HTML para garantizar que cumplan con ciertos criterios antes de ser enviados al servidor, como verificar que un campo de correo electrónico tenga un formato válido o que una contraseña cumpla con ciertas reglas de complejidad.
- **Creación de elementos de interfaz de usuario (UI):** Generar y añadir nuevos elementos HTML al DOM para mostrar mensajes emergentes, menús desplegables, ventanas modales, pestañas de navegación u otros componentes de interfaz de usuario dinámicos.
- **Manipulación de estilos y diseño:** Cambiar los estilos CSS de elementos HTML para ajustar el diseño y la apariencia de la página web en respuesta a diferentes estados o eventos, como cambiar el color de fondo de un botón al pasar el cursor sobre él o ajustar el tamaño de un elemento en función del tamaño de la pantalla.
- **Navegación y manipulación de elementos del documento:** Permitir a los usuarios interactuar con la estructura del documento, como hacer clic en enlaces para navegar entre páginas, expandir o contraer secciones de contenido, o arrastrar y soltar elementos para reordenarlos.
- **Gestión de eventos:** Asociar eventos del usuario, como clics de ratón, pulsaciones de teclas o cambios de entrada, con funciones de JavaScript para realizar acciones específicas, como enviar datos al servidor, abrir ventanas emergentes o mostrar mensajes de confirmación.
- **Optimización del rendimiento:** Minimizar las operaciones costosas en el DOM, como acceder repetidamente a elementos o modificar el contenido del DOM en bucles, para mejorar el rendimiento y la capacidad de respuesta de la aplicación web.

Eventos en JavaScript

Los eventos en JavaScript son acciones que ocurren en el navegador del usuario, como hacer clic en un elemento, mover el ratón sobre un elemento, pulsar una tecla en el teclado, cargar una página, etc.

Estos eventos pueden ser detectados por JavaScript y utilizados para desencadenar funciones o ejecutar código en respuesta a las acciones del usuario o cambios en la página.

¿Qué son los eventos?

Los eventos son interacciones del usuario o cambios en el estado de la página web que el navegador detecta y notifica a través de JavaScript. Estos eventos pueden ser de diferentes tipos, como eventos de ratón, eventos de teclado, eventos de formulario, eventos de carga de página, etc.

Tipos de eventos más comunes

Algunos de los tipos de eventos más comunes en JavaScript incluyen:

- **Eventos de ratón:** Ocurren cuando el usuario interactúa con el ratón, como hacer clic en un elemento (click), mover el ratón sobre un elemento (mouseover), sacar el ratón de un elemento (mouseout), etc.
- **Eventos de teclado:** Ocurren cuando el usuario interactúa con el teclado, como pulsar una tecla (keydown), soltar una tecla (keyup), o mantener pulsada una tecla (keypress).

- **Eventos de formulario:** Ocurren cuando el usuario interactúa con elementos de formulario, como enviar un formulario (submit), cambiar el valor de un campo de entrada (change), o enfocar un campo de entrada (focus).
- **Eventos de carga de página:** Ocurren cuando se carga la página web, como cuando se carga completamente la página (load), cuando se carga un recurso externo como una imagen o un script (load), o cuando se inicia la descarga de un recurso (beforeunload).

Eventos y DOM

Los eventos están estrechamente relacionados con el DOM, ya que la mayoría de los eventos están asociados con elementos específicos del DOM. Cuando ocurre un evento en un elemento, se propaga a través de la jerarquía del DOM, lo que significa que los eventos pueden ser capturados y manejados por diferentes elementos en la página en función de su relación con el elemento objetivo.

Estos conceptos básicos de eventos en JavaScript son fundamentales para comprender cómo interactuar dinámicamente con una página web y responder a las acciones del usuario de manera efectiva.

Manejadores de eventos

Los manejadores de eventos en JavaScript son funciones que se utilizan para manejar eventos específicos que ocurren en un elemento del DOM o en la ventana del navegador.

Los manejadores de eventos establecen una relación entre elementos específicos del DOM y comportamientos particulares asociados a eventos específicos. .Estos eventos pueden ser acciones del usuario, como hacer clic en un botón, mover el ratón sobre un elemento, escribir en un campo de texto, etc.

El proceso implica:

1. **Identificar el elemento del DOM:**
   Seleccionar el elemento al que se le desea asignar un comportamiento. Esto puede hacerse mediante métodos como getElementById, querySelector, etc.

2. **Definir el comportamiento:**
   Crear una función o método que describa qué hacer cuando ocurra el evento en ese elemento. Esta función se conoce como manejador de eventos.
3. **Asignar el manejador de eventos al elemento:**
   Utilizando métodos como addEventListener, onclick, etc., se vincula el evento específico con la función que se ejecutará cuando ocurra dicho evento en el elemento.
4. **Responder al evento:**
   Cuando el evento ocurre en el elemento correspondiente, se ejecuta la función asociada (el manejador de eventos), lo que permite que la aplicación responda de manera apropiada al evento del usuario.

Este proceso permite crear interactividad en la aplicación web, ya que define cómo se debe comportar la aplicación cuando los usuarios interactúan con ella a través de su navegador.

Asignación de manejadores de eventos

La asignación de manejadores de eventos en JavaScript es el proceso de vincular una función o código específico a un evento particular en un elemento del DOM. Esto se hace utilizando métodos proporcionados por el objeto addEventListener o mediante propiedades específicas de los elementos HTML.

Aquí hay un ejemplo de cómo se puede asignar un manejador de eventos utilizando addEventListener:

```
// Obtener una referencia al elemento HTML
const button = document.getElementById('myButton');

// Definir la función que manejará el evento
function handleClick(event) {
    console.log('El botón fue clickeado!');
}

// Asignar el manejador de eventos al botón
button.addEventListener('click', handleClick);
```

[2]

En este ejemplo, la función handleClick se ejecutará cada vez que se haga clic en el botón con el id 'myButton'.

También es posible asignar manejadores de eventos directamente a través de propiedades de los elementos HTML, como onclick, onmouseover, onchange, entre otros. Sin embargo, el uso de addEventListener es más flexible y permite adjuntar múltiples manejadores de eventos al mismo elemento, lo que facilita la organización y mantenimiento del código.

---

[2] Accesibilidad

```
// Obtener una referencia al elemento html
const button = document .getElementById("my Button);
// Definir la función que manejará el evento
function  handleClick(event) {
        console.log('El botón fue clickeado');
}
// Asigna r el manejador de eventos al botón
button.addEventListener("click", handleClick);
```

Mejores prácticas y consideraciones

Aquí hay algunas mejores prácticas y consideraciones a tener en cuenta al trabajar con eventos en JavaScript:

- **Organización del código:** Es importante mantener el código bien estructurado y organizado. Agrupa los manejadores de eventos relacionados en funciones separadas y utiliza un enfoque modular para mantener el código mantenible y fácil de entender.
- **Uso adecuado de eventos:** Evita asignar demasiados eventos a un mismo elemento, ya que esto puede hacer que el código sea difícil de seguir y mantener. En su lugar, considera utilizar delegación de eventos, donde un evento se maneja en un ancestro común de los elementos objetivo. Esto reduce la complejidad y mejora el rendimiento.
- **Evitar el acoplamiento excesivo:** Trata de mantener los manejadores de eventos lo más independientes posible del resto del código. Evita acoplar demasiado la lógica del evento con la estructura del DOM o con otras partes del código. Esto hace que el código sea más flexible y reutilizable.
- **Liberar recursos:** Recuerda siempre liberar los recursos asociados con los eventos cuando ya no sean necesarios para evitar posibles fugas de memoria. Por ejemplo, si has agregado un evento a un elemento y luego eliminas ese elemento del DOM, asegúrate de quitar también el evento para evitar problemas de rendimiento y memoria.

- **Pruebas unitarias:** Considera escribir pruebas unitarias para tus manejadores de eventos para garantizar su correcto funcionamiento y para facilitar la detección temprana de posibles problemas.

Al seguir estas mejores prácticas, podrás escribir código más limpio, mantenible y robusto al trabajar con eventos en JavaScript.

Conclusiones

En conclusión, el Document Object Model (DOM) y los eventos en JavaScript son elementos fundamentales en el desarrollo web moderno. Aquí hay una recapitulación de los conceptos clave y la importancia de estos aspectos:

- **DOM:** El DOM es una representación en memoria de la estructura de un documento HTML, que permite a los desarrolladores acceder, modificar y manipular los elementos y contenido de una página web dinámicamente. Es fundamental para la interacción y la manipulación de la interfaz de usuario en aplicaciones web.
- **Eventos:** Los eventos en JavaScript son acciones que ocurren en el navegador, como hacer clic en un elemento, desplazar el cursor del mouse sobre un elemento, o enviar un formulario. Los eventos permiten a los desarrolladores responder a las acciones del usuario y crear experiencias interactivas en la web.
- **Importancia del DOM y eventos:** Ambos son elementos esenciales en el desarrollo web porque permiten crear aplicaciones web dinámicas e interactivas. El DOM proporciona una estructura para representar el contenido de una página web, mientras que los eventos permiten que la página responda a las acciones del usuario de manera efectiva.
- **Interacción y experiencia del usuario:** El uso adecuado del DOM y los eventos es crucial para mejorar la experiencia del usuario en una página web. Permite crear interfaces de usuario intuitivas y receptivas, lo que puede aumentar la satisfacción del usuario y la eficacia de la aplicación.
- **Desarrollo de aplicaciones web modernas:** En el panorama actual del desarrollo web, donde la interactividad y la experiencia del usuario son aspectos clave, comprender y dominar el DOM y los eventos en JavaScript es fundamental para crear aplicaciones web modernas y competitivas.

En resumen, el DOM y los eventos en JavaScript son conceptos fundamentales que todo desarrollador web debe comprender y dominar para crear aplicaciones web dinámicas, interactivas y modernas. Su correcto uso puede marcar la diferencia en la experiencia del usuario y el éxito de una aplicación web.

Despedida y recursos adicionales

¡Enhorabuena por completar el módulo de DOM y eventos en JavaScript! Ahora estás un paso más cerca de dominar el desarrollo frontend avanzado. Recuerda que el aprendizaje es un viaje continuo, y siempre hay oportunidades para seguir creciendo como desarrollador.

Para continuar fortaleciendo tus habilidades en JavaScript y desarrollo web, te recomiendo explorar los siguientes recursos adicionales:

- Documentación oficial de JavaScript: Siempre es útil consultar la documentación oficial de JavaScript para profundizar en los conceptos y entender las últimas actualizaciones del lenguaje.
- Tutoriales en línea: Hay una gran cantidad de tutoriales en línea que cubren temas específicos relacionados con el DOM, eventos y JavaScript en general. Busca tutoriales en plataformas como MDN Web Docs, W3Schools, freeCodeCamp y YouTube.
- Comunidad y redes sociales: Únete a comunidades de desarrolladores en línea, participa en foros como Stack Overflow y sigue a expertos en JavaScript en redes sociales como Twitter y LinkedIn. La interacción con otros desarrolladores y el intercambio de conocimientos pueden ser muy enriquecedores.

Recuerda practicar regularmente, experimentar con diferentes proyectos y desafíos, y nunca tengas miedo de seguir aprendiendo. ***¡El mundo del desarrollo web está lleno de oportunidades emocionantes! ¡Mucho éxito en tu viaje de aprendizaje!***