Engineer

DOM

árbol vivo de objetos que el navegador crea desde una página

Qué es

El DOM, abreviatura de Document Object Model, es el árbol de objetos que un navegador construye a partir de una página y mantiene en memoria mientras esta permanece abierta. El analizador HTML lee el marcado y produce un objeto documento: cada elemento, cada secuencia de texto y cada comentario se convierten en un nodo, y cada nodo conoce a su padre, sus hijos y sus vecinos. Los scripts leen y cambian la página a través de ese árbol, y a partir de él se determina qué dibuja el navegador.

WHATWG mantiene el modelo en el estándar DOM, cuyo resumen enumera tres ámbitos sin vincularlos a ninguna plataforma concreta: árboles de nodos, eventos y cancelación de actividades en curso. El nombre es anterior al estándar. En su sentido original designaba una interfaz para acceder a un documento y modificarlo; hoy la mayoría de las personas también llaman «el DOM» al propio árbol, y ambos significados se usan a diario. El estándar HTML se apoya en él: HTML define los elementos y lo que hacen; el estándar DOM define qué son un nodo, un elemento, un atributo y un evento.

El árbol no es el archivo fuente. El HTML de una página es texto que llegó por la red. El DOM es lo que el analizador hizo con ese texto, aplicando a cada error de análisis las reglas que detalla el estándar HTML, y lo que después ha modificado cada script ejecutado. El analizador conserva detalles fáciles de olvidar. El espacio en blanco entre dos etiquetas puede convertirse en su propio nodo de texto; por eso el primer hijo de un elemento no siempre es el primer elemento de su interior, aunque el ejemplo del propio estándar señala que no toda secuencia de espacios en blanco se convierte en un nodo de texto.

Hay unos pocos tipos de nodos. El documento ocupa la raíz, con una declaración de tipo de documento, elementos, nodos de texto y comentarios debajo. Un fragmento de documento es un contenedor ligero situado fuera del árbol de la página, lo que resulta útil para montar un bloque antes de insertarlo de una sola vez. Un elemento también puede alojar un árbol de sombra, un árbol separado propio al que no llegan las consultas ordinarias del documento.

Las listas de nodos son de dos tipos, y la diferencia importa. La regla del estándar es que una colección está viva salvo que se indique lo contrario: `childNodes` y la HTMLCollection de `getElementsByClassName()` cambian al cambiar el árbol. `querySelectorAll()` es la excepción explícita y devuelve una lista estática, una instantánea tomada en el momento de la llamada.

El mismo estándar define cómo viajan los eventos. Un evento se despacha a un nodo de destino y pasa por tres fases: captura, al bajar desde lo alto del árbol por los antecesores del destino; en el propio destino; y propagación ascendente, al volver por los antecesores. La propiedad `eventPhase` indica dónde está con los valores 1, 2 o 3, y con 0 cuando no se está despachando el evento. Un escuchador añadido con `addEventListener()` admite opciones: `once` lo elimina después de su primera llamada; `passive` promete que no llamará a `preventDefault()`, y `signal` lo vincula a un `AbortController`, de modo que cancelar el controlador elimina el escuchador.

Utilice el concepto en una decisión

Aplique «¿Qué es el DOM?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Interfaces que cargan rápido, funcionan en el teléfono y se comportan igual en los navegadores que utilizan sus clientes.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Dirección del desarrollo integral de mapas GIS — cartografía integral, de la base de datos al navegador

Dirección del desarrollo integral de mapas GIS Póngase en contacto →

Por qué importa

Todo script que cambia una página trabaja a través del DOM, sea cual sea su lenguaje o framework. Un framework añade su propia capa encima, pero lo que el lector ve al final son nodos de ese árbol. Next.js describe la hidratación como la conexión de los manejadores de eventos de React al DOM: así el HTML enviado

por el servidor se convierte en una página que responde a los clics. Un equipo que conoce el árbol subyacente puede depurar lo que el framework hizo realmente, no solo lo que se le pidió.

El árbol también es donde el texto y el marcado toman caminos distintos. Una cadena asignada a `textContent` sustituye los hijos de un elemento por un único nodo de texto y nunca se interpreta como marcado. Esa misma cadena asignada a `innerHTML` se entrega al analizador HTML, y el estándar HTML señala que esa asignación no elimina nada peligroso de forma predeterminada. La elección de una u otra determina si el nombre visible de un visitante se muestra como texto o se ejecuta como código.

Lo que experimenta el lector depende de más cosas que el árbol. Los estilos deciden si un nodo es visible, dónde está y si el puntero puede alcanzarlo; nada de eso es visible para una herramienta que solo recorre los nodos. Esa diferencia es donde más suelen equivocarse las comprobaciones automatizadas y explica por qué hay que probar la página tal como se representa, además del documento que hay detrás.

Por último, el DOM es el contrato entre una página y todo lo que la lee después de que el servidor haya terminado. Las pruebas que recorren una página haciendo clic, las extensiones del navegador y el software de asistencia parten del árbol vivo. La otra cara de ese contrato es fácil de pasar por alto: un rastreador que no ejecuta JavaScript solo ve el HTML enviado y ninguno de los nodos que añaden después los scripts.

Cómo se aplica en la práctica

El siguiente script, escrito para este artículo, rellena una lista de pedidos a partir de datos, mantiene un solo escuchador de clics en la lista en lugar de uno por elemento y puede eliminarlo después con una llamada. `openOrder` representa lo que haga la página cuando se elige un pedido.

```
const list = document.querySelector("#orders");
const controller = new AbortController();

function render(orders) {
  const items = orders.map((order) => {
    const li = document.createElement("li");
    li.dataset.id = order.id;
    li.textContent = `${order.number}: ${order.status}`;
```

```
    return li;
  });
  list.replaceChildren(...items);
}

list.addEventListener(
  "click",
  (event) => {
    const li = event.target.closest("li");
    if (li) openOrder(li.dataset.id);
  },
  { signal: controller.signal },
);

// When the view goes away:
controller.abort();
```

Cada parte corresponde al estándar. `createElement()` crea un elemento sin padre hasta que se inserta. `textContent` introduce el número de pedido como texto, de modo que un estado que contenga signos de menor y mayor sigue siendo un estado. `replaceChildren()` elimina los elementos antiguos e inserta los nuevos en una sola operación. El único escuchador de la lista atiende a todos los elementos porque un clic en uno de ellos se propaga hacia arriba hasta la lista, y `closest()` recorre desde lo que recibió el clic hasta el `li` más cercano. Cancelar el controlador elimina el escuchador sin que la página tenga que conservar una referencia a la función.

El árbol también explica una clase de defectos que descubrió la revisión del propio sitio. En una revisión de accesibilidad realizada en agosto de 2026, el sitio superaba sus reglas automatizadas y aun así tenía once defectos, todos debidos a cómo se representaba la página y no a lo que decía su marcado. Ocho enlaces del pie permanecían en el orden de tabulación detrás de `pointer-events`, una propiedad que elimina la interacción del puntero y nada más. Para cualquier herramienta que leyera el documento eran enlaces corrientes; el puntero nunca los alcanzaba y el teclado seguía deteniéndose en cada uno. Una animación vinculada al desplazamiento recortaba el pie de tres páginas. De las cuatro reglas que dejó la revisión, una advierte contra confiar solo en el árbol: hay que establecer la visibilidad a partir de los píxeles representados, no únicamente del árbol del documento.

Dónde surgen los problemas

Un bucle sobre una colección viva se salta uno de cada dos elementos. La HTMLCollection de `getElementsByClassName()` está viva: un bucle que elimina la clase de cada elemento por índice reduce la colección mientras la recorre y se salta un elemento de cada dos. Recorra una lista estática en su lugar: `querySelectorAll()` la devuelve, y `Array.from()` convierte primero una colección viva en un array corriente.

```
const pending = document.getElementsByClassName("pending");
for (let i = 0; i < pending.length; i++) {
  pending[i].classList.remove("pending"); // skips every other one
}

for (const el of document.querySelectorAll(".pending")) {
  el.classList.remove("pending"); // static list: all of them
}
```

Se acumulan los escuchadores. Una vista que añade sus escuchadores cada vez que se representa y nunca los elimina ejecuta cada manejador una vez por representación: tras tres visitas, un clic envía tres solicitudes. Para eliminar a mano un escuchador hace falta exactamente el mismo objeto función que se añadió; una función flecha escrita en línea en la llamada no deja nada que se pueda volver a pasar. Pase una `signal` de un `AbortController` por vista, como en el ejemplo anterior, y un único `abort()` eliminará todos los escuchadores que añadió esa vista. Utilice `once: true` para uno que solo deba ejecutarse una vez.

El texto entra como marcado. Construir una cadena HTML a partir de datos y asignarla a `innerHTML` entrega los datos al analizador sin saneamiento predeterminado. Un nombre visible que contenga marcado, por ejemplo una etiqueta de imagen con un manejador de errores, llega a la página como un elemento real con su manejador conectado. El texto debe ir en `textContent` o en un nodo de texto creado con `createTextNode()` . Cuando el marcado deba proceder realmente de una cadena, el estándar HTML define `setHTML()` , que sanea mientras analiza y elimina el contenido inseguro.

Un observador se alimenta a sí mismo. Un `MutationObserver` informa de los cambios en los nodos que vigila y su función de devolución de llamada se ejecuta después de cada lote de cambios. Si esa función escribe en el subárbol que observa para añadir un contador o corregir una clase, crea registros nuevos y vuelve a ser llamada. La página puede bloquearse en un bucle creado por ella

misma. Vigile solo lo necesario, con `attributeFilter` y sin `subtree` cuando sea posible; haga que la función compruebe si su cambio ya está aplicado antes de escribir, o llame a `disconnect()` antes de escribir y a `observe()` después.

Lecturas adicionales

- Estándar DOM — La definición de WHATWG de nodos, árboles, colecciones vivas y estáticas, eventos, `AbortController` y `MutationObserver`.
- Análisis de documentos HTML — Cómo el analizador HTML convierte un flujo de caracteres en un objeto documento, incluidos los errores de análisis.
- Inserción dinámica de marcado — `innerHTML`, `setHTML()` y `setHTMLUnsafe()`, y cuáles de ellos sanean.
- Componentes de servidor y cliente — La explicación de Next.js sobre la hidratación: HTML del servidor y, después, manejadores de eventos conectados al DOM.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es el DOM?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/dom/