Engineer

DOM

🌲 det levende treet av objekter nettleseren bygger av en side

Hva det er

DOM, en forkortelse for Document Object Model, er treet av objekter som nettleseren bygger av en side og holder i minnet så lenge siden er åpen. HTML-parseren leser oppmerkingen, og resultatet er et dokumentobjekt: hvert element, hver tekstbit og hver kommentar blir en node, og hver node kjenner forelderen sin, barna sine og naboene sine. Skript leser og endrer siden gjennom dette treet, og det er ut fra treet nettleseren regner ut hva som skal tegnes.

WHATWG vedlikeholder modellen i DOM-standarden, som beskriver seg selv som en plattformnøytral modell for hendelser, avbryting av pågående arbeid og nodetrær. Navnet er eldre enn standarden. Opprinnelig betegnet det et grensesnitt for å nå inn i et dokument og endre det. I dag sier de fleste også «DOM-en» om selve treet, og begge betydningene er i bruk hver dag. HTML-standarden bygger videre på den: HTML fastsetter elementene og hvordan de oppfører seg, mens DOM-standarden fastsetter hva en node, et element, et attributt og en hendelse er.

Treet er ikke kildefilen. HTML-en til en side er tekst som har kommet over nettverket. DOM-en er det parseren fikk ut av den teksten, med hver parsefeil håndtert etter regler HTML-standarden skriver ut i detalj, og deretter endret av hvert eneste skript som har kjørt siden. Parseren tar vare på detaljer som er lette å glemme. Mellomrommet mellom to tagger kan bli en egen tekstnode, så det første barnet til et element er ikke alltid det første elementet inni det, selv om standardens eget eksempel påpeker at langt fra alt mellomrom blir til tekstnoder.

Det finnes en håndfull typer noder. Dokumentet sitter i roten, med en doctype, elementer, tekstnoder og kommentarer under seg. Et dokumentfragment er en lett beholder utenfor sidens tre og dermed et praktisk sted å sette sammen en

blokk før den settes inn i én operasjon. Et element kan også bære et skyggetre (shadow tree), et eget tre som dokumentets vanlige spørringer ikke når inn i.

Lister av noder finnes i to varianter, og forskjellen betyr noe. Standardens regel er at en samling er levende med mindre noe annet er sagt: `childNodes` og HTMLCollection-en fra `getElementsByClassName()` følger med når treet endrer seg. `querySelectorAll()` er det uttalte unntaket og returnerer en statisk liste, et øyeblikksbilde fra det øyeblikket kallet ble gjort.

Den samme standarden beskriver hvordan hendelser beveger seg. En hendelse sendes til en målnode og går gjennom tre faser: capture på vei ned fra toppen av treet gjennom målets forfedre, selve målet, og til slutt bubbling tilbake opp gjennom forfedrene. Egenskapen `eventPhase` forteller hvor hendelsen er, som 1, 2 eller 3, og 0 når den ikke er under utsending. En lytter som legges til med `addEventListener()`, tar imot innstillinger: `once` fjerner den etter første kall, `passive` lover at den aldri kaller `preventDefault()`, og `signal` knytter den til en `AbortController`, slik at lytteren forsvinner når kontrolleren avbrytes.

Bruk begrepet i en beslutning

Bruk «Hva er DOM?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Grensesnitt som laster raskt, fungerer på mobilen og oppfører seg likt i alle nettlesere kundene dine måtte bruke.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Styrte fullstack-utviklingen av GIS-kart — fullstack-kartutvikling, fra database til nettleser

Styrte fullstack-utviklingen av GIS-kart Ta kontakt →

Hvorfor det er viktig

Hvert skript som endrer en side, arbeider gjennom DOM-en, uansett hvilket språk eller rammeverk det er skrevet i. Et rammeverk legger sitt eget lag oppå, men det leseren til slutt ser, er noder i dette treet. Next.js beskriver hydrering som Reacts måte å feste hendelseshåndterere til DOM-en på, og det er slik HTML sendt fra serveren blir til en side som reagerer på klikk. Et team som kjenner treet under, kan feilsøke det rammeverket faktisk gjorde, ikke bare det det ble bedt om.

Treet er også stedet der tekst og oppmerking skilles. En streng som tilordnes `textContent`, erstatter barna til elementet med én enkelt tekstnode og blir aldri lest som oppmerking. Den samme strengen tilordnet `innerHTML` går til HTML-

parseren, og HTML-standarden påpeker at disse setterne i utgangspunktet ikke fjerner noe farlig. Hvilken av de to en side bruker, avgjør om visningsnavnet til en besøkende blir vist som tekst eller kjørt som kode.

Det en leser opplever, avhenger av mer enn treet. Stiler avgjør om en node er synlig, hvor den står og om en peker kan nå den, og et verktøy som bare går gjennom nodene, ser ingenting av dette. Det er i dette mellomrommet automatiske kontroller oftest tar feil, og det er grunnen til å teste siden slik den blir vist, og ikke bare dokumentet bak den.

Til slutt er DOM-en kontrakten mellom en side og alt som leser den etter at serveren har gjort sitt. Tester som klikker seg gjennom en side, nettleserutvidelser og hjelpemidler tar alle utgangspunkt i det levende treet. Den andre siden av kontrakten er lett å overse: en crawler som ikke kjører JavaScript, ser bare HTML-en som ble sendt, og ingen av nodene skript legger til etterpå.

Slik ser det ut i praksis

Skriptet nedenfor er skrevet for denne artikkelen. Det fyller en liste med ordrer ut fra data, holder én klikklytter på selve listen i stedet for én per punkt og kan fjerne lytteren igjen med ett kall. `openOrder` står for det siden gjør når en ordre velges.

```
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 },
```

```
);

// Når visningen forsvinner:
controller.abort();
```

Hver del svarer til noe i standarden. `createElement()` lager et element som står uten forelder til det settes inn. `textContent` legger ordrenummeret inn som tekst, så en status med vinkelparenteser i forblir en status. `replaceChildren()` fjerner de gamle punktene og setter inn de nye i én operasjon. Den ene lytteren på listen dekker alle punktene, fordi et klikk på et punkt bobler opp til listen, og `closest()` går fra det som ble klikket på, til nærmeste `li` . Når kontrolleren avbrytes, forsvinner lytteren uten at siden må ta vare på en referanse til funksjonen.

Treet forklarer også en type feil som nettstedets egen gjennomgang fant. I en gjennomgang av universell utforming i august 2026 besto nettstedet de automatiske reglene sine og hadde likevel elleve feil, der hver enkelt skyldtes hvordan siden ble gjengitt, ikke hva oppmerkingen sa. Åtte lenker i bunnteksten ble værende i tabulatorrekkefølgen bak `pointer-events` , en egenskap som fjerner pekerinteraksjon og ingenting annet. For alt som leste dokumentet, var de helt vanlige lenker. En peker nådde dem aldri, men tastaturet stoppet ved hver eneste en. En rullestyrt animasjon klippet bunnteksten bort på tre sider. Av de fire reglene gjennomgangen etterlot seg, lyder én som en advarsel mot å stole på treet alene: «bevis synlighet i piksler, ikke i dokumentet».

Hvor det går galt

En løkke over en levende samling hopper over annethvert punkt. HTMLCollection-en fra `getElementsByClassName()` er levende, så en løkke som fjerner klassen fra hvert element etter indeks, får samlingen til å krympe under seg og hopper over annethvert element. Gå heller gjennom en statisk liste: `querySelectorAll()` returnerer en, og `Array.from()` gjør en levende samling om til en vanlig array først.

```
const pending = document.getElementsByClassName("pending");
for (let i = 0; i < pending.length; i++) {
  pending[i].classList.remove("pending"); // hopper over annethvert
}
```

```
for (const el of document.querySelectorAll(".pending")) {
  el.classList.remove("pending"); // statisk liste: alle sammen
}
```

Lytterne hoper seg opp. En visning som legger til lytterne sine hver gang den rendres, og aldri fjerner dem, kjører hver håndterer én gang per rendering: etter tre besøk sender ett klikk tre forespørsler. Å fjerne en lytter for hånd krever nøyaktig det funksjonsobjektet som ble lagt til, og en pilfunksjon skrevet rett inn i kallet etterlater ingenting å gi tilbake. Gi heller hver visning sin egen `AbortController` og send med `signal` -en, som i eksemplet over, så fjerner ett `abort()` alle lytterne visningen la til. Bruk `once: true` for en lytter som bare skal reagere én gang.

Tekst kommer inn som oppmerking. Bygger man en HTML-streng av data og tilordner den til `innerHTML`, går dataene rett til parseren, og i utgangspunktet blir ingenting renset bort. Et visningsnavn med oppmerking i, for eksempel en bildetagg med en feilhåndterer, lander på siden som et ekte element med håndtereren festet. Tekst hører hjemme i `textContent` eller i en tekstnode laget med `createTextNode()`. Må oppmerking virkelig komme fra en streng, definerer HTML-standarden `setHTML()`, som renser under parsingen og fjerner usikkert innhold.

En observatør mater seg selv. En `MutationObserver` melder endringer i nodene den holder øye med, og tilbakekallet kjører etter hver porsjon endringer. Et tilbakekall som selv skriver i undertreet det observerer, for å legge til en teller eller rette en klasse, skaper nye endringsposter og blir kalt igjen, og siden kan låse seg i en løkke den har laget selv. Hold bare øye med det som trengs, med `attributeFilter` og helst uten `subtree`, la tilbakekallet sjekke om endringen allerede er på plass før det skriver, eller kall `disconnect()` før skrivingen og `observe()` igjen etterpå.

Les mer

- DOM Standard — WHATWGs definisjon av noder, trær, levende og statiske samlinger, hendelser, `AbortController` og `MutationObserver`.
- Parsing HTML documents — hvordan HTML-parseren gjør en strøm av tegn om til et dokumentobjekt, parsefeil inkludert.
- Dynamic markup insertion — `innerHTML`, `setHTML()` og `setHTMLUnsafe()`, og hvilke av dem som renser.

- Server and Client Components — Next.js om hydrering: HTML fra serveren, deretter hendelseshåndterere festet til DOM-en.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er DOM?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/dom/