

Engineer

DOM

🌲 det levende træ af objekter, som browseren bygger af en side



Hvad det er

DOM, en forkortelse af Document Object Model, er det træ af objekter, som browseren bygger af en side og holder i hukommelsen, så længe siden er åben. HTML-parseren læser markuppen, og resultatet er et dokumentobjekt: hvert element, hvert stykke tekst og hver kommentar bliver til en node, og hver node kender sin forælder, sine børn og sine naboer. Scripts læser og ændrer siden gennem det træ, og det er ud fra træet, browseren regner ud, hvad der skal tegnes.

WHATWG vedligeholder modellen i DOM-standarden, som beskriver sig selv som en platformsneutral model for events, afbrydelse af igangværende arbejde og nodetræer. Navnet er ældre end standarden. Oprindeligt betegnede det en grænseflade til at nå ind i et dokument og ændre det. I dag siger de fleste også »DOM'en« om selve træet, og begge betydninger er i brug hver dag. HTML-standarden bygger ovenpå: HTML fastlægger elementerne og deres opførsel, mens DOM-standarden fastlægger, hvad en node, et element, en attribut og et event er.

Træet er ikke kildefilen. En sides HTML er tekst, der er kommet over netværket. DOM'en er det, parseren fik ud af den tekst, med hver parsefejl håndteret efter regler, som HTML-standarden skriver ud i detaljer, og derefter ændret af hvert eneste script, der har kørt siden. Parseren bevarer detaljer, der er lette at glemme. Mellemrummet mellem to tags kan blive sin egen tekstnode, så det første barn af et element er ikke altid det første element inde i det, selv om standardens eget eksempel gør opmærksom på, at langtfra al whitespace bliver til tekstnoder.

Der findes en håndfuld slags noder. Dokumentet sidder i roden med en doctype, elementer, tekstnoder og kommentarer under sig. Et dokumentfragment er en let beholder uden for sidens træ og derfor et praktisk sted at samle en blok, før den sættes ind i ét hug. Et element kan også bære et shadow tree, et separat træ for sig, som dokumentets almindelige forespørgsler ikke når ind i.

Lister af noder findes i to udgaver, og forskellen betyder noget. Standardens regel er, at en samling er levende, medmindre andet er angivet: `childNodes` og den HTMLCollection, som `getElementsByClassName()` returnerer, følger med, når træet ændrer sig. `querySelectorAll()` er den angivne undtagelse og returnerer en statisk liste, et øjebliksbillede fra det tidspunkt, kaldet blev foretaget.

Samme standard beskriver, hvordan events bevæger sig. Et event sendes til en målnode og passerer tre faser: capture på vej ned fra toppen af træet gennem målets forfædre, selve målet og til sidst bubbling tilbage op gennem forfædrene. Egenskaben `eventPhase` fortæller, hvor eventet er, som 1, 2 eller 3, og 0, når det ikke er ved at blive afsendt. En lytter, der tilføjes med `addEventListener()`, tager imod indstillinger: `once` fjerner den efter første kald, `passive` lover, at den aldrig kalder `preventDefault()`, og `signal` binder den til en `AbortController`, så lytteren forsvinder, når controlleren afbrydes.

Brug begrebet i en beslutning

Brug »Hvad er DOM?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Brugerflader, der indlæses hurtigt, virker på en telefon og opfører sig ens i alle de browsere, jeres kunder nu bruger.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har styret full-stack-udviklingen af GIS-kort — full stack-kortløsning, fra database til browser

Har styret full-stack-udviklingen af GIS-kort Kontakt os →

Hvorfor det betyder noget

Hvert script, der ændrer en side, arbejder gennem DOM'en, uanset hvilket sprog eller framework det er skrevet i. Et framework lægger sit eget lag ovenpå, men det, læseren ender med at se, er noder i det her træ. Next.js beskriver hydrering som Reacts måde at hæfte eventhandlere på DOM'en, og det er sådan, HTML sendt fra serveren bliver til en side, der reagerer på klik. Et team, der kender træet nedenunder, kan fejlsøge det, frameworket faktisk gjorde, og ikke kun det, det blev bedt om.

Træet er også der, hvor tekst og markup skilles. En streng, der tildeles `textContent`, erstatter elementets børn med én enkelt tekstnode og bliver aldrig læst som markup. Den samme streng tildelt `innerHTML` går til HTML-parseren, og HTML-standarden gør opmærksom på, at de settere som udgangspunkt intet farligt fjerner. Hvilken af de to en side bruger, afgør, om en besøgendes visningsnavn bliver vist som tekst eller kørt som kode.

Det, en læser oplever, afhænger af mere end træet. Styles afgør, om en node er synlig, hvor den står, og om en pointer kan nå den, og intet af det ses af et værktøj, der kun går noderne igennem. Det er i det mellemrum, automatiske tjek oftest tager fejl, og det er grunden til at teste siden, som den bliver vist, og ikke kun dokumentet bag den.

Endelig er DOM'en kontrakten mellem en side og alt det, der læser den, efter serveren har gjort sit. Tests, der klikker sig gennem en side, browserudvidelser og hjælpemidler tager alle udgangspunkt i det levende træ. Den anden side af kontrakten er let at overse: en crawler, der ikke kører JavaScript, ser kun den HTML, der blev sendt, og ingen af de noder, som scripts tilføjer bagefter.

Sådan ser det ud i praksis

Scriptet herunder er skrevet til denne artikel. Det fylder en liste med ordrer ud fra data, holder én kliklytter på selve listen i stedet for én pr. punkt og kan fjerne lytteren igen med et enkelt kald. `openOrder` står for det, siden gør, når en ordre vælges.

```
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 forsvinder:
controller.abort();
```

Hver del svarer til noget i standarden. `createElement()` laver et element, der står uden forælder, indtil det sættes ind. `textContent` lægger ordrenummeret ind som tekst, så en status med vinkelparenteser i forbliver en status. `replaceChildren()` fjerner de gamle punkter og indsætter de nye i én operation. Den ene lytter på listen dækker alle punkter, fordi et klik på et punkt bobler op til listen, og `closest()` går fra det, der blev klikket på, til nærmeste `li`. Når controlleren afbrydes, forsvinder lytteren, uden at siden skal gemme en reference til funktionen.

Træet forklarer også en type fejl, som sitets egen gennemgang fandt. I en tilgængelighedsgennemgang fra august 2026 bestod sitet sine automatiske regler og havde alligevel elleve defekter, som hver især skyldtes, hvordan siden blev gengivet, og ikke, hvad markuppen sagde. Otte footer-links blev liggende i tab-rækkefølgen bag `pointer-events`, en egenskab, der fjerner pointer-interaktion og intet andet. For alt, der læste dokumentet, var de helt almindelige links. En pointer nåede dem aldrig, men tastaturet stoppede ved hvert eneste. En scroll-drevet animation klippede footeren af på tre sider. Af de fire regler, gennemgangen efterlod, lyder én som en advarsel mod at stole på træet alene: »bevis synlighed i pixels og ikke i dokumentet«.

Hvor det går galt

En løkke over en levende samling springer hvert andet punkt over. Den HTMLCollection, som `getElementsByClassName()` returnerer, er levende, så en løkke, der fjerner klassen fra hvert element efter indeks, får samlingen til at skrumpe under sig og springer hvert andet element over. Gå i stedet en statisk liste igennem: `querySelectorAll()` returnerer en, og `Array.from()` gør en levende samling til et almindeligt array først.

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

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

Lytterne hober sig op. En visning, der tilføjer sine lyttere, hver gang den renderes, og aldrig fjerner dem, kører hver handler én gang pr. rendering: efter tre besøg sender ét klik tre forespørgsler. At fjerne en lytter i hånden kræver præcis det funktionsobjekt, der blev tilføjet, og en pilefunktion skrevet direkte i kaldet efterlader intet at give tilbage. Giv i stedet hver visning sin egen `AbortController` og send dens `signal` med, som i eksemplet ovenfor, så fjerner ét `abort()` alle visningens lyttere. Brug `once: true` til en lytter, der kun skal reagere én gang.

Tekst kommer ind som markup. Bygger man en HTML-streng af data og tildeler den til `innerHTML`, går data direkte til parseren, og som udgangspunkt bliver intet renset væk. Et visningsnavn med markup i, for eksempel et billedtag med en fejlhandler, lander på siden som et rigtigt element med handleren påsat. Tekst hører hjemme i `textContent` eller i en tekstnode lavet med `createTextNode()`. Skal markup virkelig komme fra en streng, definerer HTML-standarden `setHTML()`, som renser under parsningen og fjerner usikkert indhold.

En observer fodrer sig selv. En `MutationObserver` melder ændringer i de noder, den holder øje med, og dens callback kører efter hver portion ændringer. Et callback, der selv skriver i det deltræ, det observerer, for at tilføje en tæller eller rette en klasse, skaber nye ændringsposter og bliver kaldt igen, og siden kan låse fast i en løkke, den selv har skabt. Hold kun øje med det, der er nødvendigt, med `attributeFilter` og helst uden `subtree`, lad callbacket tjekke, om ændringen allerede er på plads, før det skriver, eller kald `disconnect()` før skrivningen og `observe()` igen bagefter.

Læs mere

- DOM Standard — WHATWG's definition af noder, træer, levende og statiske samlinger, events, `AbortController` og `MutationObserver`.
- Parsing HTML documents — hvordan HTML-parseren gør en strøm af tegn til et dokumentobjekt, inklusive parsefejl.
- Dynamic markup insertion — `innerHTML`, `setHTML()` og `setHTMLUnsafe()`, og hvilke af dem der renser.
- Server and Client Components — Next.js om hydrering: HTML fra serveren, derefter eventhandlere hæftet på DOM'en.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er DOM?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/dom/