Engineer

DOM

det levande objektträd som webbläsaren bygger av en sida

Vad det är

DOM, en förkortning av Document Object Model, är det träd av objekt som webbläsaren bygger av en sida och håller i minnet så länge sidan är öppen. HTML-parsern läser uppmärkningen, och resultatet är ett dokumentobjekt: varje element, varje textsnutt och varje kommentar blir en nod, och varje nod känner till sin förälder, sina barn och sina grannar. Skript läser och ändrar sidan genom det trädet, och det är utifrån trädet som webbläsaren räknar ut vad som ska ritas.

WHATWG underhåller modellen i DOM-standarden, som beskriver sig själv som en plattformsneutral modell för händelser, avbrytande av pågående arbete och nodträd. Namnet är äldre än standarden. Ursprungligen betecknade det ett gränssnitt för att nå in i ett dokument och ändra det. I dag säger de flesta också ”DOM:en” om själva trädet, och båda betydelserna används dagligen. HTML-standarden bygger vidare på den: HTML fastställer elementen och hur de beter sig, medan DOM-standarden fastställer vad en nod, ett element, ett attribut och en händelse är.

Trädet är inte källfilen. En sidas HTML är text som har kommit över nätverket. DOM:en är det som parsern fick ut av texten, med varje parsningsfel hanterat enligt regler som HTML-standarden skriver ut i detalj, och därefter ändrat av varje skript som har körts sedan dess. Parsern behåller detaljer som är lätta att glömma. Blanktecknen mellan två taggar kan bli en egen textnod, så det första barnet till ett element är inte alltid det första elementet inuti det, även om standardens eget exempel påpekar att långt ifrån alla blanktecken blir textnoder.

Det finns en handfull sorters noder. Dokumentet sitter i roten, med en doctype, element, textnoder och kommentarer under sig. Ett dokumentfragment är en lätt behållare utanför sidans träd och därför ett praktiskt ställe att sätta ihop ett

block innan det infogas i ett enda steg. Ett element kan också bära ett skuggträd (shadow tree), ett eget träd som dokumentets vanliga sökningar inte når in i.

Listor av noder finns i två varianter, och skillnaden spelar roll. Standardens regel är att en samling är levande om inget annat anges: `childNodes` och den HTMLCollection som `getElementsByClassName()` returnerar följer med när trädet ändras. `querySelectorAll()` är det uttalade undantaget och returnerar en statisk lista, en ögonblicksbild från det ögonblick då anropet gjordes.

Samma standard beskriver hur händelser färdas. En händelse skickas till en målnod och passerar tre faser: capture på väg ned från toppen av trädet genom målets förfäder, själva målet och till sist bubbling tillbaka upp genom förfäderna. Egenskapen `eventPhase` talar om var händelsen är, som 1, 2 eller 3, och 0 när den inte håller på att skickas. En lyssnare som läggs till med `addEventListener()` tar emot inställningar: `once` tar bort den efter första anropet, `passive` lovar att den aldrig anropar `preventDefault()`, och `signal` knyter den till en `AbortController`, så att lyssnaren försvinner när kontrollern avbryts.

Använd begreppet i ett beslut

Använd ”Vad är DOM?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Gränssnitt som laddar snabbt, fungerar i mobilen och beter sig likadant i alla webbläsare dina kunder råkar använda.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Styrde full stack-utvecklingen av GIS-kartor — kartutveckling i full stack, från databas till webbläsare

Styrde full stack-utvecklingen av GIS-kartor Hör av er →

Varför det spelar roll

Varje skript som ändrar en sida arbetar genom DOM:en, oavsett vilket språk eller ramverk det är skrivet i. Ett ramverk lägger sitt eget lager ovanpå, men det läsaren till slut ser är noder i det här trädet. Next.js beskriver hydrering som Reacts sätt att fästa händelsehanterare vid DOM:en, och det är så HTML som skickats från servern blir en sida som svarar på klick. Ett team som känner trädet därunder kan felsöka det ramverket faktiskt gjorde, och inte bara det som det ombads göra.

Trädet är också platsen där text och uppmärkning skiljs åt. En sträng som tilldelas `textContent` ersätter elementets barn med en enda textnod och läses aldrig som uppmärkning. Samma sträng tilldelad `innerHTML` går till HTML-parsern, och

HTML-standarden påpekar att dessa tilldelningsmetoder som standard inte tar bort något farligt. Vilken av de två en sida använder avgör om en besökares visningsnamn visas som text eller körs som kod.

Det en läsare upplever beror på mer än trädet. Stilar avgör om en nod är synlig, var den står och om en pekare kan nå den, och ett verktyg som bara går igenom noderna ser ingenting av det. Det är i det glappet som automatiska kontroller oftast har fel, och det är skälet att testa sidan så som den visas och inte bara dokumentet bakom den.

Till sist är DOM:en kontraktet mellan en sida och allt som läser den efter att servern har gjort sitt. Tester som klickar sig igenom en sida, webbläsartillägg och hjälpmedel utgår alla från det levande trädet. Den andra sidan av kontraktet är lätt att missa: en crawler som inte kör JavaScript ser bara den HTML som skickades och ingen av de noder som skript lägger till senare.

Så ser det ut i praktiken

Skriptet nedan är skrivet för den här artikeln. Det fyller en lista med ordrar utifrån data, håller en enda klicklyssnare på själva listan i stället för en per punkt och kan ta bort lyssnaren igen med ett enda anrop. `openOrder` står för det som sidan gör när en order väljs.

```
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 },
);
```

```
// När vyn försvinner:
controller.abort();
```

Varje del motsvarar något i standarden. `createElement()` skapar ett element som står utan förälder tills det infogas. `textContent` lägger in ordernumret som text, så att en status med vinkelparenteser i förblir en status. `replaceChildren()` tar bort de gamla punkterna och infogar de nya i en enda operation. Den enda lyssnaren på listan räcker till alla punkter, eftersom ett klick på en punkt bubblar upp till listan, och `closest()` går från det som klickades på till närmaste `li`. När kontrollern avbryts försvinner lyssnaren, utan att sidan behöver spara en referens till funktionen.

Trädet förklarar också en sorts fel som webbplatsens egen granskning hittade. I en tillgänglighetsgranskning i augusti 2026 klarade webbplatsen sina automatiska regler och hade ändå elva fel, där vart och ett berodde på hur sidan renderades snarare än på vad uppmärkningen sade. Åtta länkar i sidfoten låg kvar i tabbordningen bakom `pointer-events`, en egenskap som tar bort pekarinteraktion och ingenting annat. För allt som läste dokumentet var de helt vanliga länkar. En pekare nådde dem aldrig, men tangentbordet stannade vid varenda en. En rullningsstyrd animation skar bort sidfoten på tre sidor. Av de fyra regler som granskningen lämnade efter sig låter en som en varning mot att lita på trädet ensamt: ”bevisa synlighet i pixlar snarare än i dokumentet”.

Var det går fel

En loop över en levande samling hoppar över varannan punkt. Den HTMLCollection som `getElementsByClassName()` returnerar är levande, så en loop som tar bort klassen från varje element efter index får samlingen att krympa under sig och hoppar över vartannat element. Gå i stället igenom en statisk lista: `querySelectorAll()` returnerar en, och `Array.from()` gör om en levande samling till en vanlig array först.

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

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

Lyssnarna hopar sig. En vy som lägger till sina lyssnare varje gång den renderas, och aldrig tar bort dem, kör varje hanterare en gång per rendering: efter tre besök skickar ett klick tre förfrågningar. Att ta bort en lyssnare för hand kräver exakt det funktionsobjekt som lades till, och en pilfunktion skriven direkt i anropet lämnar inget att skicka tillbaka. Ge i stället varje vy en egen `AbortController` och skicka med dess `signal`, som i exemplet ovan, så tar ett enda `abort()` bort alla lyssnare som vyn lade till. Använd `once: true` för en lyssnare som bara ska reagera en gång.

Text kommer in som uppmärkning. Bygger man en HTML-sträng av data och tilldelar den till `innerHTML` går datan direkt till parsern, och som standard rensas ingenting bort. Ett visningsnamn med uppmärkning i, till exempel en bildtagg med en felhanterare, landar på sidan som ett riktigt element med hanteraren på plats. Text hör hemma i `textContent` eller i en textnod skapad med `createTextNode()`. Måste uppmärkning verkligen komma från en sträng definierar HTML-standarden `setHTML()`, som sanerar under parsningen och tar bort osäkert innehåll.

En observatör matar sig själv. En `MutationObserver` rapporterar ändringar i de noder den håller ögonen på, och dess callback körs efter varje omgång ändringar. Ett callback som själv skriver i det delträd det observerar, för att lägga till en räknare eller rätta en klass, skapar nya ändringsposter och anropas igen, och sidan kan låsa sig i en loop den själv har skapat. Bevaka bara det som behövs, med `attributeFilter` och helst utan `subtree`, låt callbacket kontrollera om ändringen redan är på plats innan det skriver, eller anropa `disconnect()` före skrivningen och `observe()` igen efteråt.

Läs mer

- DOM Standard — WHATWG:s definition av noder, träd, levande och statiska samlingar, händelser, `AbortController` och `MutationObserver`.
- Parsing HTML documents — hur HTML-parsern gör om en ström av tecken till ett dokumentobjekt, parsningsfel inräknade.
- Dynamic markup insertion — `innerHTML`, `setHTML()` och `setHTMLUnsafe()`, och vilka av dem som sanerar.
- Server and Client Components — Next.js om hydrering: HTML från servern, därefter händelsehanterare fästa vid DOM:en.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är DOM?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/dom/