Engineer

DOM

живе дерево об'єктів, яке браузер будує зі сторінки

Що це таке

DOM (Document Object Model, об’єктна модель документа) — це дерево об’єктів, яке браузер будує зі сторінки й тримає в пам’яті, доки сторінку відкрито. HTML-парсер читає розмітку, а на виході дає об’єкт документа: кожен елемент, кожен фрагмент тексту й кожен коментар стає вузлом, і кожен вузол знає свого батька, своїх дітей і своїх сусідів. Скрипти читають і змінюють сторінку саме через це дерево, і саме з нього браузер виводить, що малювати на екрані.

Модель підтримує WHATWG у стандарті DOM, який описує себе як платформно нейтральну модель для подій, переривання розпочатих операцій і дерев вузлів. Назва старша за стандарт. Спершу так називали інтерфейс, через який можна дістатися до документа й змінити його; сьогодні «DOM» здебільшого кажуть і про саме дерево, і обидва значення живуть у щоденній мові. Стандарт HTML спирається на нього: HTML визначає елементи та їхню поведінку, а стандарт DOM — що таке вузол, елемент, атрибут і подія.

Дерево — не вихідний файл. HTML сторінки — це текст, що прийшов мережею. DOM — це те, що парсер зробив із цього тексту, обробивши кожну помилку розбору за правилами, які стандарт HTML розписує до дрібниць, а потім змінив кожен скрипт, що встиг виконатися. Парсер зберігає подробиці, про які легко забути. Пробіли між двома тегами можуть стати окремим текстовим вузлом, тож перший дочірній вузол елемента не завжди є першим елементом усередині нього, хоча власний приклад стандарту зауважує, що далеко не кожен пробільний фрагмент перетворюється на текстовий вузол.

Вузли бувають кількох видів. У корені стоїть документ, під ним — оголошення типу документа, елементи, текстові вузли й коментарі. Фрагмент документа — це легкий контейнер поза деревом сторінки, тому в ньому зручно зібрати цілий блок, а потім вставити його одним рухом. Елемент може також мати тіньове дерево (shadow tree), окреме дерево, куди звичайні запити документа не дістають.

Списки вузлів бувають двох типів, і різниця суттєва. Правило стандарту таке: колекція жива, якщо не сказано інше. `childNodes` і HTMLCollection, яку повертає `getElementsByClassName()`, змінюються разом із деревом. `querySelectorAll()` — прямо названий виняток: він повертає статичний список, знімок дерева на момент виклику.

Той самий стандарт описує, як рухаються події. Подію надсилають цільовому вузлу, і вона проходить три фази: перехоплення, коли рухається згори дерева вниз крізь предків цілі; саму ціль; і спливання, коли повертається вгору тими самими предками. Властивість `eventPhase` показує, де подія зараз, числом 1, 2 або 3, а 0 означає, що подію не надсилають. Обробник, доданий через `addEventListener()`, приймає параметри: `once` знімає його після першого виклику, `passive` обіцяє, що він не викликатиме `preventDefault()`, а `signal` прив’язує його до `AbortController`, тож коли контролер переривають, обробник зникає.

Застосуйте поняття до рішення

Застосуйте «Що таке DOM?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Інтерфейси, що швидко завантажуються, працюють на телефоні й поводяться однаково в усіх браузерах ваших клієнтів.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Керувала full-stack-розробкою GIS-карт — full-stack-картографія — від бази даних до браузера

Керувала full-stack-розробкою GIS-карт Зв'язатися з нами →

Чому це важливо

Кожен скрипт, що змінює сторінку, працює через DOM, хоч якою мовою і з яким фреймворком його написано. Фреймворк додає зверху власний шар, але те, що зрештою бачить читач, — це вузли саме цього дерева. Next.js описує гідратацію як спосіб, у який React прикріплює обробники подій до DOM, і саме так HTML, надісланий із сервера, перетворюється на сторінку,

що відповідає на клацання. Команда, яка розуміє дерево під фреймворком, може налагоджувати те, що фреймворк справді зробив, а не лише те, про що його просили.

У дереві також розходяться текст і розмітка. Рядок, присвоєний `textContent`, замінює дочірні вузли елемента одним текстовим вузлом і ніколи не читається як розмітка. Той самий рядок, присвоєний `innerHTML`, потрапляє до HTML-парсера, а стандарт HTML застерігає, що за замовчуванням ці сетери нічого небезпечного не вичищають. Від того, яким із двох шляхів іде сторінка, залежить, чи ім'я відвідувача буде показано як текст, чи виконано як код.

Те, що відчуває читач, залежить не лише від дерева. Стилі вирішують, чи вузол видимий, де він стоїть і чи дотягнеться до нього вказівник, а інструмент, який лише обходить вузли, цього не бачить. Саме в цьому проміжку автоматичні перевірки помиляються найчастіше, і саме тому сторінку варто перевіряти такою, якою її відображено, а не тільки документ, що стоїть за нею.

Зрештою, DOM — це угода між сторінкою та всім, що читає її після того, як сервер закінчив роботу. Тести, що проклацують сторінку, розширення браузера й допоміжні технології — усі починають із живого дерева. Інший бік цієї угоди легко прогледіти: пошуковий робот, який не виконує JavaScript, бачить тільки надісланий HTML і жодного вузла, що його скрипти додають пізніше.

Як це виглядає на практиці

Скрипт нижче написано для цієї статті. Він заповнює список замовлень із даних, тримає один обробник клацань на самому списку, а не по одному на кожен пункт, і може зняти цей обробник одним викликом. `openOrder` позначає те, що сторінка робить, коли користувач вибирає замовлення.

```
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 },
);

// Коли подання зникає:
controller.abort();
```

Кожна частина має відповідник у стандарті. `createElement()` створює елемент, який не має батька, доки його не вставлять. `textContent` кладе номер замовлення як текст, тож статус із кутовими дужками лишається статусом. `replaceChildren()` прибирає старі пункти й вставляє нові за одну операцію. Один обробник на списку обслуговує всі пункти, бо клацання по пункту спливає до списку, а `closest()` іде від того, по чому клацнули, до найближчого `li`. Переривання контролера знімає обробник, і сторінці не треба зберігати посилання на функцію.

Дерево пояснює й цілий клас дефектів, які знайшов аудит самого цього сайту. Під час перевірки доступності в серпні 2026 року сайт проходив свої автоматичні правила, а проте мав одинадцять дефектів, і кожен стосувався того, як сторінка відображається, а не того, що написано в розмітці. Вісім посилань у футері лишалися в порядку переходу клавішею Tab за `pointer-events`, властивістю, яка прибирає лише взаємодію з вказівником і нічого більше. Для всього, що читало документ, це були звичайні посилання; вказівник до них не діставав, а клавіатура зупинялася на кожному. Анімація, прив’язана до прокручування, обрізала футер на трьох сторінках. Серед чотирьох правил, які залишив по собі аудит, одне звучить як застереження не покладатися на саме дерево: «видимість треба доводити в пікселях, а не в документі».

Де виникають проблеми

Цикл по живій колекції пропускає кожен другий елемент. HTMLCollection від `getElementsByClassName()` жива, тож цикл, який за індексом знімає клас із кожного елемента, зменшує колекцію просто під собою й перескакує через кожен другий. Обходьте натомість статичний список: його повертає `querySelectorAll()`, а `Array.from()` заздалегідь перетворює живу колекцію на звичайний масив.

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

for (const el of document.querySelectorAll(".pending")) {
  el.classList.remove("pending"); // статичний список: усі
}
```

Обробники накопичуються. Подання, яке додає свої обробники під час кожного рендерингу й ніколи їх не знімає, запускає кожен обробник стільки разів, скільки було рендерингів: після трьох відвідин одне клацання надсилає три запити. Щоб зняти обробник вручну, потрібен той самий об’єкт функції, що його додали, а стрілкова функція, написана просто у виклику, не лишає нічого, що можна передати назад. Дайте кожному поданню власний `AbortController` і передавайте його `signal`, як у прикладі вище, тоді один виклик `abort()` зніме всі обробники подання. Для обробника, який має спрацювати лише раз, є `once: true`.

Текст потрапляє всередину як розмітка. Якщо зібрати з даних рядок HTML і присвоїти його `innerHTML`, дані йдуть прямо до парсера, і за замовчуванням ніщо їх не очищує. Ім’я користувача з розміткою всередині, скажімо з тегом зображення й обробником помилки, з’являється на сторінці справжнім елементом разом з обробником. Місце тексту — у `textContent` або в текстовому вузлі, створеному через `createTextNode()`. Якщо розмітка справді мусить прийти з рядка, стандарт HTML визначає `setHTML()`, який очищує її під час розбору й прибирає небезпечний вміст.

Спостерігач живить сам себе. `MutationObserver` повідомляє про зміни у вузлах, за якими стежить, а його функція зворотного виклику спрацьовує після кожної порції змін. Якщо вона сама пише в піддерево, за яким спостерігає, щоб додати лічильник чи виправити клас, то створює нові

записи й викликається знову, і сторінка може застрягти в циклі власного виробництва. Стежте лише за тим, що справді потрібно, через `attributeFilter` і за можливості без `subtree`, перевіряйте у функції, чи зміна вже на місці, перш ніж писати, або викликайте `disconnect()` перед записом і `observe()` після нього.

Що почитати далі

- DOM Standard — визначення WHATWG для вузлів, дерев, живих і статичних колекцій, подій, `AbortController` і `MutationObserver`.
- Parsing HTML documents — як HTML-парсер перетворює потік символів на об’єкт документа, разом з обробкою помилок розбору.
- Dynamic markup insertion — `innerHTML`, `setHTML()` і `setHTMLUnsafe()` та які з них очищують розмітку.
- Server and Client Components — пояснення гідратації від Next.js: HTML із сервера, а потім обробники подій, прикріплені до DOM.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке DOM?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/dom/