Engineer

JavaScript

мова, яку виконують браузери

Що це таке

JavaScript — динамічно типізована мова програмування, стандартизована під назвою ECMAScript, яку вміє виконувати кожен веббраузер і яка також працює на серверах і машинах збирання через середовища виконання на кшталт Node.js. Скорочено її називають JS. Щорічні редакції стандарту мають назви ES2015, ES2024 і так далі, старі номери (ES5, ES6) досі в ужитку, а «vanilla JavaScript» означає саму мову без фреймворку зверху. Окрім назви та фігурних дужок, із Java у неї мало спільного.

Першу версію написав Brendan Eich у Netscape, і вона вийшла разом із браузером Navigator 2.0. Ecma International ухвалила перше видання стандарту, ECMA-262, у червні 1997 року, а її комітет TC39 випускає нове видання щороку, починаючи з ECMAScript 2016. Пропозиція потрапляє до специфікації лише на етапі 4 процесу комітету, а цей етап вимагає реалізацій, які вже випущено, тож текст і рушії рідко розходяться надовго.

Сама мова майже нічого не каже про зовнішній світ. ECMA-262 визначає синтаксис, типи, об’єкти, проміси та модулі, а введення й виведення залишає середовищу, що її виконує, — у стандарті воно зветься хостом. У браузері хост — це вебплатформа: документ, події, таймери й мережеві запити, описані здебільшого в HTML Standard та споріднених стандартах. У Node.js хост натомість дає файли, сокети й процеси. Цей поділ пояснює типовий сюрприз: код може бути цілком коректним JavaScript і все одно зламатися, переїхавши зі сторінки на сервер, бо мова переїхала, а хост лишився на місці.

У браузері скрипти сторінки ділять один головний потік із розкладкою, малюванням і введенням. Завдання виконується до кінця, перш ніж браузер зможе зреагувати на наступний клік. Проміси та функції `async` виносять

очікування за межі потоку, але обчислення лишаються на ньому, якщо їх не перенести у Web Worker. Числа — це значення IEEE 754 подвійної точності, тож цілі числа точні лише до 2^53 – 1, а все більше потребує `BigInt` .

Застосуйте поняття до рішення

Застосуйте «Що таке JavaScript?» 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

Справжнє рішення полягає в тому, скільки його надсилати людям, які читають сайт. JavaScript — це мова, завдяки якій сторінка реагує без звернення до сервера, і кожен його кілобайт треба завантажити, розібрати, скомпілювати й виконати на пристрої, яким ви не керуєте. Бандл, що здається миттєвим на ноутбуці розробника, може на кілька секунд зайняти телефон середнього класу, і платить за це читач — батареєю та терпінням.

Скрипти — також найвразливіша частина сторінки. Якщо скрипт нападника виконується, він має права сторінки: може читати те, що вводить читач, і надсилати це туди, куди дозволено надсилати самій сторінці. Саме цей ризик покликана стримувати Content Security Policy. Відмова від постачання виконуваного JavaScript усуває власні браузерні скрипти сайту як поверхню атаки; вхідні дані, яким не можна довіряти, усе одно потребують безпечного оброблення, а сторінка — обмежувальної політики. Екосистема пакетів створює ще одну поверхню атаки. Типове фронтенд-збирання тягне за собою сотні транзитивних залежностей, і кожна з них — це код, що виконується з тими самими правами, що й ваш власний.

Інший бік рахунку теж реальний. Коли одна мова працює по обидва боки системи, браузер і сервер можуть спільно використовувати правило валідації чи форму даних, а деяким продуктам скриптів треба багато: офлайн-стан, перетягування, мапи, спільна робота в реальному часі. Тож корисне питання ставлять для кожної функції окремо: чи має це відбуватися в браузері читача без запиту до сервера? Якщо ні, нехай HTML зробить сервер або збирання, і віддавати варто саме його.

Пов’язаний вибір — JavaScript чи TypeScript. TypeScript додає перевірку типів і крок збирання, а компілюється назад у звичайний JavaScript. У кодовій базі, яку щотижня змінюють кілька людей, перевірка типів зазвичай окупається; для покращення на сотню рядків на статичній сторінці вистачить JavaScript і суворого лінтера.

Як це виглядає на практиці

Цей сайт постачає читачам статичні сторінки без виконуваного JavaScript. Код пошуку збережено, але вимкнено; JavaScript і далі працює в інструментах збирання та перевірки. Три сторінки показують колишній виняток для браузера й роботу поза ним.

Попередня версія трьома мовами не мала бекенду часу виконання й постачала один власний скрипт. Опис збірки Hugo документує цей вузький виняток: фільтр портфоліо, елементи керування яким залишалися прихованими, доки скрипт не запускався. Решта роботи виконувалася під час збирання у 108 файлах шаблонів, серед яких 53 partials і 18 shortcodes. Опис називає й тогочасну ціну вибору: інтерактивні можливості доводилося вирішувати під час збирання, а пошуковий індекс оцінили й відклали замість постачання. Ці числа й виняток стосуються попередньої реалізації, а не коду, який читач отримує тепер.

Той самий вміст публікується через Gemini і Gopher — два протоколи, у яких для скрипту немає місця взагалі. Одна збірка створила 777 документів у кожному форматі на зафіксованому етапі реалізації з того самого джерела, що й сторінки. Окреме порівняння збірок підтвердило нуль змінених байтів HTML під час додавання цих форматів виводу; результат не охоплює пізніших редакторських чи візуальних змін. Аргумент сайту в тому, що документові не потрібен мегабайт скриптів, аби його прочитали, а формат, який узагалі не здатен вмістити скрипт, показує це найпростіше.

Більшість JavaScript-коду цього сайту зосереджено в інструментах. На етапі реалізації, який описано в матеріалі за посиланням, перевірка якості сайту складалася з тридцяти двох скриптів і 10 242 рядків JavaScript — тоді найбільшого масиву коду в репозиторії. Певний час це був єдиний код, якого не торкалися ні форматувальник, ні лінтер, ні перевірка типів. Запровадження форматування й лінтингу для цього коду почалося з налаштування форматувальника під код: усталені для репозиторію чотири пробіли переписали б 2 173 рядки найбільшого скрипта перевірки, а

перевизначення на два пробіли для цих файлів залишило 38 рядків із реальними відхиленнями форматування. Далі лінтер виявив 13 зауважень, і всі 13 виправили, не приховавши жодного. Помилка у скрипті перевірки проявляється як перевірка, що проходить, коли мала б упасти, тому код самих перевірок обробляють так само, як вони обробляють усе інше.

Де виникають проблеми

Збої нижче випливають із того, як побудована мова, і для кожного є відомі ліки.

Сторінка намальована, але на дотики реагує із секундною затримкою. Довге завдання — гідрація великого бандла, розбір великої відповіді JSON чи сортування тисяч рядків — займає єдиний потік, що обробляє введення. Надсилайте менше коду для першої взаємодії: розбивайте бандл за маршрутами й відкладайте решту, діліть довгу роботу на шматки, що повертають керування циклу подій, а важкі обчислення переносьте у Web Worker. Вимірюйте на повільному телефоні, а не на ноутбуці, на якому код написали.

Ціни та ідентифікатори виходять трохи хибними. Кожне число тут двійкове, з рухомою комою, тож десяткові дроби округлюються, а дуже великі цілі зливаються із сусідніми. Типова жертва — 64-бітний ключ бази даних, надісланий як число в JSON:

```
0.1 + 0.2 === 0.3; // false
Number.MAX_SAFE_INTEGER; // 9007199254740991
9007199254740993 === 9007199254740992; // true
```

Зберігайте суми в цілих найменших одиницях, як-от копійках чи центах, надсилайте великі ідентифікатори як рядки, а там, де арифметика мусить бути точною, беріть `BigInt`.

Порівняння пропускає хибне значення. Оператор `==` перетворює операнди перед порівнянням, тож `"" == 0` і `"0" == false` обидва істинні. Використовуйте `===` скрізь і доручіть це правилу лінтера `eqeqeq`, щоб перетворення відбувалося лише тоді, коли код явно про нього просить.

Текст від користувача зрештою виконується як код. Присвоєння неперевіреного тексту в `innerHTML`, складання розмітки зі шматків рядків чи передавання введення в `eval` перетворює дані на скрипт. Записуйте текст через `textContent`, надсилайте Content Security Policy, що впускає скрипти лише за nonce або хешем, а список залежностей тримайте коротким і зафіксованим у lockfile, читаючи, що приносить кожне оновлення.

Код, що працює в браузері, падає на сервері. Мова переїхала, хост лишився: `window` і `document` існують лише в браузері, а Node.js читає файл як CommonJS або як ES-модуль залежно від налаштувань, яких браузер ніколи не бачить. Зосереджуйте виклики, специфічні для хоста, на межах середовищ виконання, явно задайте `"type": "module"` у `package.json` і запускайте кожну частину коду в тому середовищі, де вона справді працюватиме.

Що почитати далі

- ECMAScript Language Specification — жива чернетка ECMA-262, текст, який реалізує кожен рушій; її оновлюють щоразу, коли пропозиції досягають етапу 4.
- The TC39 Process — етапи, які проходить пропозиція, перш ніж стати частиною мови, і що вимагає кожен із них.
- ECMA-262 в Ecma International — опубліковані щорічні видання, коли проєктові треба назвати версію, на яку він орієнтується.
- HTML Standard: event loops — як браузер планує завдання, мікрозавдання й рендеринг довкола скриптів сторінки.
- HTML Standard: the script element — класичні та модульні скрипти, а також як `async` і `defer` змінюють момент їхнього запуску.

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 JavaScript?» 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/javascript/