3. Браузер. Склад і загальні принципи роботи

Коли справа доходить до потужного та універсального програмного забезпечення, ніщо не може зрівнятися з веб-браузером, будь то машина Intel з архітектурою x86 або смартфон, що використовує мікрокод ARM. Веб-браузери пропонують феноменальну продуктивність на будь-якому пристрої. Щоб обробити та відобразити веб-сайт на екрані браузер обробляє мільйони чисел. Вони настільки потужні, що можуть замінити повноцінну операційну систему, і Chrome OS — яскравий приклад.

Основною функцією браузера є представлення вибраного веб-ресурсу, запитуючи його на сервері та відображаючи у вікні браузера. Ресурс зазвичай є документом HTML, але також може бути PDF, зображенням або іншим типом вмісту. Розташування ресурсу вказується користувачем за допомогою URI (уніфікований ідентифікатор ресурсу).

Те, як браузер інтерпретує та відображає файли HTML, визначено специфікаціями HTML і CSS. Ці специфікації підтримується організацією W3C (World Wide Web Consortium https://www.w3.org/), яка є організацією стандартів для Інтернету. Раніше браузери відповідали лише частині специфікацій і розробляли власні розширення. Це спричинило серйозні проблеми сумісності для веб-авторів. Сьогодні більшість браузерів більш-менш відповідають специфікаціям.

Інтерфейс користувача браузерів має багато спільного. Серед поширених елементів інтерфейсу користувача:

- Адресний рядок для вставки URI.
- Кнопки "назад" і "вперед".
- Параметри закладок.
- Кнопки оновлення та зупинки для оновлення або зупинки завантаження поточних документів.
- Кнопка "Домашня сторінка", яка переведе вас на домашню сторінку.

Як не дивно, інтерфейс користувача браузера не вказано в жодній офіційній специфікації, він просто походить від хороших практик, сформованих протягом багатьох років досвіду та браузерами, які наслідують один одного. Специфікація HTML5 не визначає елементи інтерфейсу користувача, які має мати браузер, але містить перелік деяких загальних елементів. Серед них адресний рядок, рядок стану та панель інструментів. Існують, звичайно, функції, унікальні для конкретного браузера, наприклад менеджер завантажень Firefox чи перекладач у Chrome.

Архітектура браузера

Зрозуміти роботу веб-браузера допоможе його високорівнева архітектура, що показана на рис.

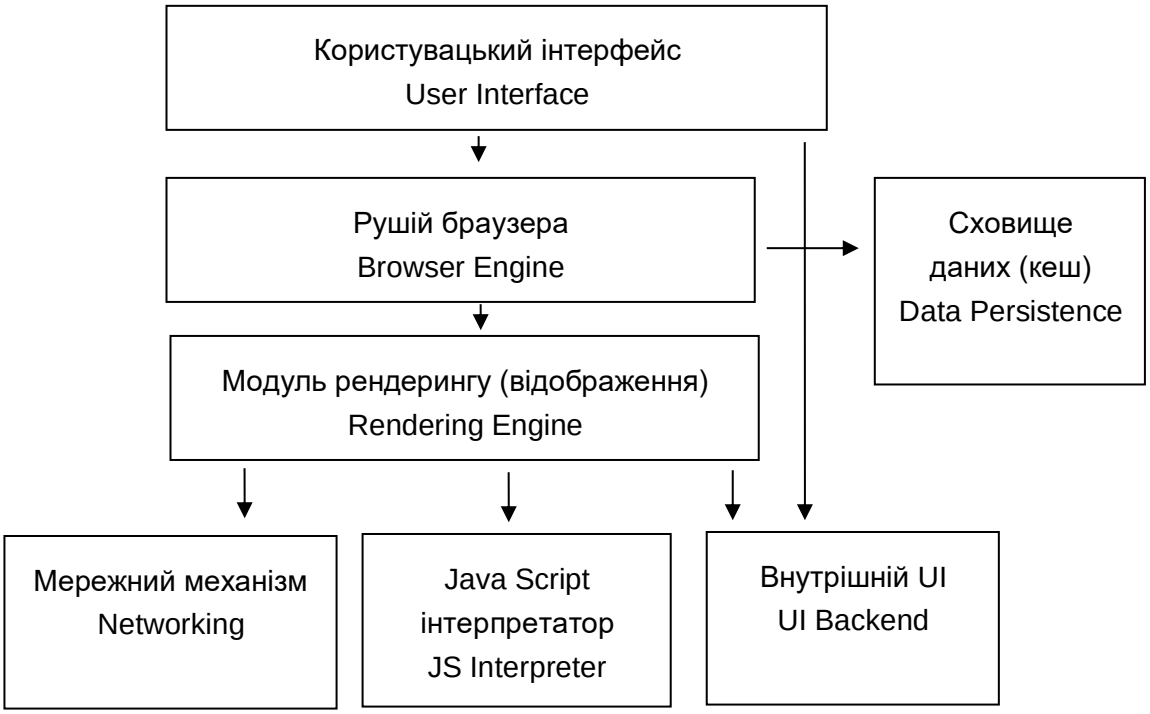


Рис.1. Архітектура браузера

- **User Interface.** Це все що бачить користувач: адресний рядок, кнопки вперед/назад, меню, закладки - за винятком області, де відображається сайт.
- **Browser Enginею** Рушій (движок) браузера, яка виступає посередником між інтерфейсом користувача та модулем рендерингу. Наприклад, клік по кнопці назад повинен сказати компоненту Rendering Engine, що потрібно відмалювати попередній стан.
- **Rendering Engine.** Модуль рендерингу, який відповідає за відображення запрошеного контенту. Залежно від типу файлу, цей компонент може парсити і рендерити HTML/XML, CSS, PDF. Різні браузери використовують різні механізми візуалізації: Internet Explorer використовує Trident, Firefox використовує Gecko, Safari використовує WebKit. Chrome і Opera використовують Blink, відгалуження WebKit.
- **Networking.** Реалізує механізм для мережних викликів, таких як HTTP-запити, з використанням різних реалізацій для різних платформ за незалежним від платформи інтерфейсом. Виконує XHR запити за ресурсами, і загалом, спілкування браузера з іншим інтернетом відбувається через цю компоненту, включаючи проксування, кешування тощо.
- **JS Interpreter.** Інтерпретатор JavaScript: використовується для аналізу та виконання коду JavaScript.
- **UI Backend.** Бекенд користувацького інтерфейсу. Надає загальний інтерфейс, не залежний від платформи. Використовується для малювання стандартних компонентів типу чекбоксів, інпутів, кнопок.
- **Data Persistence.** Сховище даних браузера для локального збереження всіх видів даних, наприклад файлів cookie. Браузери також підтримують такі механізми зберігання, як localStorage, IndexedDB, WebSQL і FileSystem.

Принцип роботи браузера

Всі дані, що передаються веб-додатком, проходять шлях між усіма рівнями моделі TCP/IP, інколи кілька разів (в залежності від кількості посередників у мережі). В попередній лекції було надано роз'яснення проходження HTTP-запиту від клієнта до сервера і отримання відповіді.

Нижче показано наступні дії браузера щодо відображення отриманої відповіді у прийнятному для користувача вигляді.

1. Отримання сторінки

Після того, як підключення встановлено, браузер може отримувати ресурси завантаженої сторінки, починаючи з отримання документа розмітки для сторінки. Для отримання сторінки виконується HTTP-запит GET.

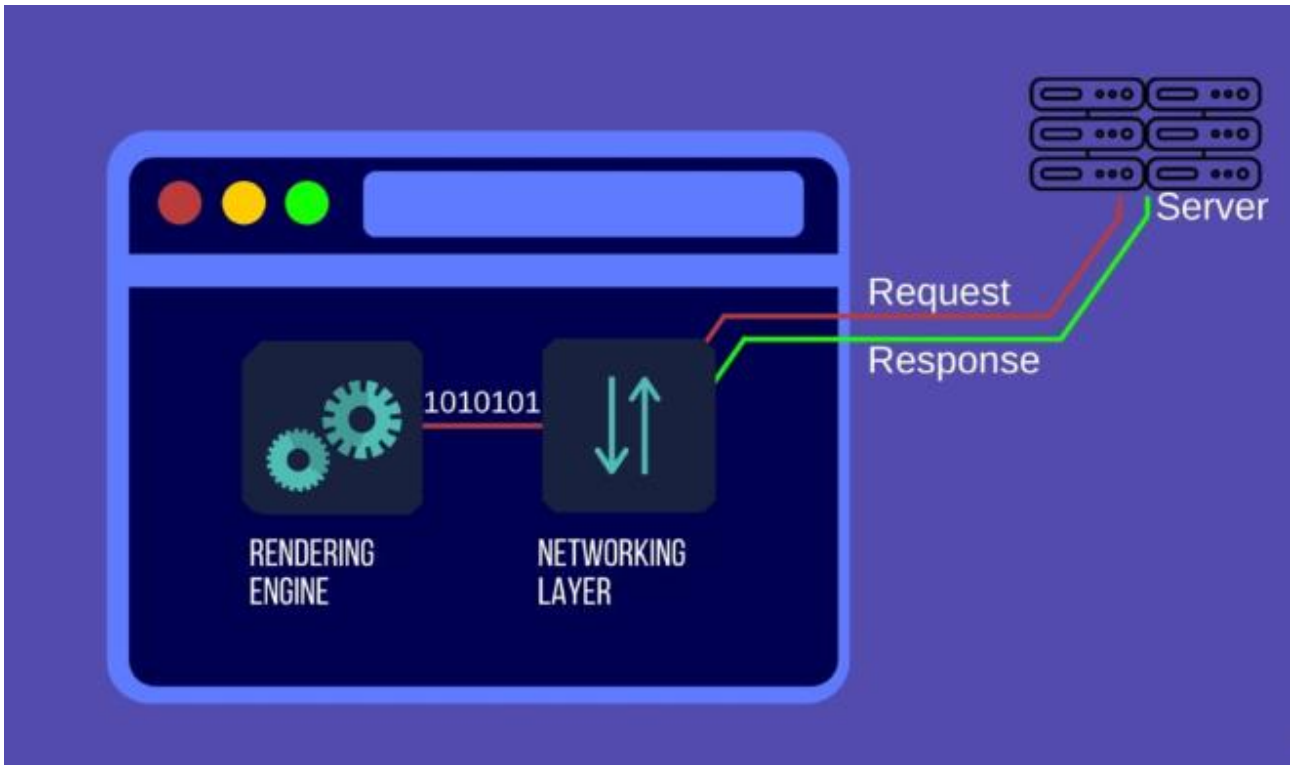


Рис.2. Отримання відповіді від сервера

GET запитує інформацію із заданого сервера, використовуючи уніфікований ідентифікатор ресурсу (URI). Специфікація правильних реалізацій методу GET лише витягує дані і не викликає змін у вихідному стані. Незалежно від того, скільки разів користувач запитує один і той же ресурс, зміни стану не відбувається.

Як тільки веб-сервер отримає запит, він його проаналізує і спробує виконати. Припустимо, що запит дійсний і файли доступні. Тоді сервер надсилає HTTP-відповідь, прикріпивши відповідні заголовки та вміст запрошеного HTML-документа у вигляді одиниць та нулів каналом зв'язку, встановленим мережним рівнем.

У браузері є ресурси, необхідні для відображення веб-сторінки, але відповідь від сервера представлена у вигляді байтів і має бути перетворена на формат, який виглядає як веб-сторінка.

Отже, за допомогою компонента Network браузер почав отримувати HTML-файл пакетами зазвичай по 8кб, а далі йде процес парсингу (специфікація процесу) і рендерингу цього файлу в компоненті Rendering Engine.

2. Отримання сенсу з бітів за допомогою механізму рендерингу

Основним завданням механізму рендерингу є перетворення біт даних у форму, яка може бути використана браузером для створення веб-сторінки. Щоб зрозуміти, як працює механізм рендерингу, важливо розуміти всі частини, які містяться у отриманій веб-сторінці.

- HTML (мова гіпертекстової розмітки) використовується визначення структури веб-сторінки.
- CSS (каскадні таблиці стилів) використовуються для вказування браузеру, як має виглядати кожен елемент на веб-сайті.
- Javascript використовується для додавання інтерактивності сайту і використовується для обробки дій користувача (кліки, пересування курсору мишки, вибір певного поля у формі тощо).

Для візуалізації веб-сторінки механізм рендерингу використовує синтаксичні аналізатори (парсери) для перетворення бітів даних на інформацію.

3. Розбір HTML

Механізм рендерингу має два різні парсери: один для HTML і один для CSS. Для підвищення юзабіліті, браузер не чекає, поки завантажиться і розпариться весь html. Натомість браузер відразу намагається відобразити користувачеві сторінку.

Нижче роз’яснено, як працює аналізатор HTML, щоб отримати уявлення про процес синтаксичного аналізу.

Аналізатор HTML приймає біти даних як вхідні дані та створює логічне подання документа HTML у пам'яті пристрою. Це логічне представлення даних відоме як структура (дерево) DOM і представляє HTML-дані в ієрархічному порядку.

- **Об'єктна модель документа (DOM, Document Object Model)** - це внутрішнє представлення об'єктів, які складають структуру та містять розмітку документа (у разі HTML), який отримано браузером. Він представляє собою сторінку, тому програми можуть змінювати структуру, стиль і зміст документа.

Дерево DOM описує вміст документа. В ньому вказуються взаємозв'язки та ієрархія різних елементів сторінки. Елементи, які розташовані всередині інших елементів, називаються «дочірніми» вузлами. Чім більше вузлів DOM, тим довше відбувається побудова дерева.

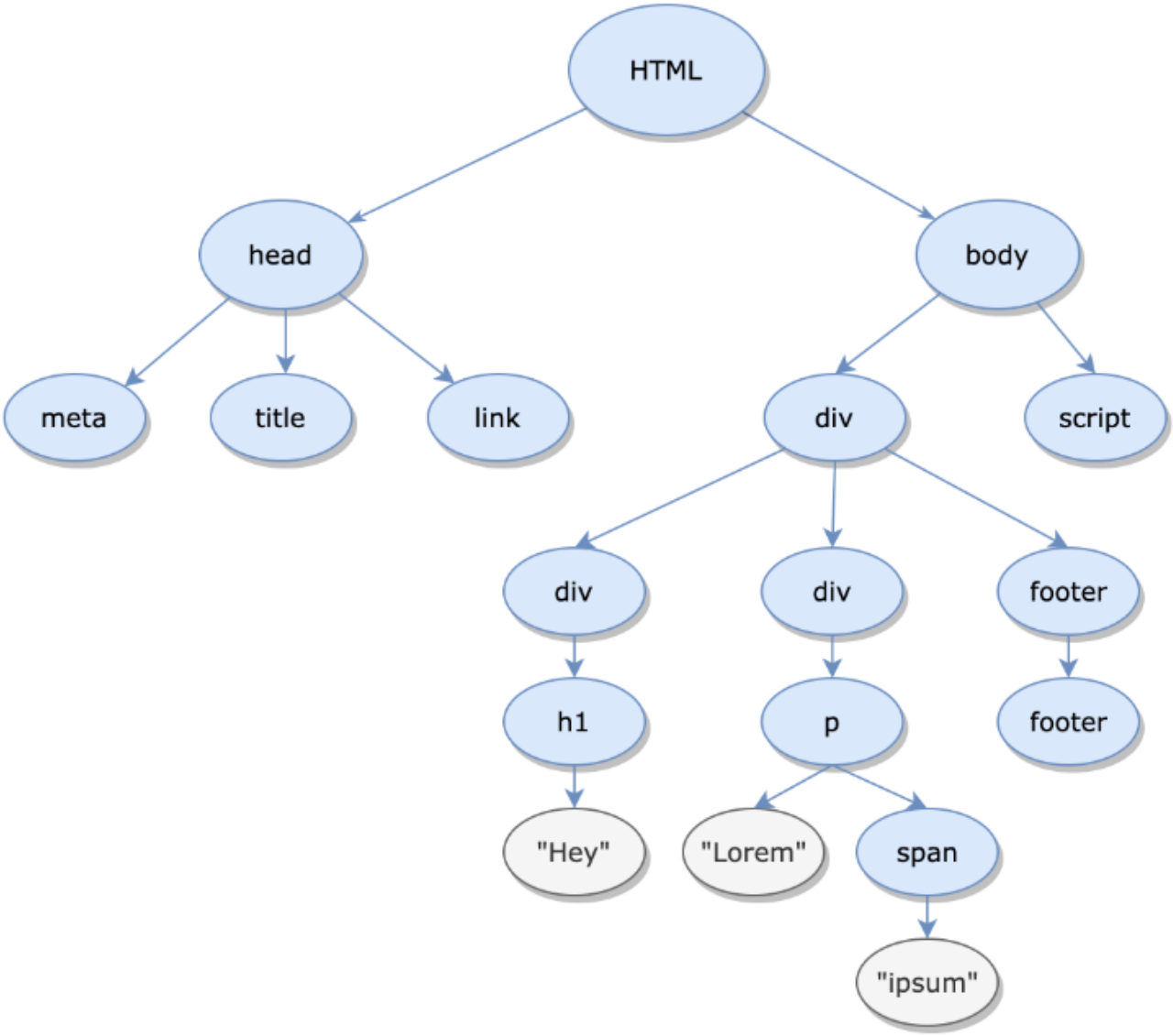


Рис.3. Дерево DOM для HTML-сторінки

Щоб створити структуру DOM, HTML-сервер виконує кілька кроків, які можна описати наступним чином.

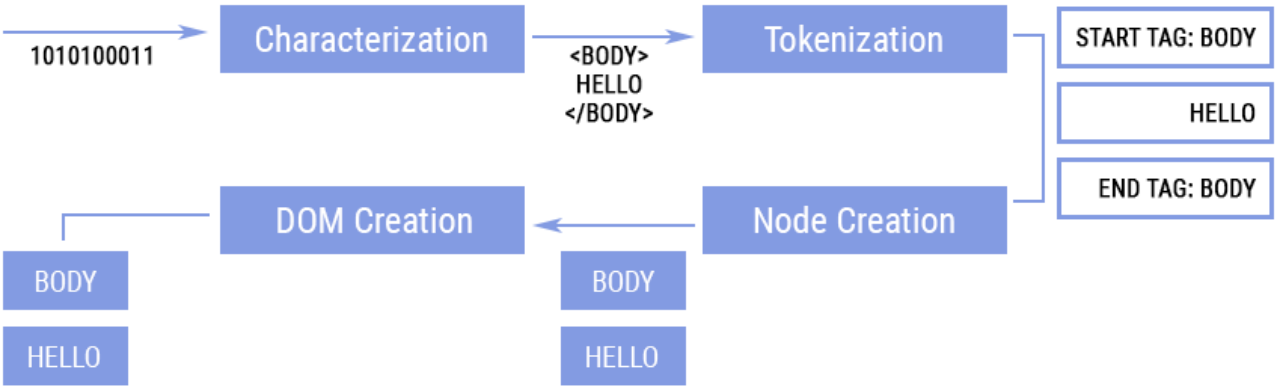


Рис.4. Розбір даних для створення структури DOM

- **Характеризація.** Браузер зчитує необроблені байти HTML з диска чи мережі та перетворює їх на окремі символи на основі вказаного кодування файлу (наприклад, UTF-8).

- **Токенізація.** Браузер перетворює рядки символів в окремі токени, як визначено стандартом W3C HTML5, наприклад, «<html>», «<body>» — та інші рядки в кутових дужках. Кожен токен має особливе значення та власний набір правил.
- **Створення вузла.** Після визначення токенів і інформації, що в них міститься, браузер створює вузли пам'яті (об’єкти), які визначають властивості токенів та правила їх обробки.
- **Створення DOM.** Розмітка HTML визначає зв’язки між різними елементами (деякі елементи містяться в інших елементах), створені об’єкти зв’язуються в деревовидній структурі даних, яка також фіксує зв’язки «батько-нащадок», визначені в оригінальній розмітці: об’єкт HTML є батьківським об’єктом для body , body є батьківським об’єктом для абзацу і так далі.

Окрім HTML-документа браузер отримує й посилання на файли CSS. Ці посилання надсилаються до синтаксичного аналізатора CSS. Цей синтаксичний аналізатор створює виведення CSSOM (об'єктна модель CSS), яка визначає, як має бути стилізований кожен елемент DOM.

- Об'єктна модель CSS (CSSOM, CSS Object Model) - це набір API-інтерфейсів, що дозволяють маніпулювати CSS і JavaScript. Це такий же DOM, але для CSS, а не для HTML. Він дозволяє динамічно читати та змінювати стиль CSS. Він дуже схожий на DOM у вигляді дерева і буде використовуватися разом із DOM для формування дерева рендерингу, щоб браузер міг почати процес рендерингу.

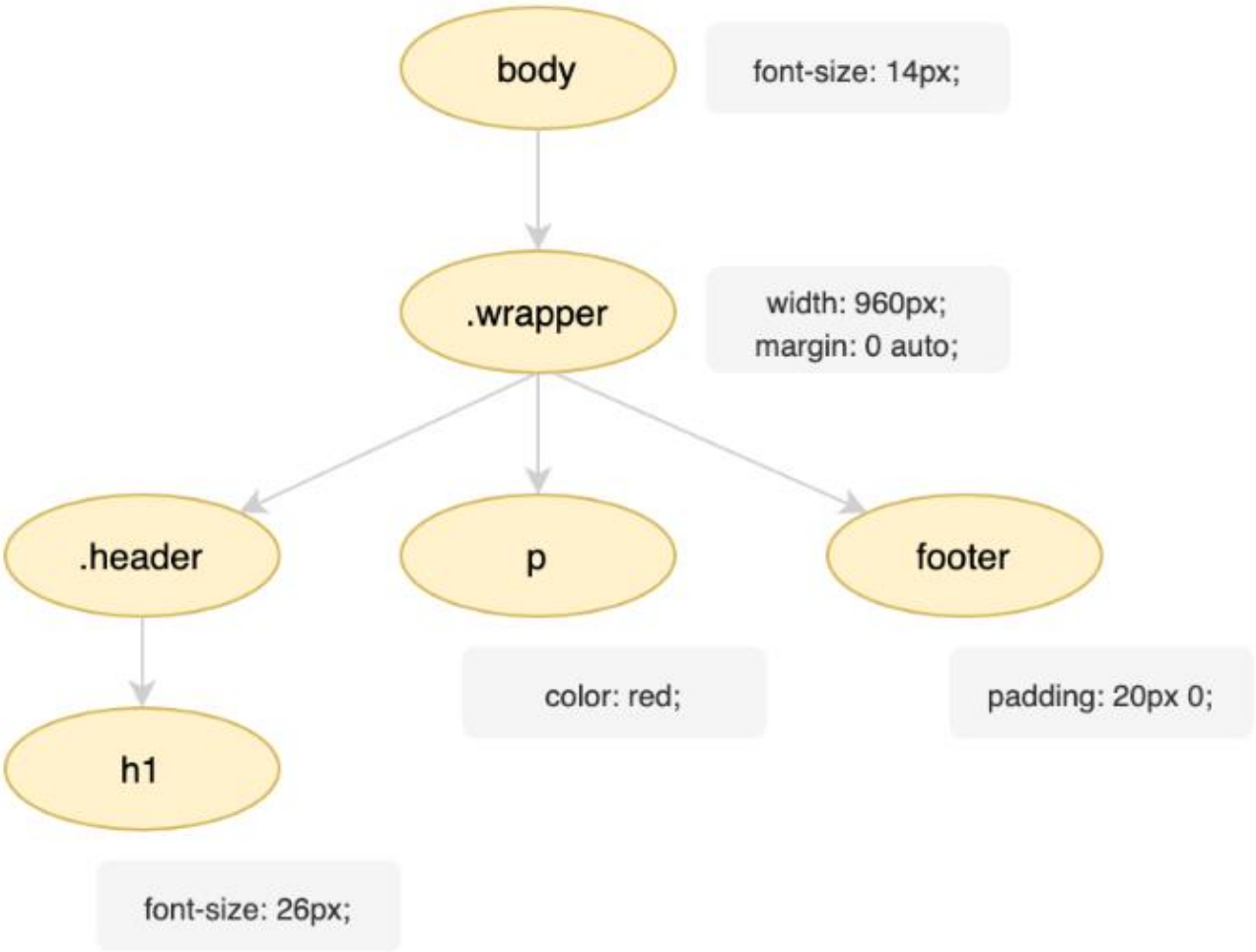


Рис.5. Об'єктна модель CSSOM

4. Створення дерева візуалізації та макету для веб-сторінки

Після створення моделі DOM і завершення синтаксичного аналізу файлу CSS механізм рендерингу використовує механізм стилів для об'єднання як CSSOM, так і DOM. Це створює дерево візуалізації, яке містить інформацію про структуру та стиль веб-сторінки, яка має відображатися. Дерево візуалізації складається лише з видимих вузлів і не має вузлів, які невидимі для користувача на екрані (з властивостями opacity та transparent).

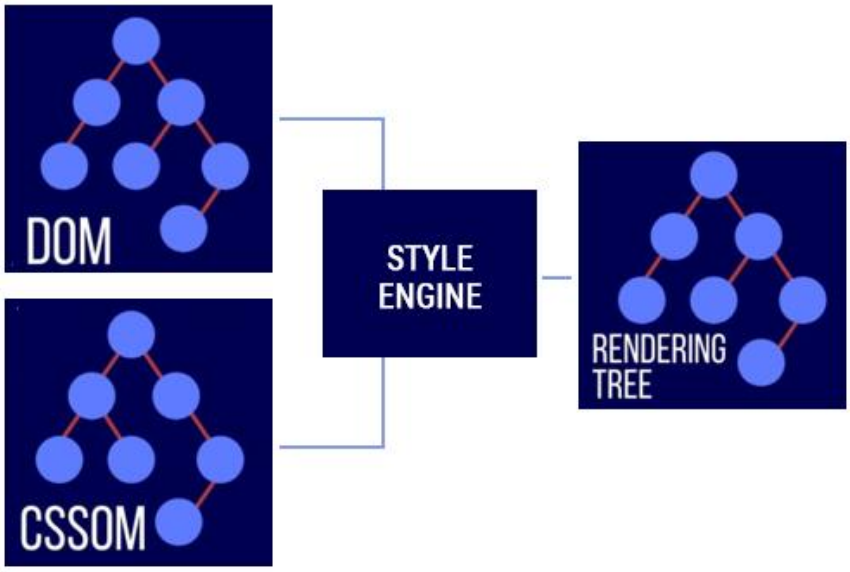
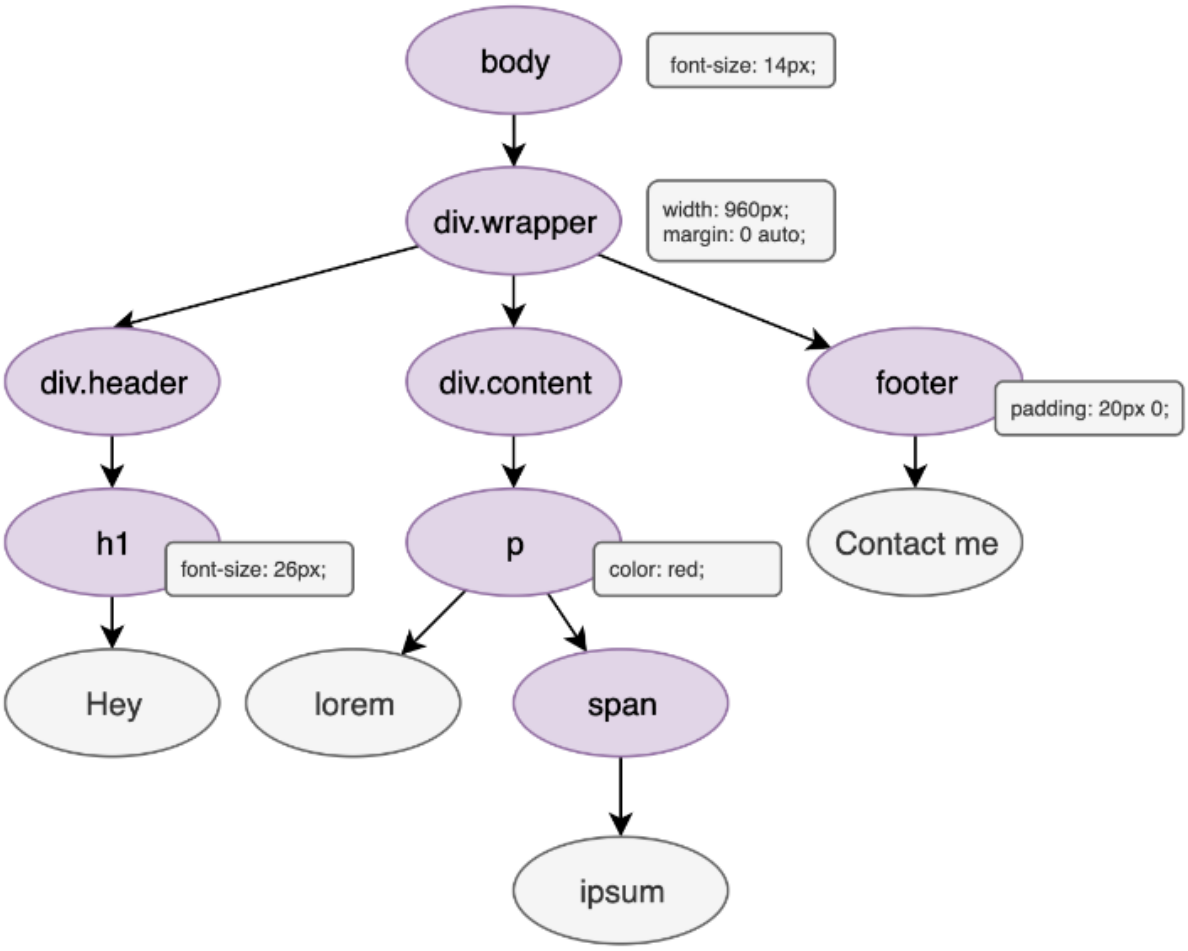


Рис.6. Об'єднання дерев DOM і CSSOM



Render Tree - дерево, що містить інформацію про те, що і як потрібно відмалювати. Тепер браузер повинен зрозуміти на якому місці та з якими розмірами відображатиметься елемент. Процес обчислення позиції та розмірів називається Layout .

Layout – це рекурсивний процес визначення положення та розмірів елементів із Render Tree. Він починається від кореневого Render Object, яким є body, і проходить рекурсивно вниз у частині або всій ієрархії дерева, вираховуючи геометричні розміри дочірніх render object'ів. Кореневий елемент має позицію (0,0) та його розміри дорівнюють розмірам видимої частини вікна, тобто розміру viewport'a.

В Html використовується потокова модель компонування (flow based layout), тобто геометричні розміри елементів у деяких випадках можна розрахувати за один прохід (якщо елементи, що зустрічаються в потоці пізніше, не впливають на позицію та розміри вже пройдених елементів).

Цей процес враховує роздільність екрана та те, як кожен елемент має бути розміщений на екрані пристрою. Він також обчислює розмір кожного елемента, який відображатиметься на екрані, та його відносне положення стосовно інших елементів.

Layout може бути глобальний, коли потрібно розрахувати положення render object'ів всього дерева, і інкрементальний, коли потрібно розрахувати лише частину дерева. Глобальний layout відбувається, наприклад, при зміні розмірів шрифту або при події resize'a. Інкрементальний layout відбувається лише для render object'ів, позначених як "dirty".

Пара слів про «систему брудних бітів (dirty bit system)». Ця система використовується браузерами для оптимізації процесу, щоб не перераховувати весь layout. При додаванні нового або зміні існуючого render object він сам і його дочірні елементи позначаються прапором dirty. Якщо render object не змінюється, але його дочірні елементи були змінені чи додані, цей render object позначається як «children are dirty».

До кінця процесу layout кожен render object має своє положення та розміри.

Коли движок рендерингу має всю інформацію про веб-сторінку у форматі, зрозумілому для системи, розпочинається рендеринг сторінки у браузері.

5. Візуалізація полотна та компонування веб-сторінки на екрані

Після того, як механізм рендерингу завершив процес макета, йому необхідно намалювати кожен піксель на екрані відповідно до макету, який було створено з використанням дерева візуалізації. Цей процес відомий як растеризація, тобто візуалізація екрану. Більшість браузерів використовують центральний процесор для виконання цього завдання. Для отримання кращих результатів частину растеризації можна передати до графічного процесора.

Браузеру потрібно обробляти все це дуже швидко. Цей етап може розподіляти елементи в дереві компоновки на шари. Розміщення вмісту на шарах у графічному процесорі (замість основного потоку на центральному процесорі) покращує продуктивність візуалізації та перемальовування. Шари підвищують продуктивність, але є високо затратними, коли справа доходить до управління пам'яттю, тому не слід зловживати ними в рамках стратегії оптимізації веб-додатку.

Багатошарова структура візуалізації допомагає браузеру швидше вносити зміни, коли користувач взаємодіє з веб-сторінкою.

Після того, як всі шари створені, механізм візуалізації надсилає цю інформацію до інтерфейсу користувача, відображаючи веб-сторінку на екрані. Цей процес відомий як відтворення веб-сторінки і є останнім кроком, який виконує механізм рендерингу.

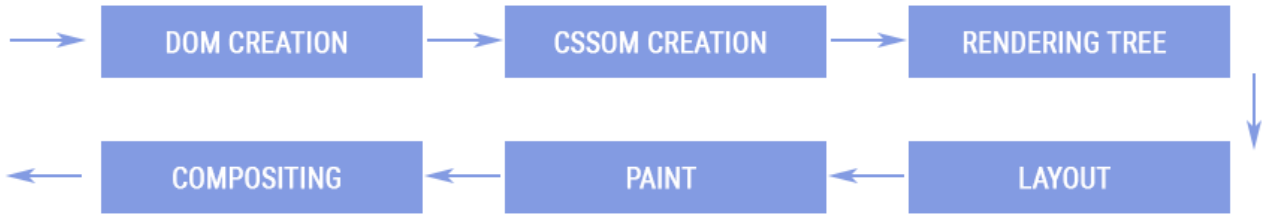
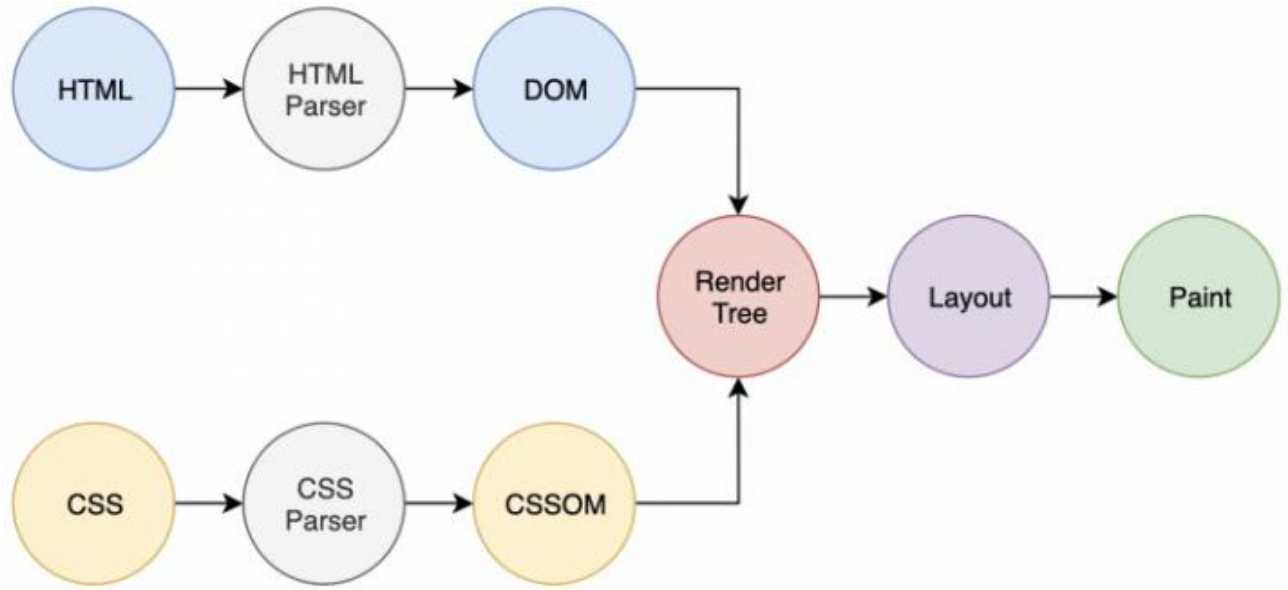


Рис.7. Багатошарова структура візуалізації сторінки

Процес відтворення веб-сторінки з бітів даних відомий як критичний шлях візуалізації і є основним фактором, який визначає продуктивність будь-якої веб-сторінки в Інтернеті.

Оптимізація критичного шляху візуалізації дозволяє прискорити початок рендерингу. Суть полягає в підвищенні швидкості завантаження сторінок для рахунок визначення пріоритетів завантажених ресурсів, контролю порядку їх завантаження та зменшення розмірів файлів цих ресурсів.

Резюмуючи сказане вище, отримуємо такий процес рендерингу веб сторінки:



Браузер отримує html файл, парсить його і будує DOM. Зустрічаючи css стилі, браузер їх підвантажує, парсить, будує CSSOM і об'єднує разом із DOM'ом – отримуємо Render Tree. Залишилося з'ясувати, де розмістити елементи з Render Tree — цим займається завдання layout. Після розташування елементів можна почати їх візуалізовувати — це завдання paint, етап на якому заповнюються пікселі екрану.

Після того, як механізм рендерингу відтворив веб-сторінку у вікні браузера наступає час для Javascript. Javascript є незалежним об'єктом, який відповідає за внесення змін до структури DOM, яка додає інтерактивність веб-сайту.

6. Додавання інтерактивності на веб-сайти за допомогою Javascript

Після того, як механізм рендерингу завершив відтворення веб-сторінки, але вона ще не є інтерактивною. Це означає, що якщо на веб-сторінці є кнопка, вона не працюватиме поки не підключиться Javascript. Javascript запускається віртуальною машиною у браузері, відомої як інтерпретатор Javascript.

Механізм Javascript і структура DOM не використовують одну пам'ять і є незалежними об'єктами. Інтерпретатор Javascript може взаємодіяти зі структурою DOM (створювати, змінювати, ігнорувати вузли дерева) і запускатись, коли на сторінці відбувається певна подія. Це допомагає браузеру відображати сторінки за допомогою механізму Javascript та відображати їх у разі виникнення події.

На сьогодні Javascript використовують практично у всіх веб-сайтах. Він відповідає за обробку даних, що вводяться користувачем та надсилає їх на віддалений сервер. Спроможність інтерпретувати код Javascript робить браузери універсальними клієнтами, дозволяючи веб-сайтам працювати однаково як на смартфоні, так і на комп'ютерах.

Javascript створено у 1996 році для браузера Netscape Navigator як мова сценаріїв, яку можна було інтерпретувати у браузері. Оскільки Javascript не створював машинний код для роботи на центральному процесорі, це робило мову надзвичайно універсальною.

На зорі Інтернету браузери відображали веб-сторінки, і при цьому не було задіяно багато Javascript. Віддалений сервер виконував більшу частину обробки, а движок Javascript мало що робив на веб-сторінці. Через це великий обсяг інформації повинен був передаватися між сервером та браузером, і така архітектура підходила для Інтернету, коли сторінки не були такими складними та інтерактивними.

У сучасному Інтернеті браузер і віддалений сервер повинні працювати злагоджено, щоб забезпечити кращий інтерфейс користувача. Сучасний браузер відповідає не лише за відображення сторінок, а й за обробку значної кількості даних за допомогою Javascript. У браузерах втілено JIT-компілятори, що зробило реалізацію Javascript сценаріїв дуже швидкими.

Робота рушія Javascript

Так само, як мережний рівень витягує HTML і CSS у вигляді байтів для механізму рендерингу, він також витягує код Javascript і передає його до інтерпретатора Javascript.

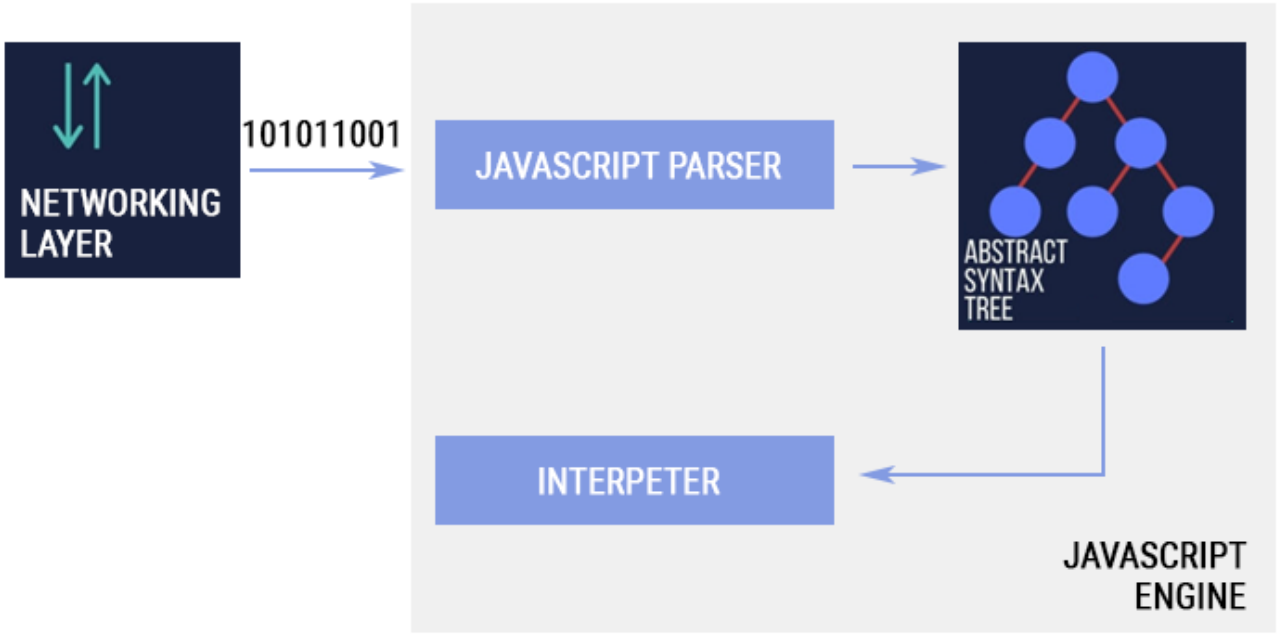


Рис.8. Робота рушія Javascript

Як тільки движок отримує код Javascript, він надсилає його до синтаксичного аналізатора, який створює абстрактне синтаксичне дерево (AST, Abstract Syntax Tree). Це дерево є логічним представленням коду JavaScript, який може бути запущений компілятором. Компілятор перетворює дерево на проміжну мову (байт-код), який може виконуватися інтерпретатором порядково.

Виконання Javascript використовується, коли код у скрипті не виконує повторювані завдання (наприклад, цикл). Якщо в коді Javascript є великі цикли, то движок намагається оптимізувати цей код і запустити його на центральному процесорі пристрою. Оскільки код виконується на центральному процесорі машини, він працює значно швидше порівняно з версією, що інтерпретується.

Для створення машинного коду механізм Javascript використовує оптимізуючий компілятор. Цей компілятор приймає байт-код, згенерований компілятором, і перетворює його на машинний код для конкретного пристрою.

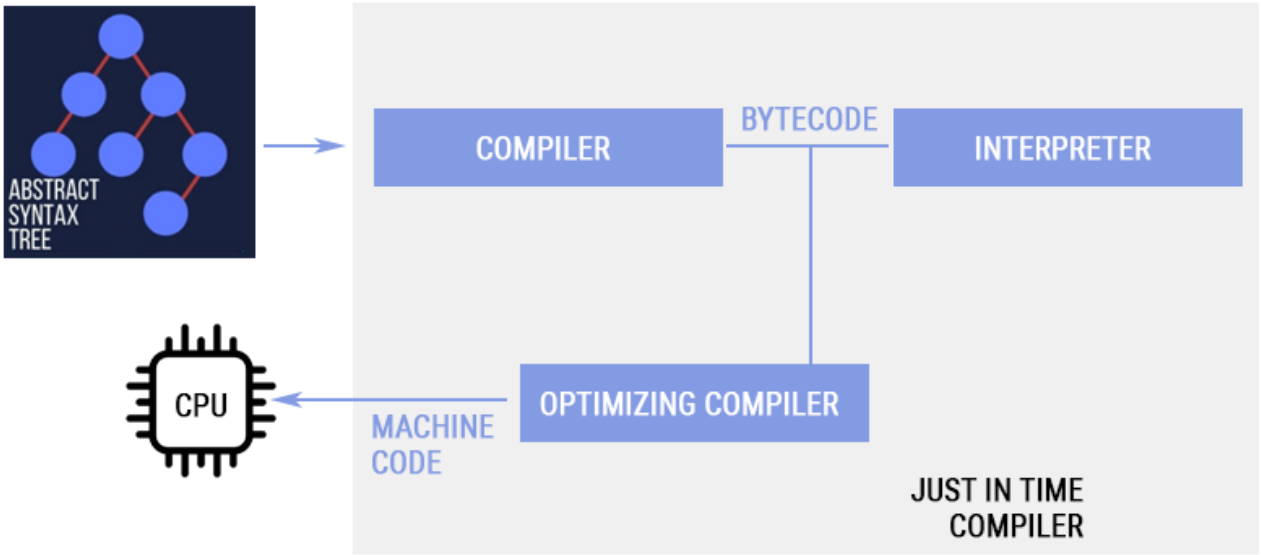


Рис.9. Процес запуску сценарію Javascript

Як тільки движок має оптимізований машинний код, він може запускати скрипт на неймовірно високій швидкості, використовуючи як процесор, так і інтерпретатор Javascript.

7. Повна інтерактивність сторінки

Після повного відображення сторінки, потрібно ще час на її завершення. Часто в таких оцінках оперують часом до відповіді користувачу (TTI, Time To Interactive) коли основний контент уже завантажився і користувач не лише бачить інтерактивні елементи, а й може повністю взаємодіяти з ними. Якщо браузер зайнятий побудовою дерев, завантаженням JavaScript і візуалізацією, він просто не зможе в цей же час відповісти на кліки користувача. Час всього TTI становить близько 50 мс.

Після завершення всіх кроків користувач може повністю побачити завантажену сторінку і працювати з нею.

Основні метрики швидкості завантаження сайту

Швидкість завантаження сайту — це важливий показник для SEO-просування, оскільки суттєво впливає на поведінковий фактор. Якщо сайт завантажується повільно, то користувач не буде довго чекати, а просто закриє вкладку і перейде до сайту конкурентів.

Але основна причина, чому варто моніторити та покращувати швидкість завантаження — це зручність для користувачів. Кожен власник веб-ресурсу хотів би, щоб клієнти знову поверталися на сайт, рекомендували його знайомим і почувалися комфортне.

TTFB (Time To First Byte) — проміжок часу від надсилання запиту на хостинг, на якому розміщено сайт, до першого байту надісланої у відповідь інформації. Між цими двома фазами відбувається чимало подій — перескерування, пошук DNS, узгодження по TLS та інші. Всі ці процеси лаконічно називають «обробкою запиту». TTFB — важлива для сайту метрика, яка має складати не більше 2-3 секунд.

FCP (First Contentful Paint) — час до того, як користувач побачить на сайті перший контент; первинний, частковий рендеринг. Ця метрика не враховує показник завантаження iframe, мається на увазі текст (включно з тим, до якого ще не підвантажилися шрифти), зображення або елементи canvas (полотно, виглядає як плашка не білого кольору). Оптимальний час — до 1.8 секунди.

LCP (Largest Contentful Paint) — це час, коли більшість контенту на видимій частині екрану вже доступна для перегляду. Хорошим показником вважається 2.5 секунди. Для розрахунку цієї метрики використовується час рендерингу найбільшого елементу (текстового блоку або зображення) від початку завантаження сайту.

FID (First Input Delay) — час від першої взаємодії користувача з інтерактивними елементами (натискання кнопки, введення тексту в поле, перехід за посиланням) до того моменту, коли браузер починає відповідати на них. Цей процес має відбуватися за 1 секунду або менше. Час очікування відповіді на дію також є важливою метрикою для репутації сайту, адже користувачеві не подобається, коли він натискає на кнопку, але нічого не відбувається.

TTI (Time To Interactive) — час до інтерактивності, коли основний контент уже підвантажився і користувач не лише бачить інтерактивні елементи, а й може повністю взаємодіяти з ними.

INP (Interaction to Next Paint) — показник швидкості відгуку сторінки на дії користувача, які відбуваються вже після повного завантаження сайту. Це проміжок часу, який проходить між кліком по елементу та відповіддю на нього. Оптимальною є швидкість реакції до 200 мілісекунд.

TTB (Total Blocking Time) — загальний час блокування, який проходить від початку рендерингу контенту (FCP) до інтерактивності сайту (TTI). У цей період користувач вже бачить наповнення, але ще не може натискати кнопки та інші активні елементи. Хорошим вважається показник в 300 мілісекунд.

CLS (Cumulative Layout Shift) — зміщення макета через підвантаження більш важких елементів. Уявімо, що користувач відкрив сайт, побачив потрібне посилання і клікнув по ньому, але в цей момент вище розкрився блок (відео або зображення), який довго вантажився. Через це макет змістився, посилання «поїхало» вниз, користувач клікнув замість нього по рекламі, і почалося завантаження сайту рекламодавця. Такі «стрибки» макету дратують, тому CLS вважається одним із головних показників, який аналізують всі основні інструменти вимірювання швидкості. У 2021 році було затверджено нові правила оцінювання цієї метрики, і тепер за основу береться часове вікно тривалістю до 5 секунд, коли відбуваються зміщення, з перервою менше ніж в 1 секунду.

Speed Index — демонструє швидкість відображення контенту в процесі завантаження. Наприклад, основний контент підвантажився на сайті за 4 секунди. Але в одному випадку протягом перших трьох секунд користувач бачить тільки білий екран, а у другому вже на першій секунді починається рендеринг дрібних деталей, заголовків, плашок. Ніби в обох випадках сайт завантажився за один і той же час, тож яка різниця, що відбувалося на екрані? Але для користувача краще бачити якийсь рух на сторінці, ніж просто чекати, споглядаючи статичне біле поле. Тому, навіть якщо в процесі з’являються не надто інформативні блоки, вони дають зрозуміти, що сайт вантажиться. Це позитивно впливає на користувацький досвід.

Core Web Vitals — це три основні метрики, які Google виділяє як найбільш важливі. До них відноситься рендерінг більшої частини контенту, час до першої інтерактивності та сукупне зміщення контенту (LCP, FID, CLS).

Висновки

Хоча браузери зараз дуже потужні, постійно з'являються інновації, які ще більше прискорюють перегляд. Одним із таких нововведень є веб-збірка, яка використовується з Javascript, щоб зробити виконання коду ще швидше за рахунок використання коду рівня збірки.

Браузери втілюють досягнення в галузі машинного навчання та штучного інтелекту. З такими бібліотеками, як Tensorflow, перехід на Javascript означає лише те, що браузери обов'язково стануть розумнішими у майбутньому; подальше покращення користувацького досвіду, який вони пропонують.

Джерела інформації

1. Принципи роботи веб–браузера https://freehost.com.ua/ukr/faq/articles/puteshestvie-veb-stranitsi-printsipi-raboti-veb---brauzera/

2. Браузер: загальні принципи роботи програми по відображенню сайтів https://ip-calculator.ru/blog/ask/brauzer-obshhie-printsipy-raboty-programmy-po-otobrazheniyu-sajtov/
3. Основні метрики швидкості завантаження сайту: TTFB, FCP, LCP та інші https://cityhost.ua/uk/blog/osnovnye-metriki-skorosti-zagruzki-sayta-ttfb-fcp-lcp-i-drugie.html
4. Як працюють браузери https://web.dev/howbrowserswork/#The_browser_high_level_structure
5. Рендеринг веб сайтів https://habr.com/ru/post/484900/