WebAssembly Essentials Code Reusable 2023

ПРОЛОГ

У світі розробки програмного забезпечення, що постійно розвивається, найважливіше бути в курсі найновіших інструментів і технологій. Це час, коли відмінності між веб-програмами та рідними програмами починають стиратися. Як розробники, інструменти, які є в нашому розпорядженні, можуть створити або зламати наші творіння. Ласкаво просимо до «WebAssembly Essentials», повного посібника для сучасних розробників, які прагнуть дослідити революційний світ WebAssembly (Wasm) і його величезний потенціал.

Історично в Інтернеті домінував JavaScript, мова, яку однаково люблять і ненавидять. Яким би універсальним не був JavaScript, він має обмеження, особливо коли мова йде про високопродуктивні програми. Увійдіть у WebAssembly — маяк надії для тих, хто прагне майже нативної продуктивності в Інтернеті. Ідея приголомшлива: взяти код із таких мов, як C++, Rust або навіть Python, і запустити його у браузері з неймовірною швидкістю. Це звучить амбітно, і справді, шлях до втілення цього в реальність був сповнений викликів та інновацій. Але плоди цих зусиль не що інше, як вражаючі.

Ця книга розгортається як ретельно підібрана детальна інструкція, призначена як для новачків, так і для досвідчених розробників. Вона представляє суміш фундаментальних концепцій, практичних ілюстрацій і передових стратегій. Вирушайте в подорож, яка починається з самої суті WebAssembly, поступово ведучи вас у глибші води, де ви навчитеся орієнтуватися в складнощах з впевненістю та досвідом. Значна частина нашого дослідження зосереджена на інструментах і їх майстерності. Ви заглибитесь у складну роботу таких інструментів, як Emscripten і Binaryen, які діють як потужні мости, полегшуючи перетворення коду з різних мов у Wasm. Але інструменти – це лише початок. Справжня магія полягає в розумінні безлічі методів перенесення додатків у формат Wasm, відкриваючи абсолютно новий потенціал веб-додатків. Будь то гра C++, модель машинного навчання Python або корпоративна програма на основі Java, обіцянка вивести їх у мережу з майже рідною продуктивністю тепер є реальною реальністю.

Але як щодо JavaScript? Не бійтеся, адже історія WebAssembly — це не історія заміни, а співпраця. Ми вирушаємо у світ прив'язок Wasm, ілюструючи, як JavaScript і WebAssembly можуть співіснувати та співпрацювати, забезпечуючи плавний зв'язок між модулями та підвищуючи універсальність ваших програм.

Особливу увагу приділяється оптимізації продуктивності — темі, яка резонує з кожним розробником. Від стратегій тонкого налаштування вашого коду до таких методів, як багатопотоковість, яка дозволяє виконувати паралельні операції, ми гарантуємо, що ваші програми не просто запускаються, а злітають угору. Ми також торкаємося таких концепцій, як відкладене завантаження та поділ коду, що забезпечує оптимальну швидкість реагування та ефективний час завантаження, що завершується чудовим користувацьким досвідом. Занурюючись глибше, ми досліджуємо нюанси керування пам'яттю в Wasm, надаючи розуміння ефективного використання ресурсів. Це доповнюється вичерпним детальним керівництвом щодо використання інструментів розробника браузера, що дає вам змогу розбирати, аналізувати та додатково підвищувати продуктивність ваших програм.

Нарешті, ми звернемося до аспекту, життєво важливого для сучасних веб-додатків: кешування. В епоху миттєвого задоволення забезпечення швидкого та ефективного доступу до веб-додатків має важливе значення. Завдяки ефективним стратегіям кешування, спеціально розробленим для Wasm, ми допоможемо вам покращити доступність і покращити взаємодію з користувачем. «Основи WebAssembly» — це більше, ніж просто книга; це подорож — подорож, яка обіцяє озброїти вас знаннями, інструментами та найкращими практиками, щоб перевизначити те, що можливо в Інтернеті. Отже, ми разом вирушимо в цю пригоду, занурюючись у світ WebAssembly, куди вабить майбутнє веб-розробки.

Веб-розробка до WebAssembly

У цю епоху інформаційних технологій Всесвітня павутина стала невід'ємною частиною нашого життя. Важливо зробити крок назад і отримати розуміння ландшафту веб-розробки, який існував до появи WebAssembly, перш ніж заглиблюватися в складність самої WebAssembly. Завдяки цьому ми можемо отримати повне розуміння революційного впливу WebAssembly на розробку сучасних веб-сайтів.

Статичні веб-сторінки, які в основному складалися з HTML і CSS, були тим, що визначило ранні етапи Всесвітньої павутини. Вони були зосереджені насамперед на передачі знань, але їм бракувало інтерактивних функцій, які стали стандартною практикою в наш час. Коли він був уперше випущений у середині 1990-х років, JavaScript швидко зарекомендував себе як основний інструмент для веб-розробників, які прагнуть створювати інтерактивні та динамічні веб-сайти. JavaScript зміг розширити свої можливості разом із розвитком браузерів, що, у свою чергу, дозволило створювати веб-додатки більш високого рівня складності.

Розквіт JavaScript

У зв'язку з розвитком JavaScript веб-програми почали ставати складнішими та наповнюватися різними функціями. Можливості JavaScript були додатково розширені завдяки таким інструментам і фреймворкам, як jQuery, Angular і React, які дозволили розробникам створювати складні односторінкові програми (SPA). Інтернет більше не використовувався виключно з метою споживання контенту; скоріше він перетворився на платформу для розгортання потужних програм, таких як графічні редактори та інструменти для співпраці в реальному часі. З іншого боку, зі збільшенням складності додатків зросла й видимість обмежень JavaScript. Було зрозуміло, що існують вузькі місця в продуктивності, особливо в обчислювально інтенсивній діяльності. Веб-розробники шукали шляхи для більш швидкого та ефективного виконання коду, що призвело до створення двигунів JavaScript, таких як Google V8. Незважаючи на ці вдосконалення, JavaScript усе ще був певним чином обмежений, головним чином тому, що він не був призначений із самого початку для обробки інтенсивної обчислювальної роботи, яка потрібна для сучасних веб-додатків.

Приблизно в той же час зростав інтерес до виведення в Інтернет програм, які спочатку не були написані для веб-середовища. Зокрема, цей інтерес був зосереджений на виведенні настільних програм у Інтернет. Це включало комп'ютерне моделювання, відеоігри та програми, які сильно покладаються на графіку. Мета розробників полягала в тому, щоб скористатися перевагами величезного потенціалу Інтернету без необхідності повністю переписувати свої програми на JavaScript. Такі інструменти, як Adobe Flash і Java-аплети, намагалися заповнити цю прогалину, але вони страждали від проблем з продуктивністю та безпекою, і врешті-решт зазнали

невдачі. Найсуттєвіша перешкода полягала в тому, щоб код запускався в браузері зі швидкістю, близькою до швидкості основного апаратного забезпечення, зберігаючи всю його потужність. Програми, написані такими мовами, як C, C++ і Rust, зможуть працювати без будь-яких проблем у веб-середовищі, якщо це станеться. Крім того, Інтернет потребував двійкового формату, який, на відміну від текстового JavaScript, дозволив би досягти значно швидшого часу завантаження.

Щоб підвищити продуктивність, Mozilla розробила ASM.js, підмножину JavaScript, яка давала розробникам можливість писати код, який виконуватиметься зі швидкістю, майже ідентичною швидкості рідної платформи. Це було першим свідченням того, що мережа готова вийти за межі обмежень, які накладає JavaScript. Водночас Google анонсувала Portable Native Client, або PNaCl, з наміром запускати код C і C++ у браузері безпечним і ефективним способом. Тим не менш, незважаючи на те, що обидва ці рішення були багатообіцяючими, кожне з них мало певні обмеження та не могло отримати підтримку всіх основних веб-браузерів.

Бачення єдиного веб-стандарту

Труднощі та експерименти, які мали місце протягом попередніх років, викристалізувалися в кришталево очевидну потребу: всесвітня павутина потребувала універсального, ефективного та безпечного стандарту, який міг би забезпечувати роботу браузерів майже рідною. Йшлося не лише про вдосконалення JavaScript; це також стосувалося переосмислення можливостей того, на що здатна веб-платформа в цілому. WebAssembly виник як прямий результат зближення досвіду, потреб та ідей багатьох різних людей. На цьому тлі WebAssembly виник не як проста заміна JavaScript, а як додатковий інструмент, який розширює можливості веб-розробки. Продовжуючи цю книгу, ми дізнаємося більше про багато аспектів WebAssembly, зокрема про її архітектуру, можливості та революційний вплив, який вона мала на Інтернет, яким ми його знаємо.

Початок і еволюція WebAssembly

У міру того як цифровий гобелен Інтернету ставав складнішим, потреба у швидшій та ефективнішій обчислювальній платформі стала очевидною. Увійдіть у WebAssembly: це не просто відповідь на проблеми зростання

Інтернету, а далекоглядний стрибок уперед у тому, чого можуть досягти програми на основі браузера.

Народження WebAssembly

Навчання навколо WebAssembly, яку часто скорочують як WASM, почалося всерйоз близько 2015 року. Ці спільні зусилля, підтримані такими технологічними гігантами, як Mozilla, Microsoft, Google і Apple, були спрямовані на розробку нового бінарного формату для Інтернету. WebAssembly було задумано як низькорівневу віртуальну машину, яка запускатиме код зі швидкістю, майже рідною, відкриваючи Інтернет для інших мов, окрім JavaScript. На відміну від попередніх спроб ASM.js і PNaCl оптимізувати продуктивність Інтернету, дизайн WebAssembly не залежав від браузера та мови. Це означало, що він міг працювати в будь-якому браузері, дозволяючи розробникам писати код кількома мовами. У червні 2015 року під керівництвом W3C було сформовано групу спільноти WebAssembly, що стало офіційним кроком до реалізації цього бачення.

Філософія дизайну WebAssembly

WebAssembly було створено з урахуванням кількох ключових принципів:

Ефективність і швидкість: основним мотиватором було виконання коду швидше, ніж JavaScript, що дозволило інтенсивним програмам працювати безперебійно.

Безпека: враховуючи недоліки таких інструментів, як Flash, WebAssembly розроблено для забезпечення безпеки в середовищі ізольованого програмного середовища браузера.

Портативність і компактність: він мав бездоганно працювати на різних пристроях і платформах, а компактний двійковий формат забезпечував швидше завантаження.

Гнучкість: розроблений не для заміни, а для доповнення JavaScript, WebAssembly мав на меті працювати рука об руку з існуючими веб-технологіями.

До березня 2017 року WebAssembly досягла ключового моменту у своїй історії, отримавши статус MVP (Minimum Viable Product). Ця важлива віха означала, що основні елементи та функціональні можливості були достатньо

стабільними та надійними, щоб їх можна було застосовувати в більш широкому масштабі. Примітно, що підтримку MVP WebAssembly було розгорнуто в усіх основних веб-браузерах — Google Chrome, Mozilla Firefox, Apple Safari та Microsoft Edge. Це широке визнання ознаменувало нову еру веб-розробки, демонструючи довіру цих гігантів галузі до можливостей WebAssembly.

В основі MVP WebAssembly був бінарний формат wasm. Цей новаторський формат запропонував безліч переваг перед існуючими парадигмами. Серед них менші розміри файлів виділялися як вагома перевага, завдяки чому передача даних через мережу стала набагато ефективнішою. Крім того, двійковий формат wasm був розроблений для синтаксичного аналізу з шаленою швидкістю, значно випереджаючи час аналізу еквівалентних файлів JavaScript. Ці сукупні переваги усунули деякі з давніх вузьких місць у веб-продуктивності, пропонуючи ефективність і швидкість.

Спільнота розробників швидко визнала трансформаційний потенціал WebAssembly. Бажаючи досліджувати та експериментувати, розробники почали портувати старі застарілі програми в Інтернет. Те, що колись було складним завданням, пов’язаним із проблемами продуктивності та сумісністю, раптом стало значно легшим. WebAssembly запропонував міст між традиційною розробкою програмного забезпечення та сучасним Інтернетом, спрощуючи завдання перенесення складних програм у веб-середовище.

Ігрова індустрія, яка часто перебуває в авангарді впровадження нових технологій, однією з перших побачила величезний потенціал WebAssembly. Багато розробників ігор виявили, що можливості WebAssembly буквально змінили правила гри. Провідні ігрові движки, як-от Unity, швидко підхопили їх, оголосивши про офіційну підтримку WebAssembly. Цей крок ознаменував сейсмічний зсув, дозволивши розробникам ігор виводити в Інтернет високоякісні ігри з високою продуктивністю без бар’єрів, які раніше стояли на шляху. Завдяки підтримці Unity повідомлення було чітким: WebAssembly — це не просто теоретична концепція, а практичний інструмент, здатний революціонізувати цілі галузі.

Розширення горизонтів за межі MVP

Після першого випуску еволюція WebAssembly була зовсім не застоєм:

Потоки: щоб використовувати потужність багатоядерних процесорів, WebAssembly представила підтримку потоків, що робить його ще більш придатним для завдань, що потребують високої продуктивності.

Збирання сміття: досліджено інтеграцію зі збирачем сміття браузера, що дозволяє мовам, які покладаються на збирання сміття, краще взаємодіяти з WebAssembly.

Інтеграція модуля ES: це було спрямовано на спрощення взаємодії між модулями JavaScript і WebAssembly.

Потокова компіляція: для ще швидшого часу завантаження такі браузери, як Firefox, представили потокову компіляцію, коли браузер компілює код WebAssembly, завантажуючи його.

Розширення мовного ландшафту

Однією з найвідоміших особливостей WebAssembly є його здатність відкривати Інтернет кількома мовами. Спочатку C і C++ були основними мовами, які можна було скомпілювати до WebAssembly. Проте ландшафт швидко розширився. Rust, мова, відома своєю продуктивністю та функціями безпеки, стала улюбленим вибором для розробки WebAssembly. Такі мови, як Python, Go і навіть мови .NET, почали вивчати та пропонувати цілі компіляції WebAssembly.

Сьогодні WebAssembly є не новинкою, а основним інструментом веб-розробки. Його застосування широке:

Веб-ігри: Завдяки майже рідній продуктивності WebAssembly зробила революцію в іграх у браузері.

Освіта: завдяки WebAssembly платформи, які навчають програмуванню, тепер можуть запускати компілятори безпосередньо в браузері.

Редагування графіки та медіафайлів: такі завдання, як редагування зображень або відео, які раніше вважалися надто інтенсивними для Інтернету, тепер здійсненні.

Наукове моделювання: WebAssembly відіграв важливу роль у перенесенні складних симуляцій, які часто призначені для спеціального програмного забезпечення, у середовище браузера.

Майбутні перспективи:

Незважаючи на те, що WebAssembly досягла багато чого, її шлях ще далекий від завершення. Тривають зусилля для подальшого розширення його можливостей, від обробки важких настільних додатків до можливих програм на стороні сервера. Поле Web3, децентралізованого Інтернету та технології блокчейн, що розвивається, також бачить потенційну інтеграцію з WebAssembly.

Від свого створення як бачення швидшої та ефективнішої мережі до сучасних програм WebAssembly є втіленням спільних інновацій. Це підкреслює природу Інтернету, що постійно розвивається, нагадуючи розробникам і користувачам про безмежні можливості, які чекають попереду. Коли ми глибше заглибимося в цю книгу, ви подорожуватимете багатогранним світом WebAssembly, відкриваючи його тонкощі, інструменти та трансформаційний вплив.

WebAssembly і JavaScript

Хоча WebAssembly і JavaScript створені для гармонійного співіснування, вони принципово відрізняються в кількох аспектах, починаючи від походження й закінчуючи характеристиками продуктивності та варіантами використання. Цей аналіз має на меті з’ясувати відмінні характеристики, які відрізняють WebAssembly від JavaScript.

JavaScript був представлений у 1990-х роках як мова сценаріїв для Інтернету, спеціально розроблена для інтерпретації браузерами. Його основною функцією було додавання інтерактивності веб-сторінкам. З часом, з появою AJAX і різних фреймворків, JavaScript розвинувся, щоб виконувати складніші операції, і став основою сучасних веб-додатків. З іншого боку, WebAssembly було представлено набагато пізніше, приблизно в середині 2010-

х років. Його створення було мотивовано необхідністю усунути вузькі місця продуктивності, з якими JavaScript стикався в певних завданнях, що потребують інтенсивних обчислень. Хоча JavaScript базувався на тексті та інтерпретувався, WebAssembly був розроблений як низькорівневий двійковий формат, скомпільований заздалегідь.

Виконання та компіляція

Одна з найбільш виражених відмінностей полягає в тому, як вони обробляються:

JavaScript: це інтерпретована мова. Коли браузер стикається з JavaScript, він спочатку аналізує текст, потім компілює його в байт-код і, нарешті, інтерпретує або JIT (Just-In-Time) компілює його в машинний код, який потім виконується. Цей багатоетапний процес, особливо для великих сценаріїв, може призвести до затримок у виконанні.

WebAssembly: двійкові файли WASM є низькорівневим форматом, близьким до машинного коду. Після завантаження модуля .wasm браузер може швидко перетворити його на машинний код і виконати. Попередня компіляція гарантує уникнення накладних витрат на продуктивність, пов’язаних з інтерпретацією.

Гнучкість мови

JavaScript як мова є ексклюзивною для веб-платформи. Розробники зобов’язані дотримуватися його синтаксису та парадигм. Однак його величезна екосистема, підкріплена такими бібліотеками, як React, Angular і Vue, пропонує багато функціональних можливостей і універсальності. WebAssembly відрізняється тим, що не залежить від мови. Замість того, щоб бути мовою, це мета компіляції. Це означає, що розробники можуть писати код такими мовами, як C, C++, Rust тощо, а потім компілювати цей код у WebAssembly. Це відкриває безліч можливостей, особливо для розробників і команд, які мають досвід роботи з мовами, відмінними від JavaScript, і хочуть використовувати це для веб-додатків.

Управління пам'яттю

JavaScript використовує збір сміття для керування пам’яттю. Хоча це зручно та зменшує ймовірність витоку пам’яті, це може внести непередбачуваність у продуктивність, особливо під час роботи з програмами

реального часу. З іншого боку, WebAssembly надає розробникам явний контроль над пам'яттю. Хоча це вимагає глибшого розуміння та ретельного керування, це забезпечує більшу гнучкість, особливо в критичних для продуктивності сценаріях.

Безпека та пісочниця

Як JavaScript, так і WebAssembly працюють в рамках обмежень безпеки браузера, гарантуючи, що шкідливий код не може завдати шкоди системі користувача. Однак їхні підходи відрізняються:

JavaScript: він працює в движку JavaScript, який забезпечує механізм ізольованого програмного середовища. Незважаючи на надійність, складність величезних можливостей JavaScript може створити потенційні проблеми для безпеки.

WebAssembly: він працює в рамках стекової віртуальної машини. Це середовище навмисно розроблено таким чином, щоб бути низькорівневим і мінімалістичним, зменшуючи площу потенційної вразливості.

Взаємодія з веб-платформою

JavaScript має невід'ємну перевагу, коли справа доходить до взаємодії з веб-інтерфейсами API та об'єктною моделлю документа (DOM). З огляду на те, що він розроблений для Інтернету, його інтеграція є бездоганною. WebAssembly у своєму оригінальному дизайні не мала прямого доступу до DOM. Натомість для цього знадобилося посередництво через JavaScript. Однак із прогресом тривають зусилля, щоб надати WebAssembly більш прямі способи взаємодії з веб-API без обов'язкового використання JavaScript.

Система набору тексту

JavaScript типізується динамічно. Змінні можуть містити значення будь-якого типу, а перевірка типу відбувається під час виконання. Хоча це забезпечує гнучкість, воно може викликати помилки виконання, якщо типи не відповідають очікуваним значенням. WebAssembly працює за іншою парадигмою. Він має статично типізовану систему, де типи визначаються під час компіляції. Це не тільки допомагає оптимізувати продуктивність, але й забезпечує рівень безпеки типу.

Паралелізм

Сучасні веб-додатки, особливо ігрові або симуляційні платформи, виграють від паралелізму. JavaScript, будучи однопоточним, покладався на потоки Web Workers, але цей підхід має обмеження. WebAssembly запроваджує підтримку потоків, що в поєднанні з його характеристиками продуктивності може призвести до високочутливих і одночасних веб-додатків. Загалом, хоча WebAssembly і JavaScript можуть працювати в тандемі, вони задовольняють різні потреби та мають унікальні характеристики. Представлення WebAssembly не означає затьмарення JavaScript; скоріше це розширення можливостей, надаючи розробникам більше інструментів для створення наступного покоління веб-досвіду.

Вивчення екосистеми WebAssembly

Екосистема WebAssembly (WASM) є великою та багатовимірною, що пропонує безліч інструментів, фреймворків і ресурсів. Ця розгалужена екосистема є однією з важливих причин, чому WebAssembly отримала швидке впровадження та продовжує залишатися привабливою платформою для веб-розробників. Ми заглибимося в його тонкощі.

Цілі та мови компіляції

Серцем екосистеми WebAssembly є численні мови, які можна скомпілювати до WASM:

Rust: з часом Rust став переважною мовою для розробки WebAssembly. Завдяки гарантіям безпеки, функціям паралелізму та продуктивності, що відповідає C++, Rust пропонує комплексний набір інструментів для WASM. Інструмент wasm-bindgen полегшує зв’язок між Rust і JavaScript.

C і C++: враховуючи їх низькорівневий характер, C і C++ були одними з перших мов, які використовувалися як цілі компіляції для WebAssembly. Emscripten, видатний інструмент в екосистемі, дозволяє конвертувати кодові бази C/C++ у WebAssembly.

AssemblyScript: інтригуюче доповнення, AssemblyScript дозволяє розробникам писати мовою, дуже близькою до TypeScript. Його розроблено спеціально для WebAssembly, що робить його привабливим варіантом для тих, хто вже знайомий із надмножиною JavaScript.

Інші мови: ландшафт WebAssembly не обмежується вищезазначеним. Інші мови, такі як Kotlin, Go і навіть Python, поступово знаходять свій шлях до структури компіляції WebAssembly.

Інструменти та ланцюги інструментів

Інструменти в екосистемі WebAssembly забезпечують ефективну компіляцію, оптимізацію, налагодження та інтеграцію:

Emscripten: крім компіляції C/C++ до WebAssembly, Emscripten пропонує набір інструментів, включаючи емулятор файлової системи та API для інтеграції з веб-платформою.

Binaryen: це компілятор і бібліотека інфраструктури ланцюга інструментів для WebAssembly. Завдяки таким компонентам, як wasm-opt для оптимізації модуля, він відіграє вирішальну роль у вдосконаленні остаточних двійкових файлів WASM.

WABT (Набір бінарних інструментів WebAssembly): WABT містить набір утиліт, таких як wat2wasm і wasm2wat, що дозволяє розробникам конвертувати між текстовим представленням WebAssembly (WAT) і двійковим форматом (WASM).

WebAssembly Explorer: онлайн-інструмент, який дозволяє розробникам переглядати результуючий код WebAssembly із джерел C/C++, допомагаючи в розумінні та оптимізації.

Середовища виконання

Хоча браузери є основною базою для виконання WebAssembly, автономні середовища виконання розширюють його можливості за межі:

Wasmtime: автономне середовище виконання у стилі JIT для WebAssembly, Wasmtime дозволяє виконувати WASM поза браузером. Це особливо корисно для програм на стороні сервера.

Wasm3: Wasm3, який має звання найшвидшого інтерпретатора WebAssembly, можна розгортати в різних середовищах, включаючи вбудовані системи.

Lucet: розроблений Fastly, Lucet призначений для периферійних обчислень. Він може створити екземпляр і виконати модулі WebAssembly за лічені мікросекунди, що підходить для програм, чутливих до затримки.