Node.js Frameworks

Getting the Most out of Node.js Frameworks: CRC Press, 2022

Вступ до фреймворків JavaScript

Якщо у вас був досвід роботи з JavaScript (JS), ви, мабуть, уже стикалися з терміном «фреймворки JavaScript». Що це за фреймворки JS, про які всі говорять? І яку спеціальність вони привносять у розвиток?

Хоча сьогодні існує тенденція називати майже кожен інший проект «фреймворком», є певні ключові характеристики, які необхідні для того, щоб фрагмент коду називався «фреймворком». У світі JS не бракує фреймворків і бібліотек.

У міру проходження розділів цієї книги ви познайомитеся з деякими з найпопулярніших, інноваційних і спеціалізованих фреймворків JS. Ви дізнаєтесь про деталі, що стосуються інсталяції та налаштування, а також про найкращі практики та методи, щоб отримати максимальну віддачу від цих фреймворків JS.

З огляду на це, перш ніж йти далі, було б гарною ідеєю дізнатися трохи про фреймворки в цілому. Як згадувалося вище, іноді багато концепцій або бібліотек неправильно називають фреймворками JS.

Щоб уникнути подібних помилок, давайте трохи ознайомимося з тим, що таке фреймворки JS, що вони можуть робити (а що ні) і як вам варто їх використовувати.

У цій главі ми не лише розглянемо, що мається на увазі під фреймворками JS, але й розглянемо, чим такі фреймворки відрізняються від бібліотек, як вони використовуються, а також коли варто чи ні використовувати їх.

ЩО ТАКЕ ФРЕЙМВОРКИ JS?

По-перше, давайте почнемо наше обговорення з того, що з’ясуємо, що таке фреймворки JS.

Простіше кажучи, фреймворк JS — це об’єкт на рівні програми, який використовується для визначення або поєднання всього дизайну програми.

Збиває з пантелику? Сформулюйте це так: фреймворк JS — це набір компонентів і блоків коду, необхідних для виконання певного набору функцій і дій. Іншими словами, фреймворк JS — це тіло попередньо визначеного коду, який ви можете застосовувати або використовувати у своїх проектах, поєднуючи його з вашим власним кодом.

Таким чином, фреймворк JS усуває тягар з боку розробника, викликаючи його різні методи та компоненти для виконання певних функцій. Зверніть увагу, що тут код розробника не викликає фреймворк — це навпаки. Фреймворк JS окреслює керівні концепції для певного типу програми (скажімо, односторінкової веб-програми), і фактичний код може бути змодельований навколо того самого.

Природно, залежно від рівня, складності та кількості функцій, які може обробляти даний фреймворк JS, він може бути дійсно великим або досить маленьким фреймворком, призначеним для дуже конкретного нішевого завдання. Менші або менш загальні фреймворки JS часто називають бібліотеками — ключова відмінність, яку ми розглянемо пізніше в цьому розділі.

Тепер, щоб краще зрозуміти, як функціонує фреймворк JS, подумайте про це: скажімо, ви бачите, що у нас є власний код, призначений для виконання певної ролі. Однак він не може функціонувати в своєму поточному стані, оскільки йому потрібні додаткові функції, залежності, бібліотеки тощо. У нас є два варіанти: ми можемо написати все це самостійно та заощадити години й дні, працюючи над ним, або ми можемо скористатися готовим Фреймворк JS, який містить попередньо визначений набір коду, який виконуватиме залежні дії, необхідні для функціонування нашого коду. Поєднавши фреймворк JS із попередньо написаним кодом, ми можемо легко виконати роботу та заощадити багато часу на кодування.

ЧИМ ФРЕЙМВОРКИ JS ВІДРІЗНЯЮТЬСЯ ВІД БІБЛІОТЕК?

Тепер, коли ми обговорили, що таке фреймворки JS, настав час звернути нашу увагу на іншу сутність, яка часто (неправильно) використовується як взаємозамінні з фреймворками — бібліотеки JS.

Бібліотека — це набір інструментів і методів коду, які допомагають виконувати певні завдання. З іншого боку, фреймворк – це скоріше вказівка, яка вказує, як ви повинні представити свій код. Фреймворк зазвичай не викликається вашим кодом, натомість це фреймворк JS викликає код. Яку б програму ви не створювали, код структурно визначається фреймворком JS загалом.

Найчастіше багато програмістів вважають різницю між бібліотеками та фреймворками досить розмитою. Однак насправді це не так вже й складно. Фреймворк JS підкаже вам, як має бути структурований ваш проект або програма. Бібліотека JS — це лише набір компонентів і блоків, які можна використовувати для виконання серії завдань у рамках зазначеного проекту чи програми.

Бібліотеки JS, як правило, досить легко включати в проекти. Незалежно від вашої методології кодування, ви можете працювати з бібліотеками JS, такими як jQuery, і з легкістю включати їх у свій проект. Однак фреймворки JS часто є вимогливими щодо реалізації та організації.

Як правило, краще використовувати фреймворк JS із самого початку проекту та моделювати свій проект навколо фреймворку, скажімо Angular. У той час як ви можете легко включити нову бібліотеку в існуючий проект, реалізація нової структури в існуючому проекті часто є складною справою і вимагає досить значних зусиль.

Таким чином, ми можемо сміливо стверджувати, що бібліотеки JS більш гнучкі, ніж фреймворки JS.

Тепер, щоб краще зрозуміти, як кожен з них може бути використаний у ваших проектах, розглянемо цю просту ілюстрацію: припустімо, що ми плануємо створити програму JS, яка діє як калькулятор і виконує основні математичні функції. Однак для кожного результату, скажімо, додавання чи віднімання, він показує анімацію під час відображення результату.

Як бачите, ваш код використовує бібліотеку JS і викликає її різні функції або методи для серії завдань, у цьому випадку анімації. Сама бібліотека міститься у фреймворку JS, який, у свою чергу, викликає ваш код для створення програми, у цьому випадку калькулятора. Це дуже спрощений і зрозумілий приклад, який допоможе вам зрозуміти, як і де фреймворки JS вписуються в загальну картину.

Щоб спростити речі, ми можемо сказати, що бібліотеки JS — це майже набір класів і методів. Найкраща і найочевидніша мета таких бібліотек — можливість повторного використання існуючого коду та економія часу та зусиль. Фреймворк, з іншого боку, надає скелет, за допомогою якого ваш додаток може визначати власні функції, які будуть викликані фреймворком за потреби.

Останнім часом певні бібліотеки JS, такі як Prototype, розвинулися до такого загального рівня використання, що їх об'єднують у фреймворки та бібліотеки. Ми звернемося до цієї концепції пізніше в цій книзі, коли обговорюватимемо прототип. Наразі буде корисно запам'ятати наступне:

• Бібліотека викликається кодом, тоді як фреймворк JS сам викликає код.

• Фреймворк JS — це напівзавершена сутність, яка забезпечує скелетну структуру для вашого коду, який базується на ньому, відповідно до певної мети чи потреби. Бібліотека, на відміну від фреймворку, — це набір функцій, які можуть використовуватися вашим проектом для виконання лише певного набору завдань.

ПЛЮСИ І МІНУСИ JS FRAMEWORKS

Зрозуміти або оцінити плюси та мінуси фреймворків JS досить легко, особливо якщо у вас є певний рівень досвіду розробки.

Хороша частина будь-якого фреймворку JS полягає в тому, що ви отримуєте міцну основу для початку. Більшість пристойних фреймворків JS, як правило, мають міцну та дуже надійну кодову базу, яка постійно знаходиться в стадії активної розробки. Виправлення помилок, оновлення безпеки та все інше доступне.

Це, природно, знімає багато головного болю з розвитку. Вам не потрібно турбуватися про те, щоб щось писати з нуля — у вас уже є міцна основа, яку забезпечує фреймворк JS, і ви можете будувати свій проект на його основі.

Окрім цього, фреймворки JS також мають різні видимі переваги. По-перше, більшість основних фреймворків JS, як правило, мають дуже лояльну та активну спільноту. Це означає, що якщо ви коли-небудь зіткнетеся з помилкою чи проблемою у своєму проекті, допомога від інших користувачів тієї самої JS-платформи завжди буде під рукою.

По-друге, нові оновлення та виправлення застосовуються до фреймворків JS досить швидко, особливо якщо спільнота активна. Це означає, що ваш проект стоїть на міцних плечах.

По-третє, загалом це економить час і легше кодувати за допомогою фреймворку JS, ніж без нього. Це може заощадити години та зусилля кодування, надаючи вам уже існуючу кодову базу для роботи.

Таким чином, фреймворки JS мають наступні переваги:

• ***Ефективне кодування***: це стало можливим завдяки існуванню надійної кодової бази та надійних стандартів кодування, які прагне забезпечити кожна пристойна структура JS.

• ***Заощаджує час/гроші/зусилля***: просто забезпечивши вам міцну основу для побудови вашого проекту, фреймворк може заощадити багато часу в довгостроковій перспективі.

• ***Краща безпека/оновлення***: спільнота ніколи не спить! Фреймворки JS, такі як React, Angular або VueJS, мають дуже лояльних користувачів, які завжди раді повідомити про проблеми та проблеми, допомогти новим користувачам і допомогти вам із будь-якими запитами, які у вас можуть виникнути!

• ***Чудова документація***: навіть якщо офіційної документації бракує, багато основних фреймворків JS мають дуже ефективну літературу, доступну в Інтернеті, яка знову може зробити навчання веселим.

Але чи означає це, що фреймворки JS не мають недоліків? Звичайно, ні. Залежно від ваших вимог і вподобань, іноді ви можете виявити, що фреймворкам JS бракує певних аспектів.

Почнемо з того, що будь-який фреймворк JS, яким би легким і зручним він не був, зазвичай працює повільніше порівняно з необробленим JS. Це означає, що ви так чи інакше йдете на компроміс щодо швидкості.

Але найбільший недолік, пов’язаний із фреймворками JS, полягає в тому, що міграція не завжди є найпростішою справою. Коли ви обираєте фреймворк JS для свого проекту, перехід на інший фреймворк у подальшому, безсумнівно, буде складним завданням, якщо не зовсім складним. Хоча здебільшого ви в безпеці і не потребуєте переходу з фреймворку JS, що, якщо згаданий фреймворк припинить розробку?

Зазвичай це відбувається з меншими та менш відомими фреймворками (а не з React чи Angular). Якщо спільнота або розробники втратять інтерес, виправлення помилок і патчі безпеки може бути важко знайти. Це означає, що ваш проект базуватиметься на застарілому коді — звичайно, логічним вибором буде перехід на інший фреймворк.

Таким чином, вибір фреймворку JS іноді може обмежувати багатьох розробників. Ось чому багато «професійних» кодерів, як правило, використовують підхід без фреймворків. Але, звичайно, це лише одна сторона історії. Часом вам потрібно заощадити час кодування. В інших випадках використання фреймворку JS просто означає, що ви не намагаєтесь винаходити колесо, а натомість обираєте стандартизований стандарт кодування, який забезпечить відповідність вашої програми найкращим практикам кодування. Те, що певне програмне забезпечення, скажімо Mac OS, може припинити розробку через десятиліття, не означає, що ви не можете використовувати його сьогодні. Технологія — це концепція, яка постійно розвивається, як і фреймворки JS.

З огляду на це, якщо ваш робочий процес почувається краще без фреймворку, можливо, вам варто використовувати фреймворки JS лише помірно або зрідка, і більше зосередитися на ванільному JS.

Почекай, vanilla JS? Що це? Давайте тепер звернемо нашу увагу на порівняння світу між фреймворком і безфреймворком (framework-less).

JS FRAMEWORKS ПРОТИ VANILLA JS

Що таке vanilla JS? Простими словами, це відноситься до простого, необробленого JS без активного фреймворку. Заради ясності можна навіть просто називати його «чистим» JS, хоча цей термін часто розглядається як неправильний термін — фреймворки в JS не є домішкою.

Але які переваги насправді має робота без фреймворку JS? Можна навести багато аргументів як за, так і проти використання JS без фреймворків. Наприклад, можна стверджувати, що необроблений JS дає вам кращий контроль і допомагає контролювати напрямок вашого проекту та світитися саме так, як ви хочете.

Для просунутих і досвідчених програмістів це дійсно може бути правдою. Іноді контроль над додатком дійсно бажаний, особливо якщо фреймворк, з яким ви працюєте, дуже самовпевнений (наприклад, Angular). Використання необробленого JS означає, що ви не зв'язані стандартами даного фреймворку та маєте більше можливостей для експериментів.

З іншого боку, використання без-фреймворків JS безперечно має крутішу та порівняно складнішу криву навчання. Ви не маєте доступу до літератури чи документації спільноти фреймворку. Крім того, абсолютним початківцям часто простіше дотримуватись підходу до розробки, оскільки вони мають уже існуючу основу для роботи. У простому JS такої розкоші не існує.

Але наведені пункти є досить другорядними аргументами. На користь фреймворків JS можна сказати, що їх найкраще використовувати, коли ви працюєте з інтерфейсом користувача (UI). Використання необробленого JS для розробки інтерфейсу користувача часто є складним, а іноді навіть заплутаним — у міру того, як ми проходимо через розділи цієї книги, і ви знайомитеся з фреймворками JS, спеціально призначеними для таких завдань, ви помітите, наскільки складним може бути життя без таких фреймворків.

І найбільший аргумент проти фреймворків JS? Якщо ви маєте практичні знання про звичайний JS, ви можете вважати себе стійкими до фреймворків, навіть якщо покладаєтеся на певний набір фреймворків. Візьміть до уваги: сьогодні багато розробників jQuery вивчають і намагаються наздогнати React або Angular. Чому це так? Тому що останні варіанти набули популярності.

Однак для чистих розробників JS, які не завжди покладаються на одну структуру. Таке наздоганяння здійснити легше. Ви можете створити програму на VueJS або Meteor, не опанувавши ці фреймворки, лише якщо у вас є хороші знання про JS.

Таким чином, навіть якщо ви працюєте з фреймворками JS, часто доцільно підтримувати свої навички JS у хорошій формі та витратити деякий час на вивчення простого JS. Це фактично гарантує, що ви не обмежені певним стандартом у певній структурі, а матимете більше можливостей для гнучкості та зростання.

. . . .

Розділ 2

Початок роботи з Node.js

ЩО ТАКЕ NODE.JS?

Node.js — середовище виконання коду JavaScript поза браузером. Ця платформа дозволяє писати серверний код для динамічних веб-сторінок і веб-додатків, а також для програм командного рядка. Використання Node.js реалізує парадигму «JavaScript для всього». Це передбачає використання однієї мови програмування для розробки веб-додатків замість використання різних мов для роботи на інтерфейсі та сервері. Node.js — це серверна платформа для роботи з JavaScript через двигун V8.

За допомогою Node легше масштабувати. Коли до сервера одночасно підключені тисячі користувачів, Node працює асинхронно, тобто розставляє пріоритети та більш грамотно розподіляє ресурси. JavaScript, наприклад, виділяє окремий потік для кожного з'єднання.

Node.js добре підходить для розробки веб-додатків RTA, які реагують на дії користувача в реальному часі. Наприклад, це може бути такий онлайн-редактор, як Google Docs, який дозволяє кільком користувачам працювати над одним документом одночасно.

Node.js легко обробляє велику кількість запитів одночасно і забезпечує продуктивність програми. Тому JavaScript на стороні сервера часто використовується для створення SPA-односторінкових веб-додатків, у яких рендеринг виконується на стороні клієнта. Node.js на сервері використовується Netflix, Uber, eBay, Groupon, Yahoo та іншими відомими організаціями та проектами.

Node.js не є універсальною платформою, яка буде домінувати у світі веб-розробки. Навпаки, це платформа для вирішення строго визначених завдань. Це конче необхідно зрозуміти. Звичайно, не варто використовувати Node.js для операцій, які інтенсивно навантажують процесор, більш того, використання Node.js у важких обчисленнях фактично зводить нанівець усі його переваги. Node.js дійсно хороший для створення швидких, масштабованих мережевих програм, оскільки він дозволяє одночасно обробляти величезну кількість з’єднань із високою пропускною здатністю, що еквівалентно високій масштабованості.

Тонкощі Node.js «під капотом» досить цікаві. Порівняно з традиційними версіями веб-сервісів, де кожне з'єднання (запит) генерує новий потік, завантажуючи оперативну пам'ять системи та, зрештою, безслідно аналізуючи цю пам'ять, Node.js набагато економічніший: він працює в одному потоці. , використовує неблокуючий ввід-вивід для викликів і підтримує десятки тисяч конкурентоспроможних з'єднань.

ЩО ТАКЕ JAVASCRIPT?

JavaScript — це багатопарадигмальна мова програмування, яка зазвичай використовується як вбудований інструмент для програмного доступу до різних об'єктів програми. З точки зору веб-розробки, без знання цієї технології неможливо створювати сучасні інтерактивні сайти. Мова JS – це те, що «оживляє» розмітку сторінки (HTML) і функціональність користувача (CMS) сайтів. Ця мова дозволяє сторінці або її окремим елементам реагувати на дії користувача. Сьогодні JavaScript є основною мовою програмування для браузерів. Вона повністю сумісна з операційними системами Windows, Linux, Mac OS, а також усіма популярними мобільними платформами.

КОРОТКА ІСТОРІЯ

Платформа Node.js була представлена в 2009 році. Її створив інженер Райан Дал, а розробку спонсорувала компанія Joyent. Компанія відома підтримкою проектів з відкритим кодом, зокрема Node.js, Illumos, SmartOS.

Раян Дал використовував для створення двигуна Node.js V8. Платформа реалізована за допомогою низькорівневої неблокуючої моделі введення-виведення, яка побудована на подієво-орієнтованій моделі.

Наприкінці 2014 року інженер Fedor Indutny, який входив до основної команди розробників платформи, створив популярний форк Node.js — io.js. Форк з'явився через невдоволення розробників політикою компанії Joyent.

Платформа io.js перевершує Node.js у продуктивності. Але творці форку вирішили возз'єднатися з Node у 2015.js, щоб вплинути на розвиток основної платформи. Зараз розробкою формально керує Node.js Foundation.

ЧОМУ НАМ СЛІД ВИВЧИТИ NODE.JS?

Ринок став свідком придбання кількох Node pioneers.js, зокрема StrongLoop від IBM, FeedHenry від Red Hat і Modulus від Progress Software.

Фахівці IT-індустрії відзначають, що мова сценаріїв на основі JavaScript є обов'язковим інструментом для розробки та доставки додатків у сучасних компаніях.

Node.js продовжує розвиватися з майже безпрецедентною швидкістю. За останні 5 років розробники додали понад 190 000 модулів до Node.js (та інших бібліотек JavaScript). Це перевищує весь репозиторій Perl CPAN, який збирався протягом 20 років, і обходить Java Maven Central, незважаючи на меншу кількість розробників Node.js.

Що стало причиною такої популярності серед корпоративних розробників і чи можуть IT-директори бути впевнені, що Node.js буде активно використовуватися щонайменше 10 років?

Екосистема підтримки модуля Node.js навколо ядра зазнала значного зростання. Спільнота Node отримала значну користь від існування Node Package Manager (NPM), який забезпечує центральне сховище спільних модулів

Це ключова частина гнучкого та легкого способу роботи з Node.js. Це дозволяє кожній програмі мати необхідні модулі у власному дереві залежностей. Таким чином, кожна програма може мати власний набір модулів, що дозволяє уникнути конфліктів залежностей з іншими програмами.

Це гнучкий інструмент на основі Node.js, що використовує сервіс npmjs.org, що призвело до значного збільшення кількості загальних модулів і до того, що npmjs.org став репозиторієм не тільки для сервера Node.js, але також для модулів JavaScript на стороні клієнта.

На ранній стадії розробки Node.js використовувався такими компаніями, як Netflix, Walmart, PayPal, Dow Jones і Groupon. Вони створили внутрішні команди, які використовували Node, дотримуючись підходу «розділяй і володарюй»: руйнування того, що раніше було більш «монолітним» підходом до створення веб-сервісів. Це дозволило їм швидко розробляти та оновлювати рішення для різних сфер бізнесу та негайно розгортати мікросервіси у виробництві.

NODE.JS СТАЄ МАСОВИМ

Node.js особливо добре підходить для компаній, які мають веб-інфраструктуру та мобільні додатки, у бекенді яких необхідно швидко впроваджувати інновації за допомогою архітектури, побудованої на мікросервісах. Сюди входять організації, які можуть здатися консервативними, але стикаються з необхідністю розробки програм, які відповідають швидким змінам у світі бізнесу, зберігаючи при цьому стабільність і безпеку існуючих систем.

Ці подвійні вимоги змусили IT-команди змінити спосіб мислення. Майже всім компаніям доводилося шукати способи швидко реагувати на запити щодо програмного забезпечення для бізнесу, зберігаючи при цьому поточний застарілий код, який змінюється значно рідше. Node.js є одним із інструментів для команд, які розробляють системи взаємодії, наприклад мобільні рішення.

Node.js часто хвалять за те, що він відповідає на корпоративні запити та дозволяє створювати програми за допомогою API, які можуть легко та ефективно отримувати доступ до серверної частини та великих обсягів даних. Дійсно, зосередження на повторно використовуваному API RESTful як більш гнучкому способі створення архітектури великомасштабних програмних систем дозволило Node.js знайти своє місце.

Node.js здатний значно скоротити час розробки програми, зберігаючи ту саму функціональність. Джон К. Оустерхаут, який допоміг розробити значну мову сценаріїв і набір інструментів Tcl/Tk, ще в 1990-х роках стверджував, що мови програмування сценаріїв за своєю суттю продуктивніші, ніж більш важкі мови програмування, такі як C або C++.

Порівняно з більш важкими стеками розробка додатків за допомогою Node.js відбувається швидше, а з розвитком екосистеми Node вона тільки прискорюється. Це той випадок, коли варто витратити час на пошук того, що вже є в спільноті Node, і з’ясувати, чи можливо повторно використовувати деякі загальні модулі.

У середовищі розробки Node дуже прихильно ставляться і поважають фахівці з цієї технології. Єдність розробників та їхній моральний дух підвищується, якщо в IT-структурі організації є команда, яка працює з Node.js. Це сприймається як цікава і дійсно крута можливість.

Node.js чудово підходить для додатків, побудованих на архітектурі мікросервісів, завдяки низькому споживанню CPU, потужності обробки та ефективному використанню оперативної пам’яті. Ця перевага особливо очевидна для завдань, які передбачають операції вводу-виводу, а не використання CPU, оскільки реалізація моделі виконання Node.js дозволяє використовувати «легкий» паралелізм на основі одного потоку моделі виконання, який не вимагає складні методи паралельного програмування.

Поєднуючи все це — мислення програмістів і різноманітні технічні переваги легкого підходу — компанії отримують новий підхід до вирішення IT-проблем із командою розробників, мотивованою для їх вирішення та озброєною високошвидкісним набором інструментів.

ДЛЯ ЧОГО БУВ СТВОРЕНИЙ NODE.JS?

Node.js — це не окрема мова програмування, а платформа для використання JavaScript на стороні сервера. Якщо говорити про мову, то і фронтенд, і бекенд використовують один і той же JavaScript. Єдина відмінність полягає в наборі API, які використовують фронтендери та бекендери.

JavaScript на основі браузера використовує веб-інтерфейси API, які надають доступ до DOM та інтерфейсу користувача сторінок і веб-додатків. JavaScript на стороні сервера використовує API, які надають доступ до файлової системи програми, http-запитів і потоків.

Тобто Node.js — це технологія використання JS на сервері. Про особливості та перспективи розвитку мови JavaScript можна дізнатися у відповідній статті, а тут мова йде про одну з технологій цієї мови.

Однією з найскладніших проблем при написанні систем, які спілкуються через мережу, є обробка введення та виведення. Читання та запис даних у та з мережі, на диск та інші пристрої. Переміщення даних вимагає часу, і правильне планування цих дій може значно вплинути на час відповіді системи на запити користувача або мережі.

У традиційному методі обробки введення та виведення передбачається, що функція, наприклад ReadFile, починає читати файл і припиняє працювати лише тоді, коли файл повністю прочитано. Це називається синхронним вводом/виводом (введення/виведення).

Node був задуманий для полегшення та спрощення використання асинхронного вводу-виводу.

Код JavaScript легко вписується в систему типу Node. Це одна з небагатьох мов, яка не має вбудованої системи вводу-виводу. Тому JavaScript легко вбудовується в досить ексцентричний підхід до введення-виведення у Node і, як наслідок, не створює двох різних систем введення та виведення. У 2009 році під час розробки Node люди вже використовували введення-виведення на основі зворотного виклику в браузері, тому спільнота навколо цієї мови звикла до асинхронного стилю програмування.

ОСНОВНІ ХАРАКТЕРИСТИКИ NODE.JS

Давайте розглянемо основні функції Node.js.

Швидкість

Однією з головних привабливих особливостей Node.js є швидкість. Код JavaScript, який виконується в середовищі Node.js, може бути вдвічі швидшим, ніж код, написаний скомпільованими мовами, такими як C або Java, і на порядки швидшим, ніж інтерпретовані мови, такі як Python або Ruby. Причиною цього є неблокуюча архітектура платформи, і конкретні результати залежать від використовуваних тестів продуктивності, але, загалом, Node.js є дуже швидкою платформою.

Простота

Платформу Node.js легко освоїти та використовувати. Насправді це досить просто, особливо в порівнянні з деякими іншими серверними платформами.

JavaScript

У середовищі Node.js виконує код, написаний на JavaScript. Це означає, що мільйони інтерфейсних розробників, які вже використовують JavaScript у браузері, можуть писати як серверний, так і клієнтський код на одній мові програмування без необхідності вивчати абсолютно новий інструмент для переходу до серверної розробки. Браузер і сервер використовують однакові концепції мови. Крім того, у Node.js можна швидко перейти до використання нових стандартів ECMAScript, оскільки вони реалізовані на платформі. Для

цього вам не потрібно чекати, поки користувачі оновлять свої браузери, оскільки Node.js — це серверне середовище, яке повністю контролюється розробником. Як наслідок, нові мовні функції доступні, коли ви встановлюєте версію Node.js, яка їх підтримує.

Двигун V8

В основі Node.js, серед інших рішень, лежить механізм JavaScript V8 з відкритим кодом від Google, який використовується в Google Chrome та інших браузерах. Це означає, що Node.js використовує роботу тисяч інженерів, які зробили середовище виконання Chrome JavaScript неймовірно швидким і продовжують працювати над вдосконаленням V8.

Асинхронність

У традиційних мовах програмування (C, Java, Python, PHP) усі інструкції за замовчуванням блокуються, якщо розробник явно не подбає про асинхронне виконання коду. У результаті, якщо в такому середовищі зробити мережевий запит на завантаження певного коду JSON, виконання потоку, з якого зроблено запит, буде призупинено, доки відповідь не буде отримано та оброблено. JavaScript значно полегшує написання асинхронного та неблокуючого коду за допомогою єдиного потоку, функцій зворотного виклику та підходу розробки, керованого подіями. Кожного разу, коли нам потрібно виконати важку операцію, ми передаємо зворотний виклик відповідному механізму, який буде викликаний відразу після завершення цієї операції. В результаті вам не потрібно чекати результатів таких операцій, щоб продовжити роботу. Подібний механізм з’явився в браузерах. Ми не можемо дозволити собі чекати, скажімо, завершення запиту AJAX, не маючи можливості відповісти на дії користувача, наприклад натискання кнопок. Щоб користувачам було зручно працювати з веб-сторінками, все, як завантаження даних з мережі, так і обробка натискань кнопок, має відбуватися одночасно, в режимі реального часу. Якщо ви коли-небудь створювали обробник події натискання кнопки, ви вже використовували методи асинхронного програмування.

Асинхронні механізми дозволяють одному серверу Node.js обробляти тисячі підключень одночасно, не навантажуючи програміста завданнями з управління потоками та організації паралельного виконання коду. Такі речі часто є джерелами помилок. Node.js надає розробнику неблокуючі базові механізми введення-виведення та, загалом, бібліотеки, що використовуються

в середовищі Node.js, написані з використанням неблокуючих парадигм. Це робить поведінку блокуючого коду винятком, а не нормою.

Коли Node.js потрібно виконати операцію введення-виведення, таку як завантаження даних із мережі, доступ до бази даних або файлової системи, замість того, щоб блокувати головний потік, Node.js, від очікування результатів такого операція ініціює своє виконання та продовжує виконувати інші дії до отримання результатів цієї операції.

Бібліотеки

Завдяки простоті та зручності роботи з менеджером пакетів для Node.js, який називається npm, екосистема Node.js справді процвітає. Тепер реєстр npm містить понад півмільйона пакетів з відкритим кодом, які будь-який розробник Node.js може вільно використовувати. Розглянувши деякі з основних функцій платформи Node.js, давайте спробуємо її в дії. Почнемо з установки.

. . . .