Engineer

Rust

компільована мова, що сама стежить за пам'яттю

Що це таке

Rust — це мова програмування для роботи, яку традиційно виконують на C і C++, де програма сама керує своєю пам’яттю і часто компонується з програмами, написаними іншими мовами. Компілятор заздалегідь перетворює вихідний код на машинний. Версію 1.0 оголосили 15 травня 2015 року як мову, що поєднує низькорівневий контроль над продуктивністю з гарантіями безпечної роботи з пам’яттю й досягає цього без збирача сміття і без середовища виконання, тож бібліотека на Rust може стати на місце бібліотеки на C.

Пам’яттю керує володіння — набір правил, які перевіряє компілятор. Кожне значення має одного власника водночас, і коли власник виходить з області видимості, значення звільняється разом зі своєю пам’яттю в момент, визначений під час компіляції, а не тоді, коли збирач сміття наступного разу зазирне. Код, якому потрібне значення без передачі володіння, позичає посилання, а правила позичання дозволяють у кожну мить або одне змінне посилання, або будь-яку кількість незмінних і вимагають, щоб кожне посилання було дійсним. Програма, що порушує правило, не компілюється. Ці самі правила виключають перегони даних, тож два потоки, що без синхронізації пишуть у ті самі дані, — це помилка компіляції, а не збій, який з’являється лише зрідка.

У Rust немає винятків. Функція, що може завершитися помилкою передбачуваним чином, повертає `Result`, який містить або значення, або помилку, і компілятор типово попереджає, якщо код, який її викликає, не використовує результат; помилка, після якої програма не може відновитися, як-от читання за межами масиву, натомість іде через макрос `panic!`.

Статичний аналіз консервативний, тож компілятор відкидає деякі програми, які були б коректними, а деяку роботу, як-от керування обладнанням чи виклик іншої мови, перевірити неможливо взагалі. Для таких випадків є `unsafe`, що відкриває п’ять додаткових можливостей, серед них розіменування сирого вказівника і виклик небезпечної функції, а такою типово є будь-яка функція, оголошена з C. Він не вимикає перевірку позичань; він позначає рядки, де за коректну роботу з пам’яттю ручається програміст, а не компілятор, тож безпечність цих операцій залежить від того, чи дотримується програміст їхніх контрактів, зокрема й у навколишньому безпечному коді. Порада книги Rust — тримати ці блоки малими й загортати їх у безпечний інтерфейс, і саме так побудовано частини стандартної бібліотеки.

Новий компілятор виходить кожні шість тижнів. Зміни, що зламали б наявний код, чекають на редакцію (edition), яка з’являється приблизно раз на три роки. Їх чотири, 2015, 2018, 2021 і 2024, і найновіша стала стабільною 20 лютого 2025 року разом із Rust 1.85.0. Кожен пакет указує свою редакцію в `Cargo.toml`, кожен компілятор збирає всі редакції, що вийшли до нього, а пакети різних редакцій компонуються разом, тож проєкт переходить на нову тоді, коли сам вирішить. Cargo, менеджер залежностей і засіб збирання, входить до складу мови, rustfmt тримає стиль коду однаковим, а сторонні бібліотеки надходять як пакети, що називаються крейтами.

Застосуйте поняття до рішення

Застосуйте «Що таке Rust?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

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. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

У C і C++ вказівник, використаний після звільнення його пам’яті, або два потоки, що пишуть ті самі дані, компілюються без жодних заперечень і дають збій пізніше, зрідка й часто далеко від причини. Rust перетворює такі програми на помилки компіляції, а оскільки перевірки відбуваються під час збирання, готова програма нічого за них не платить.

Без збирача сміття і без середовища виконання, яке треба тягти за собою, код на Rust може лежати під програмами іншими мовами. Усе, що вміє викликати C, може викликати ядро на Rust через C-інтерфейс, тож одна добре протестована реалізація може обслуговувати клієнтів на кількох платформах. Книга Rust серед застосувань мови в продакшні називає інструменти командного рядка, вебсервіси, вбудовані пристрої та значні частини браузера Firefox.

Ціну платять наперед. Код треба будувати так, щоб компілятор бачив, що він безпечний, а це часто означає вирішити, хто володіє значенням, замість вільно передавати посилання, і книга сама визнає, що до володіння треба звикнути. Коли компілятор відкидає програму, якої не може перевірити, відповіддю зазвичай стає інша структура, а не обхід перевірки.

Стабільність — друга половина угоди. Від версії 1.0 зміни, що ламають наявний код, здебільшого виключено, тож така зміна надходить лише через редакцію, яку пакет обирає сам.

Як це виглядає на практиці

У портфоліо Rust — це ядро під нативним клієнтом обміну повідомленнями для macOS: добре протестоване ядро протоколу написано на Rust, воно надає C-інтерфейс, а клієнт, 11 141 рядок Swift у 53 файлах, компонує його як статичний архів, зібраний із зафіксованої ревізії основного репозиторію. Фіксація важлива, бо C-інтерфейс — це контракт: ядро, яке змінюється, змінює контракт, не змінюючи жодного рядка Swift. Для всього, що повертає C-інтерфейс, діє своє правило володіння, і помилка в одному з них дає витік або аварійне завершення без жодного попередження компілятора між ними, тож тонка обгортка володіє кожним вказівником і кожним рядком, що перетинає межу, і жоден шар над нею не бачить сирого вказівника.

Компонування цього ядра із застосунком означає постачання всього, від чого воно залежить, і саме цьому присвячено перевірку ліцензій 644 крейтів Rust. Кожен крейт декларує свою ліцензію як вираз, і під час кожної збірки перевірка звіряє кожен вираз у повному графі залежностей зі списком дозволених умов, читаючи OR як вибір між ліцензіями, а AND — як обов’язок дотримуватися обох. Усе, що не збігається, зупиняє збірку з помилкою. Оскільки перевірка, яка ніколи не спрацьовувала, має такий самий вигляд, як і та, що спрацювати не може, її двічі навмисно змусили спрацювати: раз —

переписавши ліцензію одного крейта на умову, яку список відкидає, і раз — змінивши зафіксовану ревізію ядра без перегенерації. Вона читає лише задекларовані метадані й не є юридичною консультацією.

Власного коду на Rust цей сайт не має; з мовою він стикається опосередковано, через Ruff, що написаний на Rust і перевіряє кожен скрипт Python, якого торкається коміт.

Де виникають проблеми

Здебільшого клопіт із Rust виникає на краях того, що перевіряє компілятор: в арифметиці оптимізованої збірки, на межі з іншою мовою і в коді, що надходить разом із залежністю.

У релізній збірці арифметика переповнюється по колу. Лічильник, що під час тестування панікував через переповнення, у продакшні тихо стає малим числом. У налагоджувальних збірках Rust перевіряє цілочисельну арифметику на переповнення й панікує; збірка з `--release` цю перевірку пропускає, і значення переходить по колу, тож 256 у `u8` стає 0, а книга вважає помилкою покладатися на таку поведінку. Де значення може переповнитися, оберіть поведінку явно: `checked_*` повертає `None`, `saturating_*` зупиняється на межі, а `wrapping_*` переходить по колу навмисно.

```
fn add_stock(on_hand: u8, delivered: u8) -> Option<u8> {
    // None when the sum does not fit, in debug and release alike
    on_hand.checked_add(delivered)
}
```

Помилки пам’яті повертаються на межі з C. Програма, чия частина на Rust збирається без жодного попередження, має витоки пам’яті або аварійно завершується всередині зовнішнього виклику. Функції, оголошені в блоці `extern`, небезпечно викликати, бо інша мова не забезпечує жодного з правил Rust, тож те, хто звільняє кожен вказівник, — це домовленість, якої компілятор не бачить. Призначте одну обгортку відповідальною за кожен вказівник і рядок, що перетинає межу, запишіть кожне правило володіння поруч із викликом і відкривайте шарам вище лише безпечний інтерфейс.

Паніка доходить до C, і процес аварійно завершується. Функція на Rust, викликана з C, панікує, і замість значення помилки зупиняється вся програма. Починаючи з Rust 1.81, що вийшов 5 вересня 2024 року, паніка, яка намагається вийти з функції, оголошеної зі звичайним інтерфейсом `"C"`, перериває процес. Перетворюйте збої на значення, що повертаються, ще до межі, а функцію з `"C-unwind"` оголошуйте лише там, де код на іншому боці очікує розмотування стека.

Рутинне оновлення змінює ліцензію всієї збірки. Підвищення версії на три рівні нижче приносить крейт, ліцензію якого продукт прийняти не може, і ніщо в збірці про це не повідомляє. Кожен крейт декларує власну ліцензію, і зобов'язання накопичуються по всьому графу. Звіряйте вираз кожного крейта зі списком дозволених умов під час кожної збірки, зупиняйте збірку на всьому, що не збігається, і один раз навмисно зламайте перевірку, щоб довести, що вона спрацьовує.

Що почитати далі

- The Rust Programming Language — офіційна книга, від встановлення через володіння й обробку помилок до небезпечного коду.
- What Is Ownership? — три правила володіння і те, як пам'ять звільняється без збирача сміття.
- Unsafe Rust — п'ять небезпечних можливостей, сирі вказівники й виклик C через `extern`.
- The Manifest Format — поля `edition`, `rust-version` і `license` у `Cargo.toml`.
- Announcing Rust 1.0 — початкові цілі й обіцянка стабільності, що почалася з 1.0.

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 Rust?» 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/rust/