Engineer

Go

компільована строго типізована мова зі збиранням сміття

Що це таке

Go — це мова програмування загального призначення, що виникла в Google, задумана з огляду на системне програмування і з конкурентністю, вбудованою в саму мову, а не доданою як бібліотека. Її часто називають golang, за першою вебадресою проєкту, golang.org, але FAQ проєкту наполягає, що мова зветься просто Go, пишеться Go і ніколи GO.

Robert Griesemer, Rob Pike і Ken Thompson почали накидати її цілі на дошці 21 вересня 2007 року, а 10 листопада 2009 року проєкт став публічним і відкритим. Версія Go 1 вийшла 28 березня 2012 року з обіцянкою, що програми, написані для неї, і далі компілюватимуться й працюватимуть без змін під кожним наступним випуском Go 1. Ця обіцянка досі діє: специфікація нині описує версію мови go1.27, а випуск Go 1.27.0 вийшов 19 серпня 2026 року. Узагальнені типи, або generics, з'явилися в Go 1.18, і розробники мови назвали їх найбільшою зміною від першого відкритого випуску.

За словами FAQ, мова виросла з вибору, який її авторам набридло робити: між ефективною компіляцією, ефективним виконанням і простотою програмування. Go прагне дати всі три. Програма складається з пакетів, а весь інструментарій — одна команда, `go`, яка збирає, тестує, форматує й завантажує залежності. Ключове слово `go` перед викликом функції запускає горутину — функцію, що працює поряд із рештою програми, — і горутини такі дешеві, що, за словами FAQ, сотні тисяч їх можуть ділити один адресний простір. Канали передають значення між ними, а Effective Go підсумовує бажаний стиль гаслом, яке українською звучить так: «Не комунікуйте, ділячись пам'яттю; натомість діліться пам'яттю, комунікуючи».

Дві звички вирізняють код на Go, хоч де б він трапився. Програма `gofmt` виводить кожен вихідний файл в одному стандартному форматуванні, тож про форматування на рев'ю не сперечаються. А компілятор відмовляється збирати програму з невикористаною змінною чи невикористаним імпортом, бо перша може приховувати помилку, а другий сповільнює кожну наступну збірку.

Застосуйте поняття до рішення

Застосуйте «Що таке Go?» 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

Рішення, на яке відповідає Go, зазвичай таке: якою мовою писати мережевий сервіс, утиліту командного рядка чи частину інфраструктури. Альтернативи лежать по обидва боки від нього: скриптова мова з одного, C, C++ чи Rust з другого.

Порівняно зі скриптовою мовою Go дає просте розгортання й ранні помилки. Компонувальник за замовчуванням створює статично скомпоновані бінарні файли, тож результатом збірки є один файл, що несе власне середовище виконання й запускається на сервері, де більше нічого не встановлено. FAQ оцінює ціну в кілька мегабайтів для програми, що друкує «hello, world», проти приблизно 750 кБ для такої самої програми на C. Помилки типів виявляються під час компіляції, а не тоді, коли в продакшні спрацює рідко вживана гілка, і одна з початкових цілей полягала в тому, щоб навіть великий виконуваний файл збирався за кілька секунд на одному комп'ютері.

Порівняно з C, C++ чи Rust Go дає простоту й безпечну роботу з пам'яттю ціною контролю. Збирач сміття звільняє пам'ять, тож звичайний код позбавлений цілого класу помилок use-after-free, але програма не вирішує, коли саме пам'ять звільняється, і кожен бінарний файл несе в собі середовище виконання, яке це робить. Для більшості серверів

це правильний обмін. Для ядра операційної системи, маленького вбудованого пристрою чи гарячого циклу, який не терпить збирача сміття, — неправильний.

Тихіший аргумент — час. У 2023 році Russ Cox написав, що Go 2, яка ламає старі програми, не з'явиться ніколи і що проєкт натомість ще рішучіше робитиме ставку на сумісність. Обіцянку оплачують оновленнями, а не переписуванням. Новий основний випуск виходить кожні шість місяців, у лютому й серпні, і кожен підтримується, доки не з'являться два новіші, тож команда, якій потрібні виправлення безпеки, переходить на новішу версію приблизно раз на рік.

Мала мова дещо пише й розлого. Помилки — звичайні значення, які повертають і перевіряють там, де вони виникають, і це однаковою мірою явно й одноманітно.

Як це виглядає на практиці

Горутину дешево запустити й так само дешево втратити. Сторінка про перенесення повільної роботи в чергу завдань River каже про це прямо: якщо винести повільну роботу в горутину, вона зникне тієї ж миті, коли процес перезапуститься, посеред розгортання, і не лишиться жодного сліду, що вона взагалі мала відбутися. Компанія перенесла цю роботу в River, який тримає свою чергу в PostgreSQL, тож завдання й рядки, які воно змінює, комітяться або відкочуються разом. За ним стоять 15 модулів-воркерів і 8 запланованих завдань, а рівнем нижче — 20 завдань pg_cron. Горутина дає конкурентність; надійне збереження завдання має забезпечувати інший механізм.

Тестування входить до інструментарію, і багаторівневий набір тестів показує, що це дає на практиці. В API на Go — 981 тестова функція у 310 файлах, і це лише один із чотирьох рівнів, кожен з яких тестують там, де живе його поведінка, поряд зі специфікаціями фронтенду й власними тестами SQL у базі даних.

Безперервна програма безпеки так само спирається на інструменти мови. Залежності Go перевіряє govulncheck, і вразливість, яку розпізнають налаштовані перевірки, може спричинити невдалу збірку. Сторінка називає

справжньою ціною сканування обсяг знахідок, і govulncheck зроблено саме проти цієї ціни: він повідомляє про вразливість лише тоді, коли власні функції коду викликають уражену функцію, прямо чи через інші.

Де виникають проблеми

Функція повідомляє про помилку там, де помилки немає. Значення інтерфейсу в Go містить тип і значення, і воно дорівнює nil, лише коли обидва не задані. Функція, яка повертає nil-вказівник конкретного типу помилки через інтерфейс `error`, віддає викликачеві інтерфейс, що містить тип, а такий інтерфейс не дорівнює nil:

```
func save() error {
    var err *ValidationError // nil-вказівник
    if invalid() {
        err = &ValidationError{}
    }
    return err // ніколи не дорівнює nil: інтерфейс містить тип
}
```

Повертайте буквальний `nil` у разі успіху й оголошуйте кожен результат-помилку як `error`, а не як конкретний тип.

Сервіс, що пройшов усі тести, падає під реальним навантаженням. Операції з map навмисно не синхронізовано, бо блокування кожного доступу сповільнило б більшість програм, і FAQ попереджає, що неконтрольований доступ до map може обвалити програму. Причина в тому, що одна горутина пише в map, поки інша з нього читає чи теж пише; самі лише читання безпечні. Захистіть map м'ютексом, віддайте його одній горутині, до якої інші звертаються через канал, або використайте `sync.Map` у тих випадках, для яких його створено. Потім запустіть тести з `go test -race`. Для типової програми детектор гонок може збільшити використання пам'яті в п'ять–десять разів, а час виконання — у два–двадцять, і він знаходить гонки лише на тих шляхах коду, які справді виконуються, тож бінарний файл, зібраний із `-race` і запущений під реалістичним навантаженням, знайде більше.

Горутини, запущені в циклі, усі працюють з останнім елементом. До Go 1.22 цикл `for` мав одну змінну на всі ітерації, тож замикання, яке її захопило, бачило те значення, що вона мала, коли горутина нарешті

запускалася. Go 1.22 дає кожній ітерації власну змінну, але лише в модулях, чий go.mod оголошує `go 1.22` або новішу версію, а старий код зберігає попередню семантику, навіть коли його збирає новіший інструментарій. Підвищте версію в рядку `go` і запустіть `go vet` та тести. Тест, що почав падати після зміни, часто весь час перевіряв не те значення.

Збірка на новій машині зупиняється, щоб завантажити іншу версію Go. Починаючи з Go 1.21 рядок `go` є вимогою, і коли встановлений інструментарій старіший, ніж просить модуль, команда `go` завантажує новіший як модуль і запускає його з кешу модулів. На ноутбуці це зручно; у збірці, ізольованій від мережі, чи в зафіксованому образі контейнера це несподіванка. Установіть інструментарій, який називає go.mod, і задайте змінну середовища `GOTOOLCHAIN`, коли збірка мусить використовувати одну точну версію.

Що почитати далі

- The Go Programming Language Specification — довідник з кожної конструкції мови, що оновлюється з кожним випуском.
- Frequently Asked Questions (FAQ) — навіщо створили мову, рішення, що стоять за її дизайном, і загадки з nil-помилкою та map, описані вище.
- Go 1 and the Future of Go Programs — обіцянка сумісності та винятки, які вона допускає.
- Release History — кожен випуск від Go 1, з політикою підтримки на початку сторінки.
- Data Race Detector — як його запускати, що він ловить і чого коштує.
- Fixing For Loops in Go 1.22 — зміна змінних циклу, чому її прив’язано до go.mod і як перевірити з нею старий код.

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 Go?» 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/go/