Engineer

Go

kompilierte, streng typisierte Sprache mit Garbage Collector

Was es ist

Go ist eine universell einsetzbare Programmiersprache, die bei Google entstand. Sie wurde mit Blick auf die Systemprogrammierung entwickelt; Nebenläufigkeit ist Teil der Sprache selbst und nicht erst durch eine Bibliothek ergänzt. Nach der ersten Webadresse des Projekts, golang.org, wird sie oft golang genannt. Die FAQ stellen jedoch klar: Die Sprache heißt einfach Go, geschrieben Go und niemals GO.

Robert Griesemer, Rob Pike und Ken Thompson skizzierten die Ziele von Go erstmals am 21. September 2007 auf einem Whiteboard. Am 10. November 2009 wurde die Sprache als Open-Source-Projekt veröffentlicht. Go 1 folgte am 28. März 2012 mit dem Versprechen, dass dafür geschriebene Programme auch mit jeder späteren Go-1-Version unverändert kompilieren und laufen würden. Dieses Versprechen gilt weiterhin: Die Spezifikation beschreibt inzwischen die Sprachversion go1.27, und Go 1.27.0 erschien am 19. August 2026. Generics kamen mit Go 1.18 hinzu. Die Sprachentwickler bezeichneten sie als die größte Änderung seit der ersten Open-Source-Veröffentlichung.

Laut FAQ entstand die Sprache aus einer Entscheidung, die ihre Autoren nicht länger treffen wollten: zwischen effizienter Kompilierung, effizienter Ausführung und einfacher Programmierung. Go soll alle drei Ziele erfüllen. Ein Programm besteht aus Paketen. Die Werkzeugkette wird über einen Befehl bedient, `go`, der baut, testet, formatiert und Abhängigkeiten abruft. Das Schlüsselwort `go` vor einem Funktionsaufruf startet eine Goroutine: eine Funktion, die neben dem übrigen Programm läuft. Goroutinen sind so günstig, dass sich laut FAQ Hunderttausende einen Adressraum teilen können. Channels transportieren Werte zwischen ihnen. Effective Go fasst den vorgesehenen Stil in einem Leitsatz zusammen: „Kommunizieren Sie nicht durch gemeinsam genutzten Speicher; teilen Sie Speicher stattdessen durch Kommunikation."

Zwei Gewohnheiten prägen Go-Code überall. Das Programm `gofmt` gibt jede Quelldatei in einem einheitlichen Layout aus, sodass die Formatierung im Review nicht zur Debatte steht. Außerdem lehnt der Compiler ein Programm mit einer unbenutzten Variablen oder einem unbenutzten Import ab: Erstere kann einen Fehler verdecken, Letzterer verlangsamt jeden späteren Build.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Go?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Go beantwortet meist die Frage, in welcher Sprache ein Netzwerkdienst, ein Kommandozeilenwerkzeug oder ein Teil der Infrastruktur geschrieben werden soll. Die Alternativen liegen auf beiden Seiten: auf der einen eine Skriptsprache, auf der anderen C, C++ oder Rust.

Gegenüber einer Skriptsprache bietet Go eine einfache Bereitstellung und früh erkannte Fehler. Der Linker erzeugt standardmäßig statisch gelinkte Binärdateien. Das Ergebnis eines Builds ist damit eine einzelne Datei, die ihre Laufzeitumgebung mitbringt und auf einem Server ohne weitere installierte Software läuft. Laut FAQ kostet das bei einem Programm, das „hello, world“ ausgibt, ein paar Megabyte, gegenüber etwa 750 kB für dasselbe Programm in C. Typfehler werden beim Kompilieren sichtbar, statt erst dann, wenn ein selten genutzter Zweig in Produktion ausgeführt wird. Eines der ursprünglichen Ziele war zudem, selbst eine große ausführbare Datei auf einem einzigen Computer in wenigen Sekunden zu bauen.

Gegenüber C, C++ oder Rust bietet Go Einfachheit und Speichersicherheit, verzichtet dafür aber auf Kontrolle. Ein Garbage Collector gibt Speicher wieder frei. Gewöhnlicher Code bleibt dadurch von einer ganzen Klasse von Zugriffen auf bereits freigegebenen Speicher verschont. Das Programm bestimmt jedoch nicht genau, wann Speicher freigegeben wird, und jede Binärdatei bringt die

dafür zuständige Laufzeitumgebung mit. Für die meisten Server ist das der richtige Kompromiss. Für einen Kernel, ein kleines eingebettetes Gerät oder eine häufig ausgeführte Schleife, die keinen Collector verträgt, ist es der falsche.

Das weniger auffällige Argument ist Zeit. Russ Cox schrieb 2023, ein Go 2, mit dem alte Programme nicht mehr funktionieren, werde „niemals kommen". Stattdessen werde das Projekt „noch stärker auf Kompatibilität setzen". Der Aufwand für dieses Versprechen fällt bei Upgrades statt bei Neufassungen an. Alle sechs Monate, im Februar und August, erscheint eine neue Hauptversion. Sie wird unterstützt, bis zwei neuere Versionen existieren. Ein Team, das Sicherheitskorrekturen erhalten möchte, muss daher ungefähr einmal im Jahr auf eine neuere Version wechseln.

Bei manchen Aufgaben verlangt die kleine Sprache eine ausführliche Schreibweise. Fehler sind gewöhnliche Werte, die dort zurückgegeben und geprüft werden, wo sie auftreten. Das ist ungefähr zu gleichen Teilen explizit und repetitiv.

Wie es in der Praxis aussieht

Eine Goroutine ist leicht gestartet und ebenso leicht verloren. Die Seite über die Verlagerung langsamer Arbeit in eine River-Jobwarteschlange sagt das deutlich: Läuft langsame Arbeit in einer Goroutine, verschwindet sie beim Neustart des Prozesses mitten in einem Deployment. Es bleibt nicht einmal ein Eintrag darüber, dass sie überhaupt stattfinden sollte. Das Unternehmen verlegte diese Arbeit nach River, dessen Warteschlange in PostgreSQL liegt. Ein Job und die von ihm berührten Zeilen werden dadurch gemeinsam committet oder zurückgerollt. Dahinter stehen 15 Worker-Module und 8 geplante Aufgaben sowie darunter 20 pg_cron-Jobs. Eine Goroutine liefert Nebenläufigkeit; Dauerhaftigkeit muss von anderer Stelle kommen.

Tests gehören zur Werkzeugkette. Die mehrschichtige Testsuite zeigt, was daraus wird. Ihre Go-API enthält 981 Testfunktionen in 310 Dateien. Sie ist eine von vier Schichten, die jeweils dort getestet werden, wo ihr Verhalten liegt, neben den Testspezifikationen des Frontends und den eigenen SQL-Tests der Datenbank.

Auch das fortlaufende Sicherheitsprogramm stützt sich auf die Werkzeuge der Sprache. Go-Abhängigkeiten werden mit govulncheck geprüft. Eine von den konfigurierten Prüfungen erkannte Schwachstelle kann den Build fehlschlagen

lassen. Die Seite nennt die Menge der Befunde als eigentlichen Aufwand des Scannens. govulncheck setzt genau bei diesem Aufwand an: Es meldet eine Schwachstelle nur, wenn die eigenen Funktionen des Codes die betroffene Funktion direkt oder über weitere Funktionen aufrufen.

Wo es schiefgeht

Eine Funktion meldet einen Fehler, obwohl keiner auftrat. Ein Interface-Wert in Go enthält einen Typ und einen Wert. Er ist nur dann gleich nil, wenn beides nicht gesetzt ist. Gibt eine Funktion einen nil-Zeiger eines konkreten Fehlertyps über das Interface `error` zurück, erhält der Aufrufer ein Interface, das einen Typ enthält und daher nicht nil ist:

```
func save() error {
    var err *ValidationError // a nil pointer
    if invalid() {
        err = &ValidationError{}
    }
    return err // never equal to nil: the interface holds a type
}
```

Geben Sie bei Erfolg direkt `nil` zurück. Deklarieren Sie jedes Fehlerergebnis als `error` statt als konkreten Typ.

Ein Dienst besteht alle Tests und stürzt unter echtem Verkehr ab. Map-Operationen sind absichtlich nicht synchronisiert: Jeden Zugriff zu sperren würde die meisten Programme verlangsamen. Die FAQ warnen, dass „unkontrollierter Map-Zugriff das Programm zum Absturz bringen kann“. Ursache ist eine Goroutine, die in eine Map schreibt, während eine andere daraus liest oder hineinschreibt. Ausschließlich lesende Zugriffe sind sicher. Schützen Sie die Map mit einem Mutex, überlassen Sie sie einer einzigen Goroutine, die die übrigen über einen Channel erreichen, oder verwenden Sie `sync.Map` für die dafür vorgesehenen Fälle. Führen Sie die Tests anschließend mit `go test -race` aus. Bei einem typischen Programm kann der Race Detector den Speicherbedarf auf das Fünf- bis Zehnfache und die Laufzeit auf das Zwei- bis Zwanzigfache erhöhen. Er findet nur Race Conditions auf tatsächlich ausgeführten Codepfaden. Eine mit `-race` gebaute Binärdatei unter realistischer Last findet deshalb mehr.

In einer Schleife gestartete Goroutinen bearbeiten alle den letzten Eintrag. Vor Go 1.22 teilten sich alle Durchläufe einer `for`-Schleife eine Variable. Eine Closure, die sie erfasste, sah daher den Wert, den die Variable bei der tatsächlichen Ausführung der Goroutine gerade enthielt. Go 1.22 gibt jedem Durchlauf eine eigene Variable, aber nur in Modulen, deren go.mod `go 1.22` oder neuer deklariert. Alter Code behält seine bisherige Bedeutung auch dann, wenn ihn eine neuere Werkzeugkette baut. Erhöhen Sie die Versionsangabe in der `go`-Zeile und führen Sie `go vet` sowie die Tests aus. Ein Test, der danach fehlschlägt, hat häufig schon vorher den falschen Wert geprüft.

Ein Build auf einer neuen Maschine hält an, um ein anderes Go herunterzuladen. Seit Go 1.21 ist die `go`-Zeile eine Anforderung. Ist die installierte Werkzeugkette älter als vom Modul verlangt, lädt der Befehl `go` die neuere als Modul herunter und führt sie aus dem Modulcache aus. Auf einem Laptop ist das praktisch; in einem vom Netz getrennten Build oder einem festgelegten Container-Image überrascht es. Installieren Sie die in go.mod genannte Werkzeugkette und setzen Sie die Umgebungsvariable `GOTOOLCHAIN`, wenn ein Build genau eine bestimmte Version verwenden muss.

Weiterführende Lektüre

- Die Go-Sprachspezifikation — Das Referenzhandbuch für jedes Konstrukt der Sprache, mit jeder Veröffentlichung versioniert.
- Häufig gestellte Fragen — Warum die Sprache entstand, ihre Entwurfsentscheidungen sowie die obigen Rätsel um nil-Fehler und Maps.
- Go 1 und die Zukunft von Go-Programmen — Das Kompatibilitätsversprechen und seine zugelassenen Ausnahmen.
- Veröffentlichungshistorie — Alle Versionen seit Go 1, mit der Supportregelung am Anfang.
- Data Race Detector — Seine Ausführung, was er findet und welchen Aufwand er verursacht.
- For-Schleifen in Go 1.22 korrigieren — Die Änderung der Schleifenvariablen, ihre Bindung an go.mod und die Prüfung alten Codes dagegen.



Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Go?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/go/