Engineer

Go

et kompileret, stærkt typet sprog med garbage collection

Hvad det er

Go er et generelt programmeringssprog, der begyndte hos Google, udformet med systemprogrammering for øje og med samtidighed bygget ind i selve sproget i stedet for lagt ovenpå som et bibliotek. Det kaldes ofte golang efter projektets første webadresse, golang.org, men projektets FAQ holder fast i, at sproget bare hedder Go, skrevet Go og aldrig GO.

Robert Griesemer, Rob Pike og Ken Thompson begyndte at skitsere målene på et whiteboard den 21. september 2007, og Go blev et offentligt open source-projekt den 10. november 2009. Go 1 kom den 28. marts 2012 med et løfte om, at programmer skrevet til den ville blive ved med at kompilere og køre, uændret, under hver senere Go 1-udgave. Løftet holder stadig: specifikationen beskriver i dag sprogversion go1.27, og Go 1.27.0 udkom den 19. august 2026. Generics landede i Go 1.18, og sprogets designere kaldte dem den største ændring siden den første open source-udgave.

Ifølge FAQ'en voksede sproget ud af et valg, forfatterne var trætte af at skulle træffe: mellem effektiv kompilering, effektiv afvikling og nem programmering. Go vil give alle tre. Et program bygges af pakker, og værktøjskæden er én kommando, `go`, som bygger, tester, formaterer og henter afhængigheder. Nøgleordet `go` foran et funktionskald starter en goroutine, en funktion, der kører side om side med resten af programmet, og goroutiner er så billige, at der ifølge FAQ'en kan være hundredtusinder af dem i ét adresserum. Kanaler bærer værdier mellem dem, og Effective Go samler den tilsigtede stil i et slogan, der på dansk lyder: »Kommunikér ikke ved at dele hukommelse; del i stedet hukommelse ved at kommunikere.«

To vaner kendetegner Go-kode, uanset hvor man møder den. Programmet `gofmt` skriver hver kildefil i ét standardlayout, så formatering aldrig bliver et emne i review. Og compileren afviser et program med en ubrugt variabel eller en ubrugt import, fordi den første kan skjule en fejl, og den anden gør hvert senere build langsommere.

Brug begrebet i en beslutning

Brug »Hvad er Go?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Den del, brugerne aldrig ser: hvor data ligger, hvilke regler der gælder, og hvordan andre systemer må tale med den.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har designet og drevet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tre databasemotorer

Har designet og drevet 6 ETL/ELT-pipelines Kontakt os →

Hvorfor det betyder noget

Det valg, Go besvarer, er som regel, hvad en netværkstjeneste, et kommandolinjeværktøj eller et stykke infrastruktur skal skrives i, og alternativerne ligger på hver sin side af det: et scriptsprog på den ene, C, C++ eller Rust på den anden.

Over for et scriptsprog tilbyder Go enkel udrulning og tidlige fejl. Linkeren laver som standard statisk linkede binærfiler, så resultatet af et build er én fil, der har sin egen runtime med og kører på en server, hvor intet andet er installeret. FAQ'en sætter prisen til et par megabyte for et program, der skriver »hello, world«, mod omkring 750 kB for det samme program i C. Typefejl viser sig, når programmet kompileres, og ikke når en sjældent brugt gren kører i produktion, og et af de oprindelige mål var, at selv en stor eksekverbar fil skulle kunne bygges på få sekunder på én computer.

Over for C, C++ eller Rust tilbyder Go enkelhed og hukommelsessikkerhed for prisen af kontrol. En garbage collector frigør hukommelse, så almindelig kode slipper for en hel klasse af use-after-free-fejl, men programmet bestemmer ikke selv, præcis hvornår hukommelsen frigives, og hver binærfil har den runtime med, der gør det. For de fleste servere er det den rigtige byttehandel. For en kerne, en lille indlejret enhed eller en ydelseskritisk løkke, der ikke tåler en collector, er det den forkerte.

Det stillere argument er tid. I 2023 skrev Russ Cox, at et Go 2, der bryder gamle programmer, aldrig kommer til at ske, og at projektet i stedet ville satse dobbelt på kompatibilitet. Løftet betales med opgraderinger frem for omskrivninger. En ny hovedudgave kommer hver sjette måned, i februar og august, og hver af dem understøttes, indtil der findes to nyere, så et team, der vil have sikkerhedsrettelser, rykker frem omtrent én gang om året.

Det lille sprog skriver også nogle ting ud i fuld længde. Fejl er almindelige værdier, der returneres og tjekkes dér, hvor de opstår, hvilket er lige dele eksplicit og gentagende.

Sådan ser det ud i praksis

En goroutine er billig at starte og lige så billig at miste. Siden om at flytte langsomt arbejde over på en River-jobkø siger det ligeud: lægger man langsomt arbejde i en goroutine, forsvinder det i det øjeblik, processen genstarter, midt i en udrulning, uden spor af, at det nogensinde skulle være sket. Virksomheden flyttede det arbejde over på River, som har sin kø i PostgreSQL, så et job og de rækker, det rører, committer eller ruller tilbage sammen. Bag den ligger 15 worker-moduler og 8 planlagte opgaver, med 20 pg_cron-jobs nedenunder. En goroutine giver samtidighed; holdbarheden må komme et andet sted fra.

Test følger med værktøjskæden, og den lagdelte testsuite viser, hvad det kan blive til. Go-API'et har 981 testfunktioner fordelt på 310 filer, ét af fire lag, der hver bliver testet dér, hvor deres adfærd hører hjemme, ved siden af frontendens specs og databasens egne SQL-test.

Det løbende sikkerhedsprogram læner sig også op ad sprogets værktøjer. Go-afhængigheder tjekkes med govulncheck, og en sårbarhed, der genkendes af de konfigurerede tjek, kan få buildet til at fejle. Siden kalder mængden af fund den reelle pris ved scanningen, og govulncheck er bygget mod netop den pris: den melder kun en sårbarhed, når kodens egne funktioner kalder den berørte funktion, direkte eller gennem andre.

Hvor det går galt

En funktion melder en fejl, hvor der ingen fejl er. En interfaceværdi i Go rummer en type og en værdi, og den er kun lig med nil, når begge mangler. En funktion, der returnerer en nil-pointer af en konkret fejltype gennem interfacet

`error`, giver kalderen et interface, der rummer en type, og det interface er ikke nil:

```
func save() error {
    var err *ValidationError // en nil-pointer
    if invalid() {
        err = &ValidationError{}
    }
    return err // aldrig lig med nil: interfacet rummer en type
}
```

Returnér et bogstaveligt `nil` ved succes, og erklær hvert fejlresultat som `error` frem for som en konkret type.

En tjeneste, der bestod alle test, går ned under rigtig trafik. Operationer på maps er bevidst ikke synkroniserede, fordi låsning af hver adgang ville gøre de fleste programmer langsommere, og FAQ'en advarer om, at ukontrolleret adgang til et map kan få programmet til at gå ned. Årsagen er, at én goroutine skriver til et map, mens en anden læser eller skriver i det; rene læsninger er sikre. Beskyt mappet med en mutex, giv det til én goroutine, som de andre når gennem en kanal, eller brug `sync.Map` i de tilfælde, den er lavet til. Kør derefter testene med `go test -race`. For et typisk program kan race-detektoren øge hukommelsesforbruget fem til ti gange og køretiden to til tyve gange, og den finder kun races på kodestier, der faktisk kører, så en binærfil bygget med `-race` og sat under realistisk belastning finder flere.

Goroutiner startet i en løkke arbejder alle på det sidste element. Før Go 1.22 havde en `for`-løkke én variabel, som alle iterationer delte, så en closure, der fangede den, så den værdi, variablen havde, da goroutinen endelig kørte. Go 1.22 giver hver iteration sin egen variabel, men kun i moduler, hvis go.mod angiver `go 1.22` eller nyere, og ældre kode beholder sin gamle betydning, selv når en nyere værktøjskæde bygger den. Hæv `go`-linjen, og kør `go vet` og testene. En test, der begynder at fejle efter ændringen, tjekkede ofte den forkerte værdi hele tiden.

Et build på en ny maskine stopper op for at hente en anden Go. Siden Go 1.21 er `go`-linjen et krav, og når den installerede værktøjskæde er ældre, end et modul beder om, henter kommandoen `go` den nyere som et modul og kører den fra modulcachen. På en bærbar er det praktisk; i et air-gapped build eller et

fastlåst containerimage er det en overraskelse. Installér den værktøjskæde, som go.mod nævner, og sæt miljøvariablen `GOTOOLCHAIN`, når et build skal bruge én bestemt version.

Læs mere

- The Go Programming Language Specification — referencemanualen for hver konstruktion i sproget, versioneret med hver udgave.
- Frequently Asked Questions (FAQ) — hvorfor sproget blev lavet, designvalgene bag det og gåderne om nil-fejl og maps ovenfor.
- Go 1 and the Future of Go Programs — kompatibilitetsløftet og de undtagelser, det tillader.
- Release History — hver udgave siden Go 1, med supportpolitikken øverst.
- Data Race Detector — hvordan den køres, hvad den fanger, og hvad den koster.
- Fixing For Loops in Go 1.22 — ændringen af løkkevariabler, hvorfor den er bundet til go.mod, og hvordan gammel kode testes mod den.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er Go?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/go/