Engineer

Go

et kompilert, sterkt typet språk med søppelinnsamling

Hva det er

Go, ofte kalt golang, er et generelt programmeringsspråk som oppsto hos Google, utformet med systemprogrammering i tankene og med samtidighet bygget inn i selve språket i stedet for lagt til som et bibliotek. Kallenavnet golang kommer fra prosjektets første nettadresse, golang.org, men prosjektets FAQ er tydelig på at språket bare heter Go, skrevet Go og aldri GO.

Robert Griesemer, Rob Pike og Ken Thompson begynte å skissere målene for språket på en tavle 21. september 2007, og Go ble et offentlig prosjekt med åpen kildekode 10. november 2009. Go 1 fulgte 28. mars 2012, med et løfte om at programmer skrevet for den versjonen skulle fortsette å kompilere og kjøre, uendret, under hver senere Go 1-utgivelse. Løftet holder fortsatt: spesifikasjonen beskriver nå språkversjonen go1.27, og Go 1.27.0 kom 19. august 2026. Generiske typer kom i Go 1.18, og språkets designere kalte dem den største endringen siden den første utgivelsen med åpen kildekode.

Ifølge FAQ-en vokste språket ut av et valg forfatterne var lei av å måtte ta, mellom effektiv kompilering, effektiv kjøring og enkel programmering. Go tar sikte på å gi alle tre. Et program bygges av pakker, og verktøykjeden er én kommando, `go`, som bygger, tester, formaterer og henter avhengigheter. Nøkkelordet `go` foran et funksjonskall starter en goroutine, en funksjon som kjører side om side med resten av programmet, og goroutiner er så billige at det ifølge FAQ-en kan leve hundretusener av dem i ett og samme adresserom. Kanaler frakter verdier mellom dem, og Effective Go oppsummerer den tiltenkte stilen i et slagord: «Do not communicate by sharing memory; instead, share memory by communicating.» På norsk: ikke kommuniser ved å dele minne, men del minne ved å kommunisere.

To vaner kjennetegner Go-kode hvor den enn dukker opp. Programmet `gofmt` skriver ut hver kildefil i ett og samme standardoppsett, så formatering blir aldri et tema i en kodegjennomgang. Og kompilatoren avviser et program med en ubrukt variabel eller en ubrukt import, fordi det første kan skjule en feil og det andre gjør hvert senere bygg tregere.

Bruk begrepet i en beslutning

Bruk «Hva er Go?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Spørsmålet Go svarer på, er som regel hva en nettverkstjeneste, et kommandolinjeverktøy eller en bit infrastruktur skal skrives i, og alternativene ligger på hver sin side av det: et skriptspråk på den ene, C, C++ eller Rust på den andre.

Sammenlignet med et skriptspråk gir Go enkel produksjonssetting og tidlige feil. Lenkeren lager statisk lenkede binærfiler som standard, så resultatet av et bygg er én fil som har med seg sitt eget kjøretidsmiljø og kjører på en server der ingenting annet er installert. FAQ-en anslår prisen til et par megabyte for et program som skriver ut «hello, world», mot rundt 750 kB for det samme programmet i C. Typefeil viser seg når programmet kompileres, og ikke når en sjelden brukt gren kjøres i produksjon, og et av de opprinnelige målene var at selv en stor kjørbar fil skulle bygges på noen få sekunder på én enkelt datamaskin.

Sammenlignet med C, C++ eller Rust gir Go enkelhet og minnesikkerhet, mot at du gir fra deg kontroll. En søppelinnsamler tar tilbake minne, så vanlig kode slipper en hel klasse feil av typen bruk etter frigjøring (use-after-free), men programmet bestemmer ikke selv nøyaktig når minnet frigjøres, og hver binærfil har med seg kjøretidsmiljøet som gjør jobben. For de fleste servere er det en god avveining. For en systemkjerne, en liten innebygd enhet eller en ytelseskritisk løkke som ikke tåler en søppelinnsamler, er den feil.

Det stillere argumentet er tid. I 2023 skrev Russ Cox at en Go 2 som bryter gamle programmer, aldri kommer til å skje, og at prosjektet i stedet skulle satse enda hardere på kompatibilitet. Løftet betales med oppgraderinger og ikke med omskrivinger. En ny hovedversjon kommer hver sjette måned, i februar og august, og hver av dem får støtte til det finnes to nyere, så et team som vil ha sikkerhetsrettinger, må gå videre omtrent én gang i året.

Det lille språket skriver også noen ting ut i full lengde. Feil er vanlige verdier som returneres og sjekkes der de oppstår, og det gjør koden eksplisitt og repeterende i omtrent like stor grad.

Slik ser det ut i praksis

En goroutine er billig å starte og like billig å miste. Siden om River-jobbkøen sier det rett ut: gjør du tregt arbeid i en goroutine, forsvinner det i det øyeblikket prosessen starter på nytt, midt i en produksjonssetting, uten spor av at det noen gang skulle skje. Arbeidet ble derfor flyttet over i River, som holder køen sin i PostgreSQL, så en jobb og radene den berører, committes eller rulles tilbake sammen. Bak den står 15 worker-moduler og 8 planlagte oppgaver, og under dem 20 pg_cron-jobber. En goroutine gir samtidighet; varig lagring må komme fra et annet sted.

Testing følger med verktøykjeden, og den lagdelte testsuiten viser hva det blir til i praksis. Go-API-et har 981 testfunksjoner fordelt på 310 filer. Det er ett av fire lag som hvert testes der oppførselen deres hører hjemme, ved siden av frontendens spesifikasjoner og databasens egne SQL-tester.

Det kontinuerlige sikkerhetsprogrammet lener seg også på verktøyene som følger med språket. Go-avhengigheter kontrolleres med govulncheck, og en sårbarhet som gjenkjennes av de konfigurerte kontrollene, kan få bygget til å feile. Siden peker på mengden funn som den egentlige kostnaden ved skanning, og govulncheck er laget for å holde den kostnaden nede: verktøyet melder en sårbarhet bare når kodens egne funksjoner kaller den berørte funksjonen, direkte eller via andre funksjoner.

Hvor det går galt

En funksjon melder en feil der alt gikk bra. En interfaceverdi i Go inneholder en type og en verdi, og den er lik nil bare når verken typen eller verdien er satt. En funksjon som returnerer en nil-peker av en konkret feiltype gjennom interfacet

`error`, gir kalleren en interfaceverdi som har en type, og den verdien er ikke nil:

```
func save() error {
    var err *ValidationError // en nil-peker
    if invalid() {
        err = &ValidationError{}
    }
    return err // aldri lik nil: interfaceverdien har en type
}
```

Returner en bokstavelig `nil` ved suksess, og deklarer hvert feilresultat som `error` og ikke som en konkret type.

En tjeneste som besto alle testene, krasjer under ekte trafikk. Operasjoner på `map` er med vilje usynkroniserte, fordi låsing ved hver tilgang ville gjort de fleste programmer tregere, og FAQ-en advarer om at ukontrollert tilgang til en `map` kan få programmet til å krasje. Årsaken er et datakappløp: én goroutine skriver til en `map` mens en annen leser fra eller skriver til den samme. Lesing alene er trygt. Beskytt den med en mutex, la én enkelt goroutine eie den og de andre nå den gjennom en kanal, eller bruk `sync.Map` i de tilfellene den er laget for. Kjør så testene med `go test -race`. For et typisk program kan race-detektoren øke minnebruken fem til ti ganger og kjøretiden to til tjue ganger, og den finner bare datakappløp i kode som faktisk kjøres, så en binærfil bygget med `-race` og satt under realistisk last finner flere.

Goroutiner som startes i en løkke, jobber alle med det siste elementet. Før Go 1.22 hadde en `for`-løkke én variabel som alle gjennomløpene delte, så en closure som fanget den, så den verdien variabelen hadde da goroutinen endelig kjørte. Go 1.22 gir hvert gjennomløp sin egen variabel, men bare i moduler der go.mod oppgir `go 1.22` eller nyere, og eldre kode beholder sin gamle betydning selv når en nyere verktøykjede bygger den. Hev `go`-linjen, og kjør `go vet` og testene. En test som begynner å feile etter endringen, har ofte sjekket feil verdi hele tiden.

Et bygg på en ny maskin stopper for å laste ned en annen Go. Siden Go 1.21 er `go`-linjen et krav, og når den installerte verktøykjeden er eldre enn det en modul ber om, henter `go`-kommandoen den nyere verktøykjeden som en modul, lagrer den blant de nedlastede modulene og kjører den derfra. På en bærbar PC er det praktisk; i et bygg som er avskåret fra nettet, eller i et fastlåst

containerbilde, kommer det som en overraskelse. Installer verktøykjeden som go.mod oppgir, og sett miljøvariabelen `GOTOOLCHAIN` når et bygg må bruke én bestemt versjon.

Les mer

- The Go Programming Language Specification — referansehåndboken for hver konstruksjon i språket, versjonert med hver utgivelse.
- Frequently Asked Questions (FAQ) — hvorfor språket ble laget, designvalgene bak det, og gåtene om nil-feil og `map` ovenfor.
- Go 1 and the Future of Go Programs — kompatibilitetsløftet og unntakene det tillater.
- Release History — hver utgivelse siden Go 1, med retningslinjene for støtte øverst.
- Data Race Detector — hvordan du kjører den, hva den fanger, og hva den koster.
- Fixing For Loops in Go 1.22 — endringen av løkkevariabelen, hvorfor den er knyttet til go.mod, og hvordan du tester gammel kode mot den.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er Go?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/go/