Engineer

Go

ett kompilerat, starkt typat språk med skräpsamling

Vad det är

Go är ett allmänt programspråk som föddes hos Google, utformat med systemprogrammering i åtanke och med samtidighet inbyggd i själva språket i stället för tillagd som ett bibliotek. Det kallas ofta golang, efter projektets första webbadress, golang.org, men projektets FAQ är bestämd: språket heter bara Go, skrivs Go och aldrig GO.

Robert Griesemer, Rob Pike och Ken Thompson började skissa på målen för språket på en whiteboardtavla den 21 september 2007, och Go blev ett offentligt projekt med öppen källkod den 10 november 2009. Go 1 följde den 28 mars 2012, med ett löfte om att program skrivna för den versionen skulle fortsätta att kompilera och köra, oförändrade, under varje senare utgåva av Go 1. Löftet håller fortfarande: specifikationen beskriver nu språkversionen go1.27, och Go 1.27.0 släpptes den 19 augusti 2026. Generiska typer kom i Go 1.18, och språkets konstruktörer kallade dem den största förändringen sedan den första utgåvan med öppen källkod.

Enligt FAQ:n växte språket fram ur ett val som upphovsmännen hade tröttnat på att göra, mellan effektiv kompilering, effektiv körning och enkel programmering. Go tar sikte på att ge alla tre. Ett program byggs av paket, och verktygskedjan är ett enda kommando, `go`, som bygger, testar, formaterar och hämtar beroenden. Nyckelordet `go` framför ett funktionsanrop startar en goroutine, en funktion som körs vid sidan av resten av programmet, och goroutiner är så billiga att hundratusentals, enligt FAQ:n, kan dela ett och samma adressrum. Kanaler förmedlar värden mellan dem, och Effective Go sammanfattar den avsedda stilen i en slogan: ”Do not communicate by sharing memory; instead, share memory by communicating.”

Två vanor känns igen i Go-kod var den än finns. Programmet `gofmt` skriver ut varje källfil i en och samma standardlayout, så formateringen blir aldrig en fråga i kodgranskningen. Kompilatorn vägrar dessutom att bygga ett program med en oanvänd variabel eller en oanvänd import, eftersom den första kan dölja en bugg och den andra gör varje senare bygge långsammare.

Använd begreppet i ett beslut

Använd ”Vad är Go?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Frågan som Go besvarar gäller oftast vad en nätverkstjänst, ett kommandoradsverktyg eller en bit infrastruktur ska skrivas i, och alternativen finns på var sin sida om det: ett skriptspråk på den ena, C, C++ eller Rust på den andra.

Jämfört med ett skriptspråk erbjuder Go enkel driftsättning och tidiga fel. Länkaren bygger statiskt länkade binärer som standard, så resultatet av ett bygge är en enda fil som bär sin egen körmiljö och kör på en server där inget annat är installerat. FAQ:n anger priset till ett par megabyte för ett program som skriver ut ”hello, world”, mot ungefär 750 kB för samma program i C. Kompilatorn visar typfel när programmet byggs och inte när en sällan använd gren körs i produktion, och ett av de ursprungliga målen var att även en stor körbar fil skulle byggas på några sekunder på en enda dator.

Jämfört med C, C++ eller Rust erbjuder Go enkelhet och minnessäkerhet till priset av kontroll. En skräpsamlare återtar minnet, så vanlig kod slipper en hel klass av misstag av typen use-after-free, men programmet bestämmer inte exakt när minnet släpps, och varje binär levereras med den körmiljö som gör jobbet. För de flesta servrar är det rätt avvägning. För en systemkärna, en liten inbyggd enhet eller en het loop som inte tål en skräpsamlare är det fel avvägning.

Det tystare argumentet är tid. År 2023 skrev Russ Cox att ett Go 2 som bryter gamla program ”is never going to happen” och att projektet i stället skulle ”double down on compatibility”. Löftet betalas med uppgraderingar snarare än omskrivningar. En ny huvudversion släpps var sjätte månad, i februari och augusti, och var och en får stöd tills två nyare finns, så ett team som vill ha säkerhetsrättningar går vidare ungefär en gång om året.

Det lilla språket kräver också att en del saker skrivs ut i sin helhet. I Go är fel vanliga värden, som returneras och kontrolleras där de uppstår, vilket är lika delar tydligt och upprepande.

Så ser det ut i praktiken

En goroutine är billig att starta och lika billig att förlora. Sidan om att flytta långsamt arbete till en jobbkö i River säger det rakt ut: lägger du långsamt arbete i en goroutine försvinner det ”i samma stund som processen startas om, vilket den kommer att göra, mitt under en driftsättning, utan några spår av att det någonsin var tänkt att ske”. Företaget flyttade arbetet till River, som håller sin kö i PostgreSQL, så att ett jobb och ”raderna det berör committas eller rullas tillbaka tillsammans”. Bakom kön finns 15 workermoduler och 8 schemalagda uppgifter, med 20 jobb i pg_cron under dem. En goroutine ger samtidighet; beständigheten måste komma från något annat håll.

Tester ingår i verktygskedjan, och den skiktade testsviten visar vad det kan bli. Dess Go-API har 981 testfunktioner fördelade på 310 filer, ett av fyra lager vid sidan av frontendens testspecifikationer och databasens egna tester i SQL. Sidan formulerar uppgiften så här: ”Varje lager behövde få sitt beteende verifierat där beteendet faktiskt finns, i stället för att allt verifierades utifrån genom en webbläsare.”

Det kontinuerliga säkerhetsprogrammet lutar sig också mot språkets verktyg. Go-beroenden skannas med govulncheck, och en sårbarhet som känns igen av de konfigurerade kontrollerna kan få bygget att misslyckas. Samma sida kallar mängden fynd för den verkliga kostnaden med skanning, och govulncheck är konstruerat mot just den kostnaden: verktyget rapporterar en sårbarhet bara när kodens egna funktioner anropar den berörda funktionen, direkt eller via andra.

Var det går fel

En funktion rapporterar ett fel fast allt gick bra. Ett gränssnittsvärde i Go håller en typ och ett värde, och det är lika med nil bara när båda saknas. En funktion som returnerar en nil-pekare av en konkret feltyp genom gränssnittet `error` ger anroparen ett gränssnitt som håller en typ, och det gränssnittet är inte nil:

```
func save() error {
    var err *ValidationError // en nil-pekare
    if invalid() {
        err = &ValidationError{}
    }
    return err // aldrig lika med nil: gränssnittet håller en typ
}
```

Returnera ett bokstavligt `nil` när allt gick bra, och deklarera varje felresultat som `error` och inte som en konkret typ.

En tjänst som klarade alla tester kraschar under verklig trafik. Operationer på en map är medvetet osynkroniserade, eftersom låsning vid varje åtkomst skulle göra de flesta program långsammare, och FAQ:n varnar för att okontrollerad åtkomst till en map kan krascha programmet. Orsaken är att en goroutine skriver till en map medan en annan läser eller skriver i den; enbart läsningar är säkra. Skydda mapen med en mutex, låt en enda goroutine äga den och de andra nå den via en kanal, eller använd `sync.Map` i de fall den är gjord för. Kör sedan testerna med `go test -race`. För ett typiskt program kan race-detektorn öka minnesanvändningen fem till tio gånger och körtiden två till tjugo gånger, och den hittar bara datakapplöpningar på kodvägar som faktiskt körs, så en binär som byggts med `-race` och körs under realistisk last hittar fler.

Goroutiner som startas i en loop arbetar alla på det sista elementet. Före Go 1.22 hade en `for` -loop en enda variabel som delades av alla iterationer, så en closure som fångade den såg det värde variabeln råkade ha när goroutinen till slut kördes. Go 1.22 ger varje iteration en egen variabel, men bara i moduler vars go.mod anger `go 1.22` eller senare, och äldre kod behåller sin gamla betydelse även när en nyare verktygskedja bygger den. Höj `go` -raden och kör `go vet` och testerna. Ett test som börjar fallera efter ändringen har ofta kontrollerat fel värde hela tiden.

Ett bygge på en ny maskin stannar för att ladda ned en annan Go. Sedan Go 1.21 är `go` -raden ett krav, och när den installerade verktygskedjan är äldre än vad en modul begär hämtar kommandot `go` den nyare som en modul och kör den från modulcachen. På en laptop är det bekvämt; i ett bygge utan nätåtkomst eller med en låst containeravbild är det en överraskning. Installera den verktygskedja som go.mod anger, och sätt miljövariabeln `GOTOOLCHAIN` när ett bygge måste använda en exakt version.

Läs mer

- The Go Programming Language Specification — referensmanualen för varje konstruktion i språket, versionerad med varje utgåva.
- Frequently Asked Questions (FAQ) — varför språket skapades, designbesluten bakom det och gåtorna om nil-fel och maps ovan.
- Go 1 and the Future of Go Programs — kompatibilitetslöftet och de undantag det tillåter.
- Release History — varje utgåva sedan Go 1, med stödpolicyn överst.
- Data Race Detector — hur du kör den, vad den fångar och vad den kostar.
- Fixing For Loops in Go 1.22 — ändringen av loopvariabeln, varför den är knuten till go.mod och hur du testar gammal kod mot den.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Go?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/go/