Ecosystem

Agenda

< 1 “Usage”>
< 2 “Build Process”>
< 3 “Central Repository”>
< 4 “Bibliotheken”>
< 5 “Testing”>

<Usage>

Go Usage

Wichtige Anwendungsbereiche: Cloud, DevOps, Web-Backend
Bekannte Projekte: Kubernetes, Docker, Terraform

<Build-Prozess>

Build-Prozess

Golang kompilierte Sprache
Kompiliert direkt in Maschinencode
Build-prozess wichtig für effizientes laufen

go build

Funktion: kompiliert den Quellcode eines Go-Programms & erstellt eine ausführbare Datei.

Details:

go build durchsucht den aktuellen Ordner nach Go-Dateien (*.go).

Falls Code fehlerfrei ist -> generiert ausführbare Datei.

Berücksichtigt auch Abhängigkeiten.

go run

Funktion: Führt ein Go-Programm direkt aus.

Details:

Der Befehl kombiniert Kompilieren & Ausführen in einem Schritt.

kein Binary im Verzeichnis hinterlassen.

go install

Funktion: Baut das Programm und installiert Binary in $GOBIN (Standardmäßig $GOPATH/bin).

Details:

genutzt, um ausführbare Programme dauerhaft zu installieren.

Falls $GOBIN im PATH enthalten ist, kann das Programm von überall ausgeführt werden.

go clean

Funktion: Entfernt generierte Dateien und Build-Artefakte, um das Projektverzeichnis sauber zu halten.

Details:

Löscht temporäre Dateien, Cache-Daten oder Build-Reste.

Hilfreich, wenn man eine frische Kompilierung sicherstellen will.

go env

Funktion: Zeigt Umgebungsvariablen an, die den Go-Build-Prozess beeinflussen.

Details:

Zeigt Werte für wichtige Go-Umgebungsvariablen wie GOPATH, GOROOT, GOOS, etc.

Kann zur Fehlerbehebung oder Konfigurationsprüfung genutzt werden.

go list

Funktion: Listet verfügbare Go-Pakete auf.

Details:
Kann genutzt werden, um Abhängigkeiten oder installierte Module zu überprüfen.
Mit Argumenten lassen sich gezielt Pakete filtern.

“

go vet

Funktion: Analysiert Code auf potenzielle Fehler und schlechte Programmierpraktiken.

Details:

- Findet Fehler wie:
  - Falsche Format-Strings bei fmt.Printf
  - Unbenutzte Variablen
  - Potenzielle Logikfehler
- Wird oft als Teil der Code-Qualitätsprüfung verwendet.

Cross-Kompilierung

Go unterstützt Cross-Kompilierung nativ.

Entwickler können Programme für verschiedene Plattformen mit Umgebungsvariablen wie GOOS und GOARCH kompilieren:

GOOS=linux GOARCH=amd64

Terminal

```
GOOS=windows GOARCH=amd64 go build
```

<Central Repository>
Go Module

Central Repository

zentraler Speicherort
Softwarepakete, Bibliotheken oder Abhängigkeiten gespeichert und verwaltet

Hauptquelle für Entwickler, um benötigte Abhängigkeitenzu verwenden.

Go Module

Paketverwaltungssystem, 2018 in Go 1.11 eingeführt

Ort, an dem Go-Module gespeichert, abgerufen werden

Proxy für die Go-Modul-Speicherung

Abhängigkeiten effizient zu verwalten

Zwischenablage der verschiedenen Versionen eines Moduls

speichert die Module und deren Versionen in einer Art Cache.

Warum Go Module?

Unzuverlässigkeit von externen Repositories:
mussten auf externe GitHub- oder andere Repositories verlassen
-> offline
-> Eingeschränkter Betrieb

Schnelligkeit und Effizienz:
Proxy hilft, Module schneller zu laden,
gecachte Module verwendet statt vom Original-Repository zu laden.

Sicherheitsaspekte: Durch das Caching und Überprüfen von Modulen wird die Integrität der Module gewährleistet.

Wie funktioniert Go Module?

Die Abhängigkeiten werden in einer go.mod Datei verwaltet.
Die go.sum Datei speichert Integrität

Befehle	Beschreibung
go mod init <modulname>	Erstellt eine go.mod Datei und initialisiert das Modul.
go mod tidy	Entfernt unbenutzte Abhängigkeiten und lädt fehlende nach.
go get <paket>	Fügt eine neue Abhängigkeit hinzu oder aktualisiert eine vorhandene.
go list	Listet alle Module des Projekts auf.

Alternativen und Privatproxies

Go Proxy von anderen Anbietern oder private Proxys,
speziell für bestimmte Unternehmen eingerichtet

eigene Proxy-Server betreiben, um ihre Module innerhalb eines Unternehmens zu verwalten.

Best Practices

Vermeidung unnötiger Abhängigkeiten: Verwenden von go mod tidy, um ungenutzte Pakete zu entfernen.

Regelmäßiges Testen: Vor jedem Build go test ./... ausführen, um Fehler frühzeitig zu erkennen.

Automatisierte Builds: Nutzung von CI/CD-Pipelines, um Builds und Tests zu automatisieren.

Verwendung von Versionierung: Nutzen von go mod für eine zuverlässige Verwaltung der Abhängigkeiten.

<Bibliotheken>

Standardbibliothek

Umfangreiche und leistungsfähige Standardbibliothek

Erfasst Themen wie:

- Allgemeine Pakete
- Datenstrukturen & Algorithmen
- Netzwerk & Web
- Kryptographie & Sicherheit
- Parallelität

https://pkg.go.dev/std

Third-Party-Bibliotheken & Frameworks

Web-Frameworks	(Gin, Echo, Fiber)
Datenbanken & ORM	(GORM, sqlx, ent)
HTTP-Clients & API-Requests	(resty, goretryablehttp, graphql)
CLI-Entwicklung	(cobra, urfave/cli)
Logging	(logrus, zap, zerolog)
Konfiguration & ENV	(viper, godotenv)
Testing & Mocking	(testify, ginkgo, go-sqlmock)
Parallelität & Background Task	(workerpool, go-workers)
Security & Verschlüsselung	(jwt-go, argon2, bcrypt)

<Testing>

Unittests mit go test

Tests werden in Dateien mit _test.go am Ende des Namens gespeichert und mit go test ausgeführt.

test.go

```go
package main
import "testing"

func TestAddition(t *testing.T) {
    result := 2 + 3
    if result != 5 {
        t.Errorf("Erwartet 5, aber
erhalten %d", result)
    }
}
```

Benchmarking

Go unterstützt Benchmark-Tests nativ. Diese werden mit go test -bench . ausgeführt.

benchmark.go

```go
func BenchmarkAddition(b
*testing.B) {
    for i := 0; i < b.N; i++ {
        _ = 2 + 3
    }
}
```

Code Coverage

Mit go test -cover kann die Testabdeckung gemessen werden

Wird in durchgangenen Lines gemessen.

Terminal

```
go test -cover
```

Erweiterte Teststrategien

Tabelle-getriebene Tests: Erlaubt das Testen mehrerer Eingaben mit weniger Code.

Advanced_test.go

```go
func TestMultiplication(t *testing.T) {
    tests := []struct {
        a, b, expected int
    }{
        {2, 3, 6},
        {0, 5, 0},
        {-1, 8, -8},
    }

    for _, tt := range tests {
        result := tt.a * tt.b
        if result != tt.expected {
            t.Errorf("Erwartet %d, aber erhalten %d",
           tt.expected, result)
        }
    }
}
```

RPN calculator Test

<noch Fragen?>

Quellen

https://pkg.go.dev/std
https://de.wikipedia.org/wiki/Go_(Programmiersprache)
https://en.wikipedia.org/wiki/Software_repository
https://go.dev/doc/effective_go
https://go.dev/doc/tutorial/add-a-test
https://pkg.go.dev/golang.org/x/build
https://pkg.go.dev/cmd/go
https://images.app.goo.gl/pPnxYwouzUU4Wtfk8
https://images.app.goo.gl/uay1oGmTNQ6UKXt56
https://en.wikipedia.org/wiki/Cross_compiler
https://opensource.com/article/21/1/go-cross-compiling
https://go.dev/doc/cmd
https://pkg.go.dev/testing
https://blog.jetbrains.com/go/2022/11/22/comprehensive-guide-to-testing-in-go/