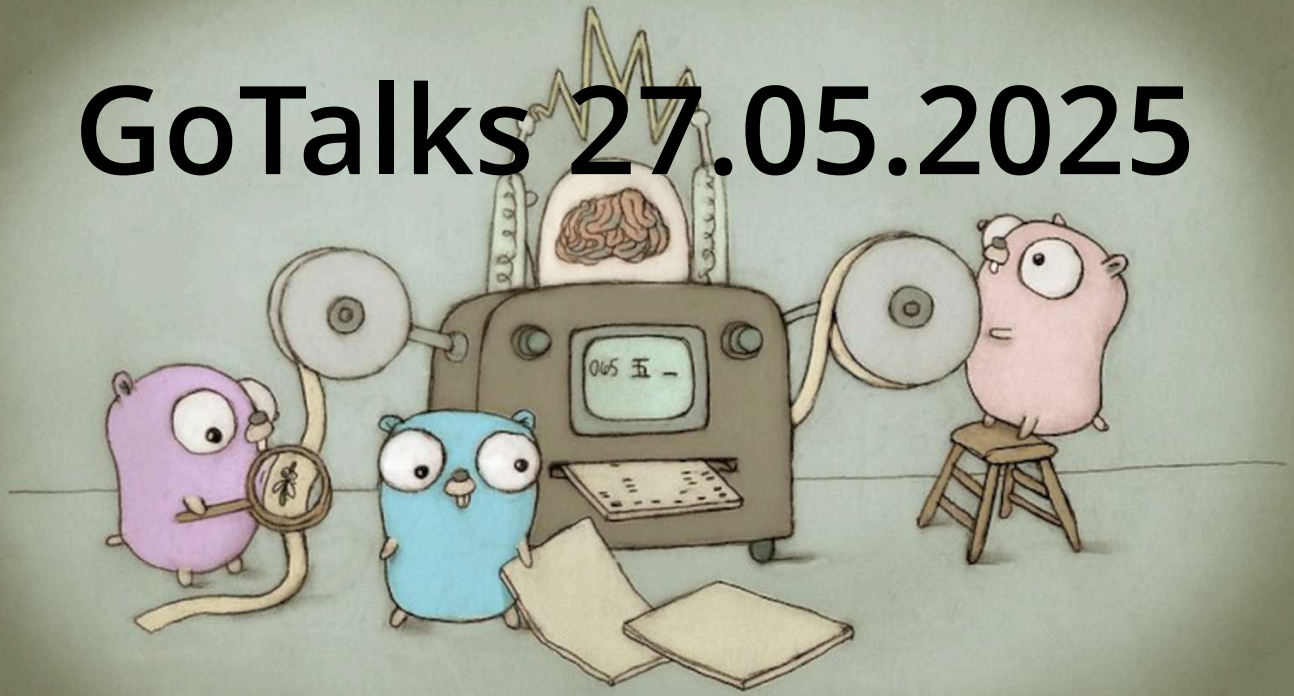
GoTalks 27.05.2025



How to reach us

meetup.com/Golang-ZG

@golangzg

github.com/golanghr/golangzg

@golangzg.bsky.social

invite.slack.golangbridge.org

modules and packages

Modules

- Go 1.13+
  - Semver - Semantic versioning (`MAJOR.MINOR.PATCH`)
- options
  - 1 repository => 1+ module
  - 1 module => 1+ packages
  - 1 package => 1+ files (same folder)

recommendation:

- 1 repository => 1 module

how to start the project

- Application

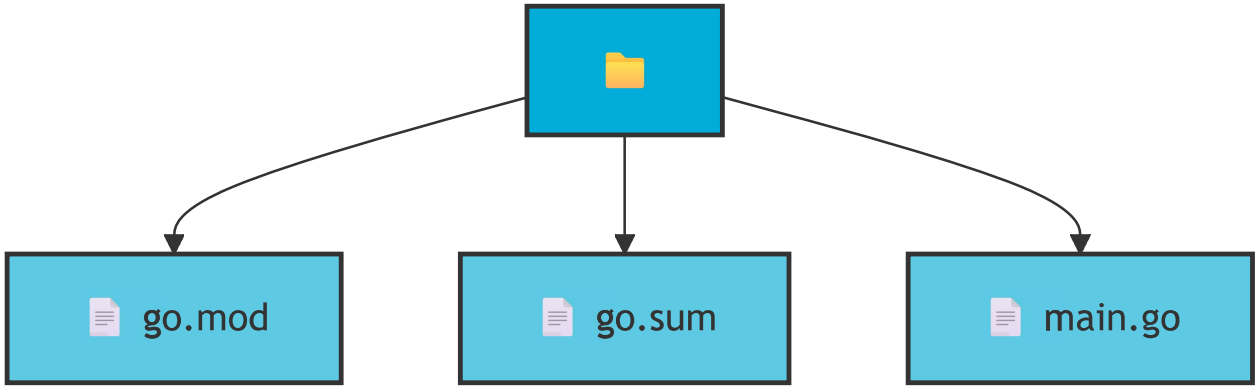
```
go mod init my-app
```

```
go mod init github.com/org/my-app
```

- Library

```
go mod init github.com/org/my-lib
```

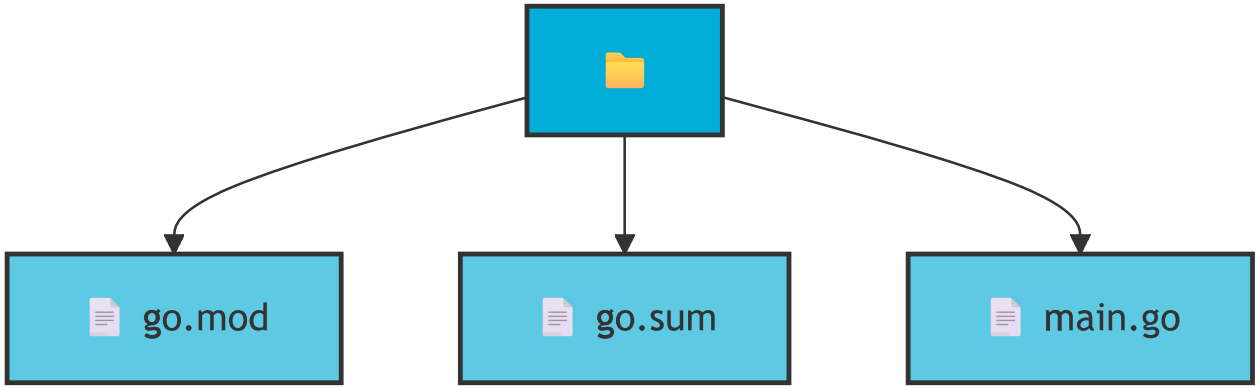
single folder module



my-app | my-lib

```
package main
```

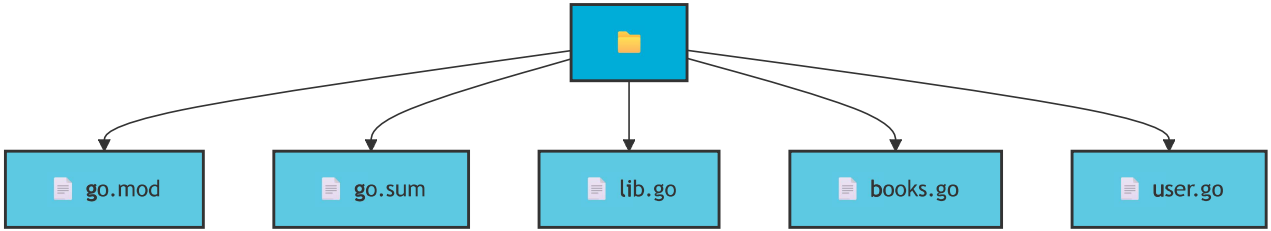
single folder module



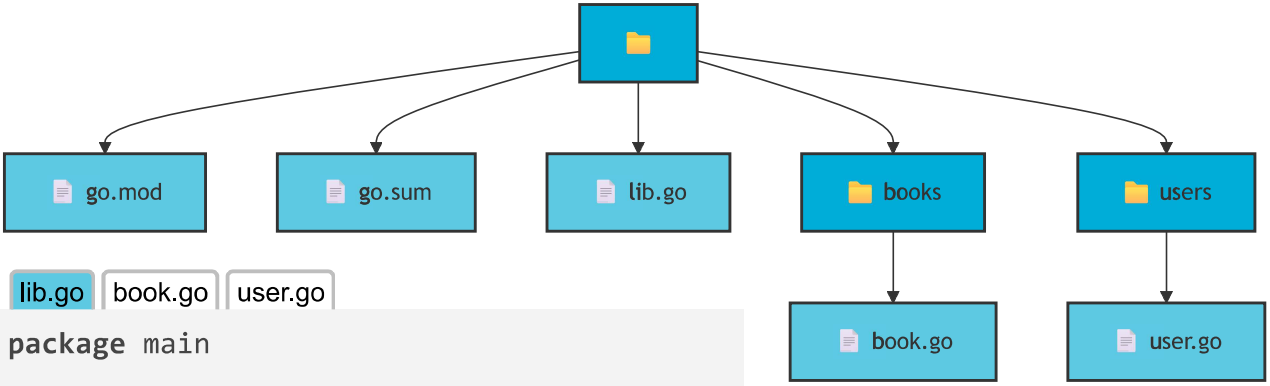
my-app | my-lib

```
package github.com/org/my-lib
```

single folder module



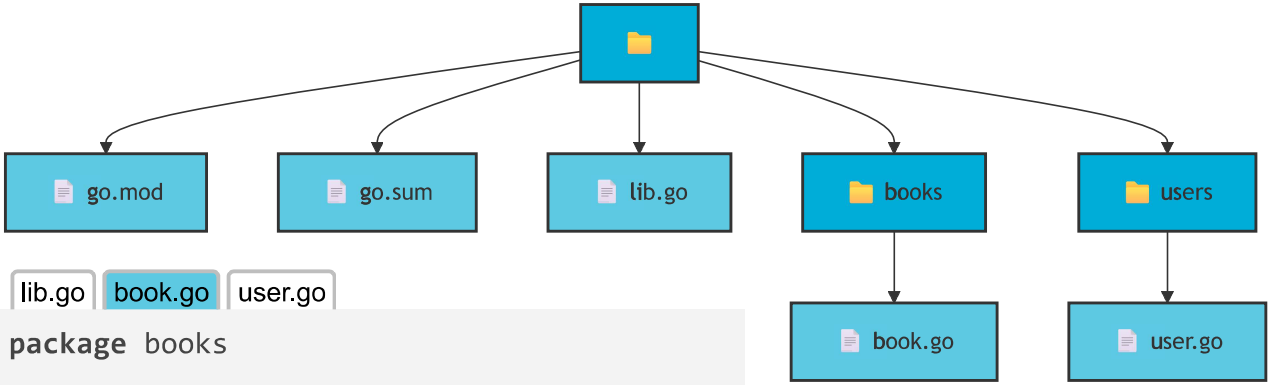
multi package module



lib.go | book.go | user.go

```go
package main

import (
  "fmt"

  "github.com/org/my-app/books"
  "github.com/org/my-app/users"
)
```

multi package module

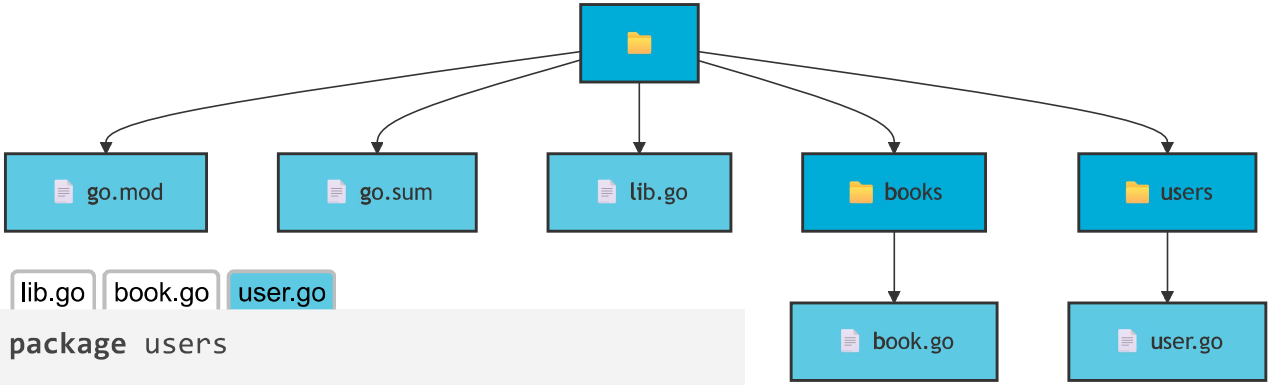


lib.go | book.go | user.go

```go
package books

type Book struct {
  ID     int
  Title  string
  Author string
}
```

GolangZG

multi package module



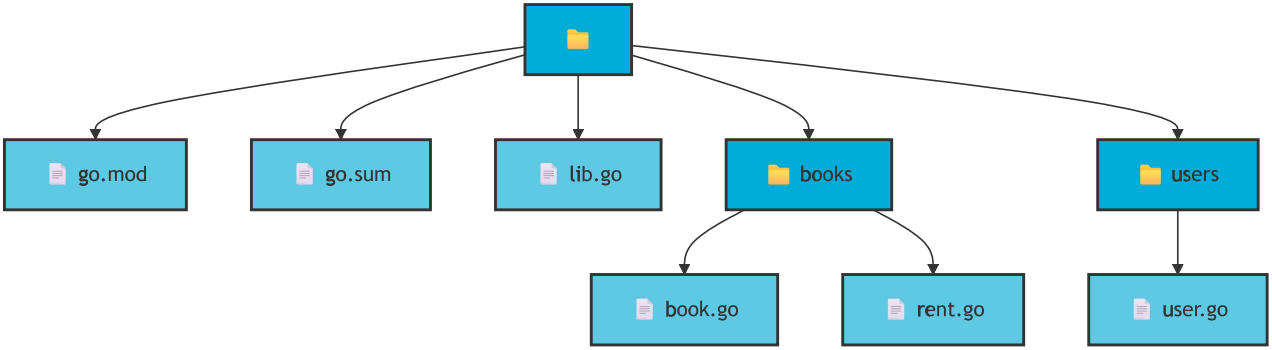
lib.go | book.go | user.go

```go
package users

type User struct {
  ID     int
  Name   string
  Email string
}
```

packages - lib.go

```
import (
    "fmt"

    "github.com/org/my-lib/books"
    "github.com/org/my-lib/users"
)

func main() {
    book := books.Book{
        ID:     1,
        Title:  "Go Programming",
        Author: "Alice Gopher",
    }
    user := users.User{
        ID:    1,
        Name:  "Bob Smith",
        Email: "bsmith@domain.com",
    }
    fmt.Println("Book:", book.Title, "by", book.Author)
    fmt.Println("User:", user.Name+",", user.Email)
}
```

packages - `renting`



GO packages - renting

```
package books
```

```
func Borrow(book Book, user users.User) error {
    library.Orders = append(library.Orders, Renting{
        ID:     len(library.Orders) + 1,
        BookID: book.ID,
        UserID: user.ID,
        Start:  time.Now(),
    })
    return nil
}
```

packages - `users` - import cycle

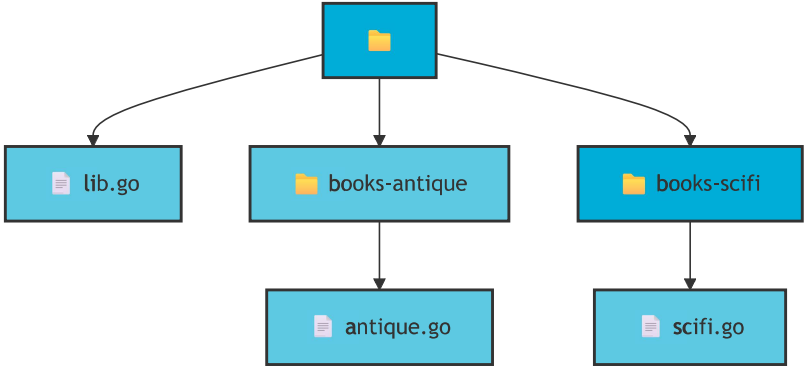
```go
package users

import "github.com/org/my-lib/books"

type User struct {
    ID    int
    Name  string
    Email string
}

func (u *User) GetRented() []books.Book {
    return []books.Book{}
}
```

packages - `users` - import cycle

- solutions
  - move code to a new package (`renting`)
  - move types to a new package (`types`)
  - use interfaces to break the cycle
  - ...

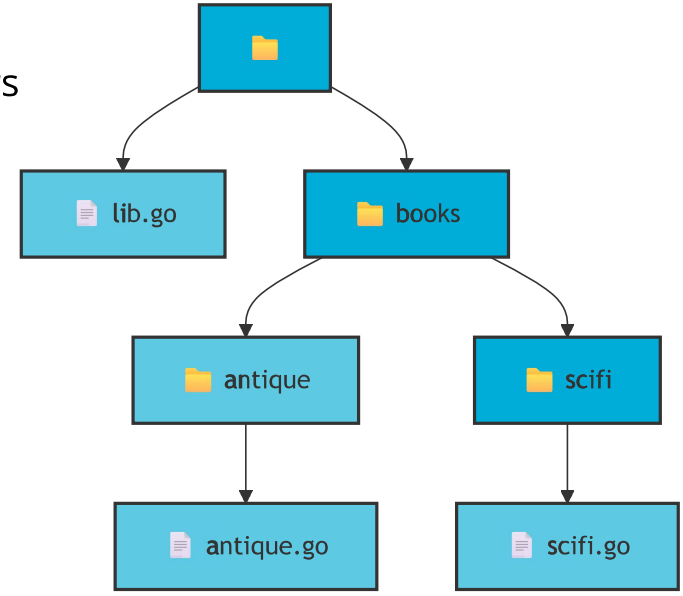
naming packages

- package name
  - short and descriptive
  - lower case
  - no dash or mixed case



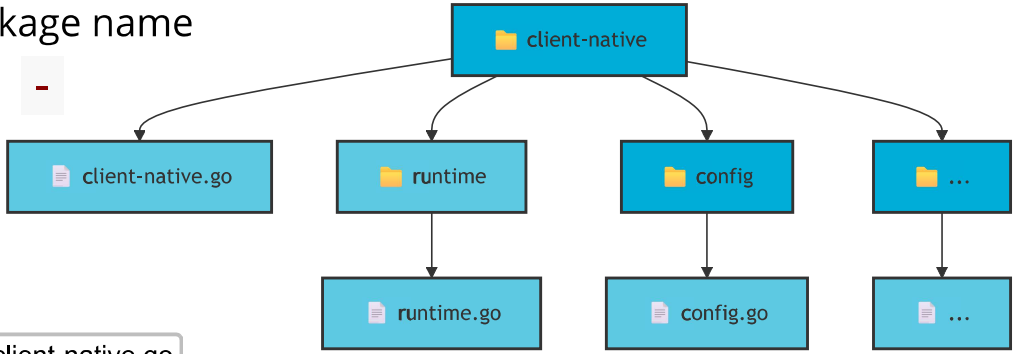
naming packages

- package name
  - subfolders



naming packages

- package name
  - -

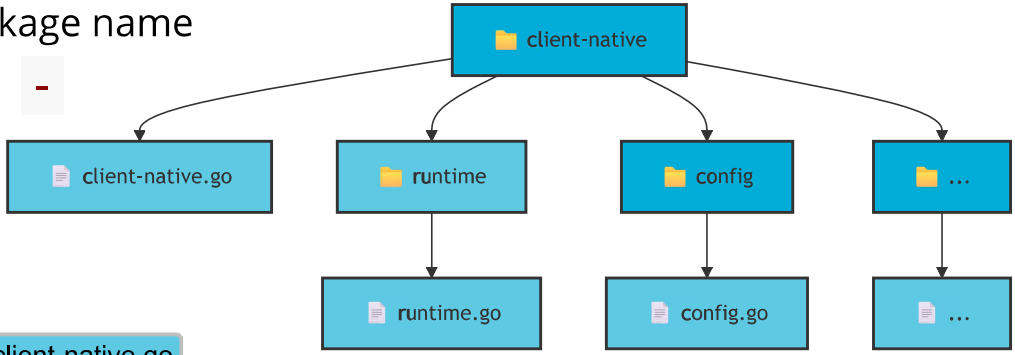


go.mod | client-native.go

```
module github.com/haproxytech/client-native/v6

go 1.24

require (
```

naming packages

- package name
  - -



go.mod | client-native.go

```
package clientnative

import (
    "errors"
    ...
```

module documentation - pkg.go.dev

README ¶

Go Ordered map

Ordered map

Ordered map preserves order of insertion in map. you can fetch data directly, or iterate in loop over them. compared to regular map, iterating will always happend in same order

Example

see Example, map[int]string test and map[string]struct test

Expand ▾

Documentation

Overview

▾ Example

```
m := orderedmap.New[string, customType]()

// add 3 values to map
m.Set("1", customType{1, "1"}) // map[1:{1 1}]
m.Set("2", customType{2, "2"}) // map[1:{1 1} 2:{2 2}]
m.Set("3", customType{3, "3"}) // map[1:{1 1} 2:{2 2} 3:{3 3}]

// delete one with key "2"
m.Delete("2") // map[1:{1 1} 3:{3 3}]

// insert to specific index
err := m.Insert(1, "42", customType{42, "42"}) // map[1:{1 1} 3:{3 3} 42:{42 42}] order:[1 42 3]
if err != nil {
    fmt.Println(err)
}

c := m.Cursor()
for c.Reset(); c.Valid(); c.Next() { // c.Reset() is not needed on first run
```

module documentation - pkg.go.dev

doc_test.go

```
package orderedmap_test

import (
  "fmt"

  orderedmap "github.com/oktalz/ordered-map"
)

func Example() {
  m := orderedmap.New[string, customType]()
  ...
```

module documentation - pkg.go.dev

doc_test.go

```
// This is a package-level example
func ExampleCursor() {
  m := orderedmap.New[string, customType]()

  m.Set("1", customType{1, "1"})

  ...
}
```

GO structure

- big or small modules / packages ?
  - Go's dead code elimination (DCE) is good, especially for functions and variables that are `truly unreachable`.
  - Link-time elimination of unused functions
  - Multiple passes ?
    - DCE is often performed at various stages of the compilation and linking process, increasing its effectiveness

GO structure

- `Limitations`
  - Reflection can hinder DCE !!
    - relies on a static view of the program’s call graph
    - linker must assume that any exported methods might be invoked at runtime
  - Exported variables
  - `Unreachable` vs. `Unused`
    - Go's DCE primarily focuses on `unreachable` code