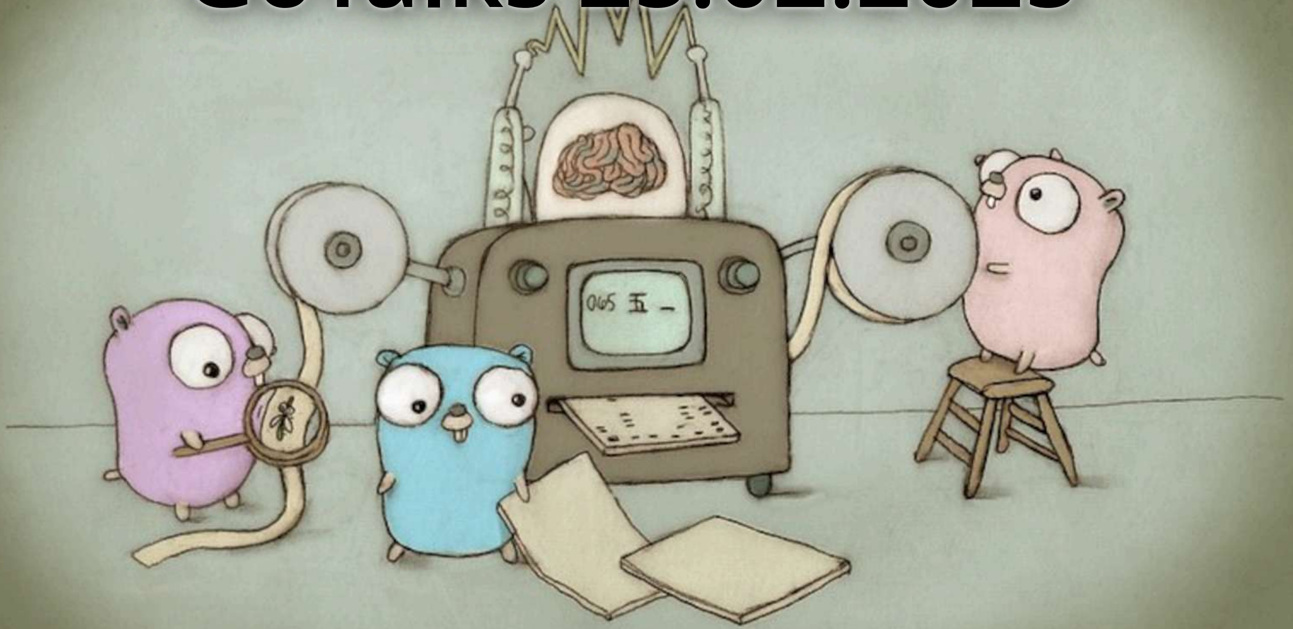
GoTalks 25.02.2025



How to reach us

meetup.com/Golang-ZG

@golangzg

github.com/golanghr/golangzg

@golangzg.bsky.social

invite.slack.golangbridge.org

GolangZG

Tehničko veleučilište u Zagrebu

- tvz.hr
    - Borongajska cesta 83E, 10000, Zagreb, Croatia
    - maps.app.goo.gl/1jKZDsiqMbhZc7136

GolangZG

FOSDEM video recordings

Open Source Developers’ European Meeting

- Brussels / 1 & 2 February 2025
- fosdem.org
- fosdem.org/2025/ud2120/

GolangZG

go tool

```
module myapp

go 1.24.0

tool (
        github.com/google/go-licenses
        golang.org/x/vuln/cmd/govulncheck
        mvdan.cc/gofumpt
)
```

```
go get -tool golang.org/x/vuln/cmd/govulncheck
```

```
go tool govulncheck -show verbose ./...
```

go tool

- advantages:
    - no need for external tooling (or extra installations)
    - automatically track versions
    - update is similar as any other package
- disadvantages:
    - dependencies mixed with code dependencies
        - unexpected behavior of application / tool
    - each time `go tool` is called, tool is compiled
- `-modfile`
    - separate file

```
go get -tool -modfile tools/go.mod golang.org/x/vuln/cmd/govulncheck
```

GolangZG

strings & bytes

```go
text := "Hello\nWorld\nGo Programming\n"

for _, line := range strings.Split(text,"\n") {
    fmt.Printf("%q\n", line)
}

fmt.Println("================")

lines := strings.FieldsFunc(text, func(r rune) bool {
  return r == '\n'
})

for _, field := range lines {
  fmt.Printf("%q\n", field)
}
```

strings & bytes

The `strings` package adds several functions that work with iterators:

- `Lines` returns an iterator over the newline-terminated lines in a string.
- `SplitSeq` returns an iterator over all substrings of a string split around a separator.
- `SplitAfterSeq` returns an iterator over substrings of a string split after each instance of a separator.
- `FieldsSeq` returns an iterator over substrings of a string split around runs of whitespace characters, as defined by unicode.IsSpace.
- `FieldsFuncSeq` returns an iterator over substrings of a string split around runs of Unicode code points satisfying a predicate.

```go
text := "Hello\nWorld\nGo Programming\n"
for line := range strings.Lines(text) {
    fmt.Printf("%q\n", line)
}
```

Not a type alias

```go
type myEnum string

const (
    MY_ENUM_FOO myEnum = "foo"
    MY_ENUM_BAR myEnum = "bar"
)

func main() {
    a := MY_ENUM_FOO
    s := "foo"

    // cannot use s (variable of type string) as myEnum value in assignment
    // a = s
    a = myEnum(s)
    fmt.Printf("a is %T with value %v\n", a, a)
}
```

Generic type aliases

```go
type Set[T comparable] = map[T]struct{}

func Add[T comparable](set Set[T], value T) Set[T] {
    set[value] = struct{}{}
    return set
}

func main() {
    set := Set[int]{
        0: {},
        1: {},
    }

    set = Add(set, 1)
    set = Add(set, 2)

    fmt.Printf("set is %T\n", set)
    fmt.Printf("set = %v\n", Values(set))
}
```

Weak pointers

```go
func main() {
    data := "TVZ"
    p := &data

    // Make creates a weak pointer from a pointer to some value of type T.
    // func Make[T any](ptr *T) Pointer[T] {
    w := weak.Make(p)

    d := w.Value()
    PrintPtr(d)

    runtime.GC()

    d = w.Value()
    PrintPtr(d)
}
```

GolangZG

Weak pointers

```go
func main() {
	p := &Location{
		Name:      "TVZ",
		Latitude:  45.810955326640254,
		Longitude: 16.04167597609013,
	}
	w := weak.Make(p)

	d := w.Value()
	PrintLocation(d)
	_ = p // not really necessary

	runtime.GC()

	d = w.Value()
	PrintLocation(d)
}
```

GolangZG

AddCleanup

- The new `runtime.AddCleanup` function
  - finalization mechanism that is more flexible, more efficient, and less error-prone than `runtime.SetFinalizer`.
- AddCleanup attaches a cleanup function to an object that will run once the object is no longer reachable.
  - unlike SetFinalizer,
    - multiple cleanups may be attached to a single object
    - cleanups may be attached to interior pointers
    - cleanups do not generally cause leaks when objects form a cycle,
    - cleanups do not delay the freeing of an object or objects it points to.
- New code should prefer AddCleanup over SetFinalizer.

GolangZG

Weak pointers + AddCleanup

Cache | Cache Add | Cache Get | Usage

```go
type Cache struct {
	m  map[int]weak.Pointer[string]
	mu sync.RWMutex
}

func NewCache() *Cache {
	return &Cache{
		m: map[int]weak.Pointer[string]{},
	}
}
```

- nothing smart, just keep a string for certain int

GolangZG

Weak pointers + AddCleanup

Cache | Cache Add | Cache Get | Usage

```go
func (c *Cache) Add(key int, value *string) {
	c.mu.Lock()
	defer c.mu.Unlock()
	wp := weak.Make(value)
	c.m[key] = wp
	// CAUTION: runtime.AddCleanup runs once the object
	// is no longer reachable, but not immediately
	// when the object is no longer reachable
	runtime.AddCleanup(&value, func(key int) {
		c.mu.Lock()
		defer c.mu.Unlock()
		fmt.Println("value for key", key, "removed")
		delete(c.m, key)
	}, key)
}
```

- on adding, automatically handle deletion too

Weak pointers + AddCleanup

Cache | Cache Add | **Cache Get** | Usage

```go
func (c *Cache) Get(key int) (string, bool) {
	c.mu.RLock()
	defer c.mu.RUnlock()
	wp, ok := c.m[key]
	if !ok {
		return "", false
	}
	valPtr := wp.Value()
	if valPtr == nil {
		return "", false
	}
	return *valPtr, true
}
```

GO Weak pointers + AddCleanup

Cache | Cache Add | Cache Get | Usage

```go
cache := NewCache()

str := "Zagreb"
cache.Add(1, &str)

_, ok := cache.Get(1)
fmt.Println("cached value OK:", ok)

runtime.GC() // real work simulated

_, ok = cache.Get(1)
fmt.Println("cached value OK:", ok)
```

GolangZG

Weak pointers + AddCleanup

```go
cache := NewCache()

str := "Zagreb"
cache.Add(1, &str)

str = "Prague"
cache.Add(1, &str)

value, ok := cache.Get(1)
fmt.Println("str:", str)
fmt.Println("cached value OK:", ok, value)

runtime.GC() // real work simulated

value, ok = cache.Get(1)
fmt.Println("cached value OK:", ok, value)

runtime.GC()
```

Swiss tables

- abseil.io/about/design/swisstables
    - more efficient memory allocation of small objects
    - new runtime-internal mutex implementation
    - performance improvements can be expected

Directory-scoped filesystem access

- The new `os.Root` type
  - provides the ability to perform filesystem operations within a specific directory.
- The `os.OpenRoot` function opens a directory and returns an `os.Root`.
  - Methods on `os.Root` operate within the directory and do **not** permit paths that refer to locations outside the directory, including ones that follow symbolic links out of the directory.
  - Methods on `os.Root` mirror most of the file system operations available in the os package, including for example `os.Root.Open`, `os.Root.Create`, `os.Root.Mkdir`, and `os.Root.Stat`

GolangZG

FIPS 140-3 compliance

- wikipedia.org/wiki/FIPS_140-3
  - The Federal Information Processing Standard Publication 140-3 is a U.S. government computer security standard used to approve cryptographic modules.
- go.dev/doc/security/fips140
- The Go Cryptographic Module is a set of internal standard library packages that are transparently used to implement FIPS 140-3 approved algorithms.
  - Applications require no changes to use the Go Cryptographic Module for approved algorithms.
- The new `GOFIPS140` environment variable can be used to select the Go Cryptographic Module version to use in a build.
- The new `fips140` GODEBUG setting can be used to enable FIPS 140-3 mode at runtime.
- Go 1.24 includes Go Cryptographic Module version v1.0.0

Go 1.24

- golang.org/doc/go1.24
- improvements to the runtime have decreased CPU overheads by `2-3%` on average
- #cgo `noescape`
  - compiler => memory passed to the C func does not escape.
- #cgo `nocallback`
  - compiler => C func does not call back to any Go functions
- `omitzero` - unlike `omitempty`, `omitzero` omits zero-valued `time.Time` values
- ```
  go test -json
  ```
- ```
  go install -json
  ```
- ```
  go build -json
  ```
- ...
  ```

https - deep dive