What’s new in Go

February 2023

Go 1.20 release

- Language changes
- Tool improvements
- Standard library additions
- Compiler: Preview of profile-guided optimization

Go release cycle.
Currently supported versions: Go 1.19 and 1.20

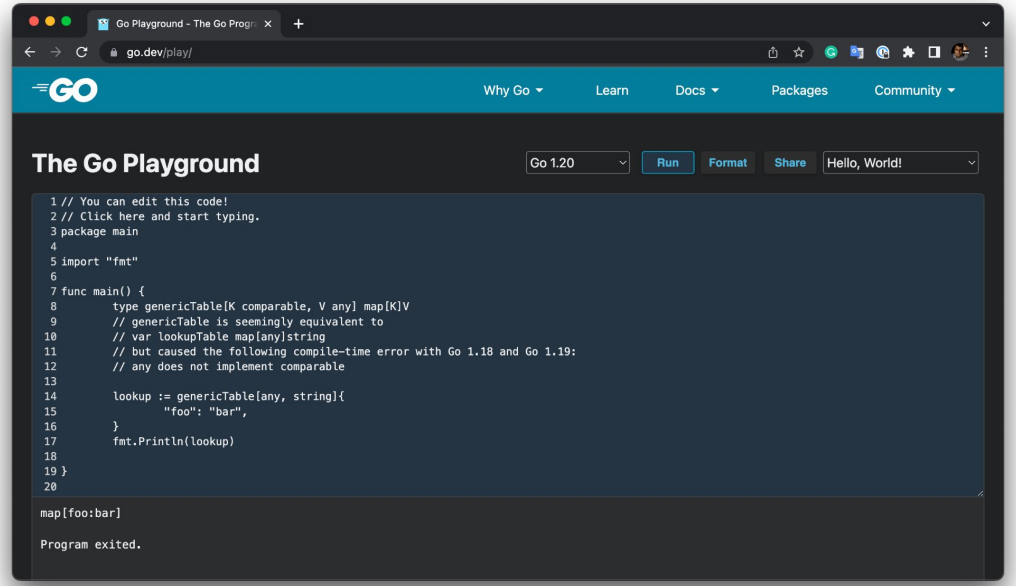
References:

- https://github.com/golang/go/wiki/Go-Release-Cycle
- https://go.dev/doc/go1compat
- https://github.com/golang/go/discussions/55090
- https://github.com/golang/go/discussions/55092

Language changes

comparable constraint is now satisfied by ordinary comparable types, such as any (a.k.a. interface{})

Reference: https://go.dev/blog/comparable



unsafe package

Functions added to the package:

- unsafe.SliceData
- unsafe.String
- unsafe.StringData

They complete the set of functions for implementation-independent slice and string manipulation.

Direct conversion from array to slice

Go Playground - The Go Progr

go.dev/play/

GO

Why Go | Learn | Docs | Packages | Community

The Go Playground

Go 1.20 | Run | Format | Share | Hello, World!

```go
package main

import "fmt"

func main() {
	slice := []string{"a", "b", "c"}
	array := [3]string(slice)

	// Until Go 1.19, you would get:
	// ./prog.go:9:21: cannot convert slice (variable of type []string) to type [3]string
	// But now your code compiles just fine!
	fmt.Printf("slice=%v type=%T\n", slice, slice)
	fmt.Printf("array=%v type=%T\n", array, array)
}
```

Reference:
https://go.dev/ref/spec#Conversions_from_slice_to_array_or_array_pointer

```
slice=[a b c] type=[]string
array=[a b c] type=[3]string

Program exited.
```

Go specification finally defines the exact order in which array elements and struct fields are compared

This clarifies what happens in case of panics during comparisons.

Reference:
https://go.dev/ref/spec#Comparison_operators

Performance improvements

Performance improvements

- Compiler and garbage collector have, once again, reduced memory overhead and improved overall CPU performance by 2%
- Work on compilation time led to build improvements bringing build speeds back in line with Go 1.17

Tool improvements

- Cover tool
- $GOROOT/pkg no longer ships with pre-compiled standard library packages
- $ go test -json is more robust in the presence of stray writes to stdout
- go build, go install, etc., new flags: -pgo and -cover
- The go command now disables cgo by default on systems without a C toolchain
- go vet improvements leading to, among others, more loop variable reference mistakes

Coverage profiling support for integration tests

Back to Go Testing

Table of Contents:

Reference:
https://go.dev/testing/coverage/

Beginning in Go 1.20, Go supports collection of coverage profiles from applications and from integration tests, larger and more complex tests for Go programs.

Overview

Go provides easy-to-use support for collecting coverage profiles at the level of package unit tests via the "`go test -coverprofile=... <pkg_target>`" command. Starting with Go 1.20, users can now collect coverage profiles for larger integration tests: more heavy-weight, complex tests that perform multiple runs of a given application binary.

For unit tests, collecting a coverage profile and generating a report requires two steps: a `go test -coverprofile=...` run, followed by an invocation of `go tool cover {-func,-html}` to generate a report.

For integration tests, three steps are needed: a build step, a run step (which may involve multiple invocations of the binary from the build step), and finally a reporting step, as described below.

Building a binary for coverage profiling

To build an application for collecting coverage profiles, pass the `-cover` flag when invoking `go build` on your application binary target. See the section below for a sample `go build -cover` invocation. The resulting binary can then be run using an environment variable setting to capture coverage profiles (see the next section on running).

Running a coverage-instrumented binary

Binaries built with "`-cover`" write out profile data files at the end of their execution to a directory specified via the environment variable `GOCOVERDIR`. Example:

```
$ go build -cover -o myprogram.exe myprogram.go
$ mkdir somedata
$ GOCOVERDIR=somedata ./myprogram.exe
I say "Hello, world." and "see ya"
$ ls somedata
covcounters.c6de772f99010ef5925877a7b05db4cc.2424989.1670252383678349347
covmeta.c6de772f99010ef5925877a7b05db4cc
$
```

Note the two files that were written to the directory `somedata`: these (binary) files contain the coverage results. See the following section on reporting for more on how to produce human-readable results from these data files.

If the `GOCOVERDIR` environment variable is not set, a coverage-instrumented binary will still execute correctly, but will issue a warning. Example:

```
$ ./myprogram.exe
warning: GOCOVERDIR not set, no coverage data emitted
I say "Hello, world." and "see ya"
$
```

Tests involving multiple runs

Integration tests can in many cases involve multiple program runs; when the program is built with "`-cover`", each run will produce a new data file. Example

```
$ mkdir somedata2
$ GOCOVERDIR=somedata2 ./myprogram.exe          // first run
I say "Hello, world." and "see ya"
$ GOCOVERDIR=somedata2 ./myprogram.exe -flag    // second run
I say "Hello, world." and "see ya"
$ ls somedata2
covcounters.890814fca98ac3a4d41b9bd2a7ec9f7f.2456041.1670259309405583534
covcounters.890814fca98ac3a4d41b9bd2a7ec9f7f.2456047.1670259309410891043
covmeta.890814fca98ac3a4d41b9bd2a7ec9f7f
$
```

Coverage data output files come in two flavors: meta-data files (containing the items that are invariant from run to run, such as source file names and function names), and counter data files (which record the parts of the program that executed).

In the example above, the first run produced two files (counter and meta), whereas the second run generated only a counter data file: since meta-data doesn't change from run to run, it only needs to be written once.

```
$ ls somedata
covcounters.c6de772f99010ef5925877a7b05db4cc.2424989.1670252383678349347
covmeta.c6de772f99010ef5925877a7b05db4cc
$ go tool covdata percent -i=somedata
    main    coverage: 100.0% of statements
    mydomain.com/greetings  coverage: 100.0% of statements
$
```

```
$ go tool covdata
error: missing command selector
usage: go tool covdata [command]

Commands are:

textfmt     convert coverage data to textual format
percent     output total percentage of statements covered
pkglist     output list of package import paths
func        output coverage profile information for each function
merge       merge data files together
subtract    subtract one set of data files from another set
intersect   generate intersection of two sets of data files
debugdump   dump data in human-readable format for debugging purposes

For help on a specific subcommand, try:

go tool covdata <cmd> -help
```

Standard library additions

- crypto/ecdh package
- errors.Join
- http.ResponseController
- httputil.ReverseProxy includes a new Rewrite hook function, superseding the Director hook
- context.WithCancelCause
- os/exec.Cmd fields Cancel and WaitDelay

context.WithCancelCause

Now you can cancel a context with a cause.

The cause can be retrieved using ctx.Cause()

type CancelCauseFunc

```
type CancelCauseFunc func(cause error)
```

cancel(err) instead of cancel()

func WithCancelCause

added in go1.20

```
func WithCancelCause(parent Context) (ctx Context, cancel CancelCauseFunc)
```

WithCancelCause behaves like WithCancel but returns a CancelCauseFunc instead of a CancelFunc. Calling cancel with a non-nil error (the "cause") records that error in ctx; it can then be retrieved using Cause(ctx). Calling cancel with nil sets the cause to Canceled.

Example use:

```
ctx, cancel := context.WithCancelCause(parent)
cancel(myError)
ctx.Err() // returns context.Canceled
context.Cause(ctx) // returns myError
```

func Join

added in go1.20

```
func Join(errs ...error) error
```

Join returns an error that wraps the given errors. Any nil error values are discarded. Join returns nil if errs contains no non-nil values. The error formats as the concatenation of the strings obtained by calling the Error method of each element of errs, with a newline between each string.

▼ Example

```
package main

import (
    "errors"
    "fmt"
)

func main() {
    err1 := errors.New("err1")
    err2 := errors.New("err2")
    err := errors.Join(err1, err2)
    fmt.Println(err)
    if errors.Is(err, err1) {
        fmt.Println("err is err1")
    }
    if errors.Is(err, err2) {
        fmt.Println("err is err2")
    }
}
```

errors.Join

```
Output:

err1
err2
err is err1
err is err2
```

New Go 1.20 os/exec.Cmd fields

- `Cancel func() error`
- `WaitDelay time.Duration`

Useful for managing graceful termination of processes initiated with CommandContext.

Proposal: https://go.dev/issue/50436

Proposals

Some interesting proposals updates

- Extend forwards compatibility for Go https://go.dev/issue/57001
- Extend backwards compatibility for Go https://go.dev/issue/56986
- CL for context.WithoutCancel https://go.dev/issue/40221 (accepted): https://go.dev/cl/459016
- log/slog https://go.dev/issue/56345 (active)
- spec: define initialization order more precisely https://go.dev/issue/57411 (accepted)
- CL for testing: optionally include full (or relative) path name https://go.dev/issue/37708 (accepted): https://go.dev/cl/463837 (+2)

Proposal accepted: context.WithoutCancel

Given a parent Context, return a new child Context, with the same values of the parent, but that is not canceled when the parent is canceled.

- Multiple implementations exists in the ecosystem
- It's great… But please don't abuse context to pass values everywhere!

https://github.com/golang/go/issues/40221

Proposal: Telemetry in the Go toolchain

- Transparent Telemetry for Open-Source Projects https://research.swtch.com/telemetry-intro
- Transparent Telemetry (@rsc) https://research.swtch.com/telemetry
- Reference: https://github.com/golang/go/discussions/58409

Gorilla toolkit was discontinued in December

Not related: https://words.filippo.io/full-time-maintainer/

How to stay up to date with Go development

Tip of the month:

- Cup o' Go podcast https://cupogo.dev/

More tips:

- Gophers on Slack https://invite.slack.gobridge.org/

You can do the next “What’s new in Go”!