Go 1.23 Release Notes

Table of Contents

Introduction to Go 1.23

The latest Go release, version 1.23, arrives six months after Go 1.22. Most of its changes are in the implementation of the toolchain, runtime, and libraries. As always, the release maintains the Go 1 promise of compatibility. We expect almost all Go programs to continue to compile and run as before.

Changes to the language

The “range” clause in a “for-range” loop now accepts iterator functions of the following types

```
func(func() bool)
func(func(K) bool)
func(func(K, V) bool)
```

as range expressions. Calls of the iterator argument function produce the iteration values for the “for-range” loop. For details see the `iter` package documentation, the language spec, and the Range over Function Types blog post. For motivation see the 2022 “range-over-func” discussion.

Go 1.23 includes preview support for generic type aliases. Building the toolchain with `GOEXPERIMENT=aliastypeparams` enables this feature within a package. (Using generic alias types across package boundaries is not yet supported.)

Tools

Telemetry

Starting in Go 1.23, the Go toolchain can collect usage and breakage statistics that help the Go team understand how the Go toolchain is used and how well it is working. We refer to these statistics as Go telemetry.

Go telemetry is an *opt-in system*, controlled by the `go telemetry` command. By default, the toolchain programs collect statistics in counter files that can be inspected locally but are otherwise unused (`go telemetry local`).

To help us keep Go working well and understand Go usage, please consider opting in to Go telemetry by running `go telemetry on`. In that mode, anonymous counter reports are uploaded to telemetry.go.dev weekly, where they are aggregated into graphs and also made available for download by any Go contributors or users wanting to analyze the data. See "Go Telemetry" for more details about the Go Telemetry system.

Go command

Setting the `GOROOT_FINAL` environment variable no longer has an effect (#62047). Distributions that install the `go` command to a location other than `$GOROOT/bin/go` should install a symlink instead of relocating or copying the `go` binary.

The new `go env -changed` flag causes the command to print only those settings whose effective value differs from the default value that would be obtained in an empty environment with no prior uses of the `-w` flag.

The new `go mod tidy -diff` flag causes the command not to modify the files but instead print the necessary changes as a unified diff. It exits with a non-zero code if updates are needed.

The `go list -m -json` command now includes new `Sum` and `GoModSum` fields. This is similar to the existing behavior of the `go mod download -json` command.

The new `godebug` directive in `go.mod` and `go.work` declares a GODEBUG setting to apply for the work module or workspace in use.

Vet

The `go vet` subcommand now includes the stdversion analyzer, which flags references to symbols that are too new for the version of Go in effect in the referring file. (The effective version is determined by the `go` directive in the file's enclosing `go.mod` file, and by any `//go:build` constraints in the file.)

For example, it will report a diagnostic for a reference to the `reflect.TypeFor` function (introduced in go1.22) from a file in a module whose go.mod file specifies `go 1.21`.

Cgo

`cmd/cgo` supports the new `-ldflags` flag for passing flags to the C linker. The `go` command uses it automatically, avoiding "argument list too long" errors with a very large `CGO_LDFLAGS`.

Trace

The `trace` tool now better tolerates partially broken traces by attempting to recover what trace data it can. This functionality is particularly helpful when viewing a trace that was collected during a program crash, since the trace data leading up to the crash will now be recoverable under most circumstances.

Runtime

The traceback printed by the runtime after an unhandled panic or other fatal error now indents the second and subsequent lines of the error message (for example, the argument to panic) by a single tab, so that it can be unambiguously distinguished from the stack trace of the first goroutine. See #64590 for discussion.

Compiler

The build time overhead to building with Profile Guided Optimization has been reduced significantly. Previously, large builds could see 100%+ build time increase from enabling PGO. In Go 1.23, overhead should be in the single digit percentages.

The compiler in Go 1.23 can now overlap the stack frame slots of local variables accessed in disjoint regions of a function, which reduces stack usage for Go applications.

For 386 and amd64, the compiler will use information from PGO to align certain hot blocks in loops. This improves performance an additional 1-1.5% at a cost of an additional 0.1% text and binary size. This is currently only implemented on 386 and amd64 because it has not shown an improvement on other platforms. Hot block alignment can be disabled with `-gcflags=[<packages>=]-d=alignhot=0`.

Linker

The linker now disallows using a `//go:linkname` directive to refer to internal symbols in the standard library (including the runtime) that are not marked with `//go:linkname` on their definitions. Similarly, the linker disallows references to such symbols from assembly code. For backward compatibility, existing usages of `//go:linkname` found in a large open-source code corpus remain supported. Any new references to standard library internal symbols will be disallowed.

A linker command line flag `-checklinkname=0` can be used to disable this check, for debugging and experimenting purposes.

When building a dynamically linked ELF binary (including PIE binary), the new `-bindnow` flag enables immediate function binding.

Standard library

Timer changes

Go 1.23 makes two significant changes to the implementation of `time.Timer` and `time.Ticker`.

First, `Timer`s and `Ticker`s that are no longer referred to by the program become eligible for garbage collection immediately, even if their `Stop` methods have not been called. Earlier versions of Go did not collect unstopped `Timer`s until after they had fired and never collected unstopped `Ticker`s.

Second, the timer channel associated with a `Timer` or `Ticker` is now unbuffered, with capacity 0. The main effect of this change is that Go now guarantees that for any call to a `Reset` or `Stop` method, no stale values prepared before that call will be sent or received after the call. Earlier versions of Go used channels with a one-element buffer, making it difficult to use `Reset` and `Stop` correctly. A visible effect of this change is that `len` and `cap` of timer channels now returns 0 instead of 1, which may affect programs that poll the length to decide whether a receive on the timer channel will succeed. Such code should use a non-blocking receive instead.

These new behaviors are only enabled when the main Go program is in a module with a `go.mod` `go` line using Go 1.23.0 or later. When Go 1.23 builds older programs, the old behaviors remain in effect. The new GODEBUG setting `asynctimerchan=1` can be used to revert back to asynchronous channel behaviors even when a program names Go 1.23.0 or later in its `go.mod` file.

New unique package

The new `unique` package provides facilities for canonicalizing values (like "interning" or "hash-consing").

Any value of comparable type may be canonicalized with the new `Make[T]` function, which produces a reference to a canonical copy of the value in the form of a `Handle[T]`. Two `Handle[T]` are equal if and only if the values used to produce the handles are equal, allowing programs to deduplicate values and reduce their memory footprint. Comparing two `Handle[T]` values is efficient, reducing down to a simple pointer comparison.

Iterators

The new `iter` package provides the basic definitions for working with user-defined iterators.

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

- All returns an iterator over slice indexes and values.
- Values returns an iterator over slice elements.
- Backward returns an iterator that loops over a slice backward.
- Collect collects values from an iterator into a new slice.
- AppendSeq appends values from an iterator to an existing slice.
- Sorted collects values from an iterator into a new slice, and then sorts the slice.
- SortedFunc is like `Sorted` but with a comparison function.
- SortedStableFunc is like `SortFunc` but uses a stable sort algorithm.
- Chunk returns an iterator over consecutive sub-slices of up to n elements of a slice.

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

- All returns an iterator over key-value pairs from a map.
- Keys returns an iterator over keys in a map.
- Values returns an iterator over values in a map.
- Insert adds the key-value pairs from an iterator to an existing map.
- Collect collects key-value pairs from an iterator into a new map and returns it.

New structs package

The new `structs` package provides types for struct fields that modify properties of the containing struct type such as memory layout.

In this release, the only such type is `HostLayout` which indicates that a structure with a field of that type has a layout that conforms to host platform expectations. HostLayout should be used in types that are passed to, returned from, or accessed via a pointer passed to/from host APIs. Without this marker, struct layout order is not guaranteed by the language spec, though as of Go 1.23 the host and language layouts happen to match.

Minor changes to the library

`archive/tar`

If the argument to `FileInfoHeader` implements the new `FileInfoNames` interface, then the interface methods will be used to set the Uname/Gname of the file header. This allows applications to override the system-dependent Uname/Gname lookup.

`crypto/tls`

The TLS client now supports the Encrypted Client Hello draft specification. This feature can be enabled by setting the `Config.EncryptedClientHelloConfigList` field to an encoded ECHConfigList for the host that is being connected to.

The `QUICConn` type used by QUIC implementations includes new events reporting on the state of session resumption, and provides a way for the QUIC layer to add data to session tickets and session cache entries.

3DES cipher suites were removed from the default list used when `Config.CipherSuites` is nil. The default can be reverted by adding `tls3des=1` to the GODEBUG environment variable.

The experimental post-quantum key exchange mechanism X25519Kyber768Draft00 is now enabled by default when `Config.CurvePreferences` is nil. The default can be reverted by adding `tlskyber=0` to the GODEBUG environment variable. This can be useful when dealing with buggy TLS servers that do not handle large records correctly, causing a timeout during the handshake (see TLS post-quantum TL;DR fail).

Go 1.23 changed the behavior of `X509KeyPair` and `LoadX509KeyPair` to populate the `Certificate.Leaf` field of the returned `Certificate`. The new `x509keypairleaf` GODEBUG setting is added for this behavior.

`crypto/x509`

`CreateCertificateRequest` now correctly supports RSA-PSS signature algorithms.

`CreateCertificateRequest` and `CreateRevocationList` now verify the generated signature using the signer's public key. If the signature is invalid, an error is returned. This has been the behavior of `CreateCertificate` since Go 1.16.

The `x509sha1` GODEBUG setting will be removed in the next Go major release (Go 1.24). This will mean that `crypto/x509` will no longer support verifying signatures on certificates that use SHA-1 based signature algorithms.

The new `ParseOID` function parses a dot-encoded ASN.1 Object Identifier string. The `OID` type now implements the `encoding.BinaryMarshaler`, `encoding.BinaryUnmarshaler`, `encoding.TextMarshaler`, `encoding.TextUnmarshaler` interfaces.

`database/sql`

Errors returned by `driver.Valuer` implementations are now wrapped for improved error handling during operations like `DB.Query`, `DB.Exec`, and `DB.QueryRow`.

`debug/elf`

The `debug/elf` package now defines `PT_OPENBSD_NOBTCFI`. This `ProgType` is used to disable Branch Tracking Control Flow Integrity (BTCFI) enforcement on OpenBSD binaries.

Now defines the symbol type constants `STT_RELC`, `STT_SRELC`, and `STT_GNU_IFUNC`.

`encoding/binary`

The new `Encode` and `Decode` functions are byte slice equivalents to `Read` and `Write`. `Append` allows marshaling multiple data into the same byte slice.

`go/ast`

The new `Preorder` function returns a convenient iterator over all the nodes of a syntax tree.

`go/types`

The `Func` type, which represents a function or method symbol, now has a `Func.Signature` method that returns the function's type, which is always a `Signature`.

The `Alias` type now has an `Rhs` method that returns the type on the right-hand side of its declaration: given `type` A = B, the `Rhs` of A is B. (#66559)

The methods `Alias.Origin`, `Alias.SetTypeParams`, `Alias.TypeParams`, and `Alias.TypeArgs` have been added. They are needed for generic alias types.

By default, go/types now produces `Alias` type nodes for type aliases. This behavior can be controlled by the `GODEBUG` `gotypesalias` flag. Its default has changed from 0 in Go 1.22 to 1 in Go 1.23.

`math/rand/v2`

The `Uint` function and `Rand.Uint` method have been added. They were inadvertently left out of Go 1.22.

The new `ChaCha8.Read` method implements the `io.Reader` interface.

`net`

The new type `KeepAliveConfig` permits fine-tuning the keep-alive options for TCP connections, via a new `TCPConn.SetKeepAliveConfig` method and new KeepAliveConfig fields for `Dialer` and `ListenConfig`.

The `DNSError` type now wraps errors caused by timeouts or cancellation. For example, `errors.Is(someDNSErr, context.DeadlineExceedeed)` will now report whether a DNS error was caused by a timeout.

The new `GODEBUG` setting `netedns0=0` disables sending EDNS0 additional headers on DNS requests, as they reportedly break the DNS server on some modems.

`net/http`

`Cookie` now preserves double quotes surrounding a cookie value. The new `Cookie.Quoted` field indicates whether the `Cookie.Value` was originally quoted.

The new `Request.CookiesNamed` method retrieves all cookies that match the given name.

The new `Cookie.Partitioned` field identifies cookies with the Partitioned attribute.

The patterns used by `ServeMux` now allow one or more spaces or tabs after the method name. Previously, only a single space was permitted.

The new `ParseCookie` function parses a Cookie header value and returns all the cookies which were set in it. Since the same cookie name can appear multiple times the returned Values can contain more than one value for a given key.

The new `ParseSetCookie` function parses a Set-Cookie header value and returns a cookie. It returns an error on syntax error.

`ServeContent`, `ServeFile`, and `ServeFileFS` now remove the `Cache-Control`, `Content-Encoding`, `Etag`, and `Last-Modified` headers when serving an error. These headers usually apply to the non-error content, but not to the text of errors.

Middleware which wraps a `ResponseWriter` and applies on-the-fly encoding, such as `Content-Encoding: gzip`, will not function after this change. The previous behavior of `ServeContent`, `ServeFile`, and `ServeFileFS` may be restored by setting `GODEBUG=httpservecontentkeepheaders=1`.

Note that middleware which changes the size of the served content (such as by compressing it) already does not function properly when `ServeContent` handles a Range request. On-the-fly compression should use the `Transfer-Encoding` header instead of `Content-Encoding`.

For inbound requests, the new `Request.Pattern` field contains the `ServeMux` pattern (if any) that matched the request. This field is not set when `GODEBUG=httpmuxgo121=1` is set.

net/http/httptest

The new `NewRequestWithContext` method creates an incoming request with a `context.Context`.

net/netip

In Go 1.22 and earlier, using `reflect.DeepEqual` to compare an `Addr` holding an IPv4 address to one holding the IPv4-mapped IPv6 form of that address incorrectly returned true, even though the `Addr` values were different when comparing with == or `Addr.Compare`. This bug is now fixed and all three approaches now report the same result.

os

The `Stat` function now sets the `ModeSocket` bit for files that are Unix sockets on Windows. These files are identified by having a reparse tag set to `IO_REPARSE_TAG_AF_UNIX`.

On Windows, the mode bits reported by `Lstat` and `Stat` for reparse points changed. Mount points no longer have `ModeSymlink` set, and reparse points that are not symlinks, Unix sockets, or dedup files now always have `ModeIrregular` set. This behavior is controlled by the `winsymlink` setting. For Go 1.23, it defaults to `winsymlink=1`. Previous versions default to `winsymlink=0`.

The `CopyFS` function copies an `io/fs.FS` into the local filesystem.

On Windows, `Readlink` no longer tries to normalize volumes to drive letters, which was not always even possible. This behavior is controlled by the `winreadlinkvolume` setting. For Go 1.23, it defaults to `winreadlinkvolume=1`. Previous versions default to `winreadlinkvolume=0`.

On Linux with pidfd support (generally Linux v5.4+), `Process`-related functions and methods use pidfd (rather than PID) internally, eliminating potential mistargeting when a PID is reused by the OS. Pidfd support is fully transparent to a user, except for additional process file descriptors that a process may have.

`path/filepath`

The new `Localize` function safely converts a slash-separated path into an operating system path.

On Windows, `EvalSymlinks` no longer evaluates mount points, which was a source of many inconsistencies and bugs. This behavior is controlled by the `winsymlink` setting. For Go 1.23, it defaults to `winsymlink=1`. Previous versions default to `winsymlink=0`.

On Windows, `EvalSymlinks` no longer tries to normalize volumes to drive letters, which was not always even possible. This behavior is controlled by the `winreadlinkvolume` setting. For Go 1.23, it defaults to `winreadlinkvolume=1`. Previous versions default to `winreadlinkvolume=0`.

`reflect`

The new methods synonymous with the methods of the same name in `Value` are added to `Type`:

1. `Type.OverflowComplex`
2. `Type.OverflowFloat`
3. `Type.OverflowInt`
4. `Type.OverflowUint`

The new `SliceAt` function is analogous to `NewAt`, but for slices.

The `Value.Pointer` and `Value.UnsafePointer` methods now support values of kind `String`.

The new methods `Value.Seq` and `Value.Seq2` return sequences that iterate over the value as though it were used in a for/range loop. The new methods `Type.CanSeq` and `Type.CanSeq2` report whether calling `Value.Seq` and `Value.Seq2`, respectively, will succeed without panicking.

runtime/debug

The `SetCrashOutput` function allows the user to specify an alternate file to which the runtime should write its fatal crash report. It may be used to construct an automated reporting mechanism for all unexpected crashes, not just those in goroutines that explicitly use `recover`.

runtime/pprof

The maximum stack depth for `alloc`, `mutex`, `block`, `threadcreate` and `goroutine` profiles has been raised from 32 to 128 frames.

runtime/trace

The runtime now explicitly flushes trace data when a program crashes due to an uncaught panic. This means that more complete trace data will be available in a trace if the program crashes while tracing is active.

slices

The `Repeat` function returns a new slice that repeats the provided slice the given number of times.

sync

The `Map.Clear` method deletes all the entries, resulting in an empty `Map`. It is analogous to `clear`.

sync/atomic

The new `And` and `Or` operators apply a bitwise `AND` or `OR` to the given input, returning the old value.

syscall

The syscall package now defines `WSAENOPROTOOPT` on Windows.

The `GetsockoptInt` function is now supported on Windows.

`testing/fstest`

`TestFS` now returns a structured error that can be unwrapped (via method `Unwrap() []error`). This allows inspecting errors using `errors.Is` or `errors.As`.

`text/template`

Templates now support the new “else with” action, which reduces template complexity in some use cases.

`time`

`Parse` and `ParseInLocation` now return an error if the time zone offset is out of range.

On Windows, `Timer`, `Ticker`, and functions that put the goroutine to sleep, such as `Sleep`, got their time resolution improved to 0.5ms instead of 15.6ms.

`unicode/utf16`

The `RuneLen` function returns the number of 16-bit words in the UTF-16 encoding of the rune. It returns -1 if the rune is not a valid value to encode in UTF-16.

Ports

Darwin

As announced in the Go 1.22 release notes, Go 1.23 requires macOS 11 Big Sur or later; support for previous versions has been discontinued.

Linux

Go 1.23 is the last release that requires Linux kernel version 2.6.32 or later. Go 1.24 will require Linux kernel version 3.2 or later.

OpenBSD

Go 1.23 adds experimental support for OpenBSD on 64-bit RISC-V (`GOOS=openbsd`, `GOARCH=riscv64`).

ARM64

Go 1.23 introduces a new `GOARM64` environment variable, which specifies the minimum target version of the ARM64 architecture at compile time. Allowed values are `v8.{0-9}` and `v9.{0-5}`. This may be followed by an option specifying extensions implemented by target hardware. Valid options are `,lse` and `,crypto`.

The `GOARM64` environment variable defaults to `v8.0`.

RISC-V

Go 1.23 introduces a new `GORISCV64` environment variable, which selects the RISC-V user-mode application profile for which to compile. Allowed values are `rva20u64` and `rva22u64`.

The `GORISCV64` environment variable defaults to `rva20u64`.

Wasm

The `go_wasip1_wasm_exec` script in `GOROOT/misc/wasm` has dropped support for versions of `wasmtime` < 14.0.0.