Go 1.20 Release Notes

Table of Contents

Introduction to Go 1.20

The latest Go release, version 1.20, arrives six months after Go 1.19. 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

Go 1.20 includes four changes to the language.

Go 1.17 added conversions from slice to an array pointer. Go 1.20 extends this to allow conversions from a slice to an array: given a slice `x`, `[4]byte(x)` can now be written instead of `*(*[4]byte)(x)`.

The `unsafe` package defines three new functions `SliceData`, `String`, and `StringData`. Along with Go 1.17's `Slice`, these functions now provide the complete ability to construct and deconstruct slice and string values, without depending on their exact representation.

The specification now defines that struct values are compared one field at a time, considering fields in the order they appear in the struct type definition, and stopping at the first mismatch. The specification could previously have been read as if all fields needed to be compared beyond the first mismatch. Similarly, the specification now defines that array values are compared one element at a time, in increasing index order. In both cases, the difference affects whether certain comparisons must panic. Existing programs are unchanged: the new spec wording describes what the implementations have always done.

Comparable types (such as ordinary interfaces) may now satisfy `comparable` constraints, even if the type arguments are not strictly comparable (comparison may panic at runtime). This makes it possible to instantiate a type parameter constrained by `comparable` (e.g., a type parameter for a user-defined generic map key) with a non-strictly comparable type argument such as an interface type, or a composite type containing an interface type.

Ports

Windows

Go 1.20 is the last release that will run on any release of Windows 7, 8, Server 2008 and Server 2012. Go 1.21 will require at least Windows 10 or Server 2016.

Darwin and iOS

Go 1.20 is the last release that will run on macOS 10.13 High Sierra or 10.14 Mojave. Go 1.21 will require macOS 10.15 Catalina or later.

FreeBSD/RISC-V

Go 1.20 adds experimental support for FreeBSD on RISC-V (`GOOS=freebsd`, `GOARCH=riscv64`).

Tools

Go command

The directory `$GOROOT/pkg` no longer stores pre-compiled package archives for the standard library: `go install` no longer writes them, the `go` build no longer checks for them, and the Go distribution no longer ships them. Instead, packages in the standard library are built as needed and cached in the build cache, just like packages outside `GOROOT`. This change reduces the size of the Go distribution and also avoids C toolchain skew for packages that use cgo.

The implementation of `go test -json` has been improved to make it more robust. Programs that run `go test -json` do not need any updates. Programs that invoke `go tool test2json` directly should now run the test binary with `-v=test2json` (for example, `go test -v=test2json` or `./pkg.test -test.v=test2json`) instead of plain `-v`.

A related change to `go test -json` is the addition of an event with `Action` set to `start` at the beginning of each test program's execution. When running multiple tests using the `go` command, these start events are guaranteed to be emitted in the same order as the packages named on the command line.

The `go` command now defines architecture feature build tags, such as `amd64.v2`, to allow selecting a package implementation file based on the presence or absence of a particular

architecture feature. See `go help buildconstraint` for details.

The `go` subcommands now accept `-C <dir>` to change directory to <dir> before performing the command, which may be useful for scripts that need to execute commands in multiple different modules.

The `go build` and `go test` commands no longer accept the `-i` flag, which has been deprecated since Go 1.16.

The `go generate` command now accepts `-skip <pattern>` to skip `//go:generate` directives matching `<pattern>`.

The `go test` command now accepts `-skip <pattern>` to skip tests, subtests, or examples matching `<pattern>`.

When the main module is located within `GOPATH/src`, `go install` no longer installs libraries for non-`main` packages to `GOPATH/pkg`, and `go list` no longer reports a `Target` field for such packages. (In module mode, compiled packages are stored in the build cache only, but a bug had caused the `GOPATH` install targets to unexpectedly remain in effect.)

The `go build`, `go install`, and other build-related commands now support a `-pgo` flag that enables profile-guided optimization, which is described in more detail in the Compiler section below. The `-pgo` flag specifies the file path of the profile. Specifying `-pgo=auto` causes the `go` command to search for a file named `default.pgo` in the main package's directory and use it if present. This mode currently requires a single main package to be specified on the command line, but we plan to lift this restriction in a future release. Specifying `-pgo=off` turns off profile-guided optimization.

The `go build`, `go install`, and other build-related commands now support a `-cover` flag that builds the specified target with code coverage instrumentation. This is described in more detail in the Cover section below.

`go version`

The `go version -m` command now supports reading more types of Go binaries, most notably, Windows DLLs built with `go build -buildmode=c-shared` and Linux binaries without execute permission.

Cgo

The `go` command now disables `cgo` by default on systems without a C toolchain. More specifically, when the `CGO_ENABLED` environment variable is unset, the `CC` environment variable is unset, and the default C compiler (typically `clang` or `gcc`) is not found in the path, `CGO_ENABLED` defaults to `0`. As always, you can override the default by setting `CGO_ENABLED` explicitly.

The most important effect of the default change is that when Go is installed on a system without a C compiler, it will now use pure Go builds for packages in the standard library that use cgo, instead of using pre-distributed package archives (which have been removed, as noted above) or attempting to use cgo and failing. This makes Go work better in some minimal container environments as well as on macOS, where pre-distributed package archives have not been used for cgo-based packages since Go 1.16.

The packages in the standard library that use cgo are `net`, `os/user`, and `plugin`. On macOS, the `net` and `os/user` packages have been rewritten not to use cgo: the same code is now used for cgo and non-cgo builds as well as cross-compiled builds. On Windows, the `net` and `os/user` packages have never used cgo. On other systems, builds with cgo disabled will use a pure Go version of these packages.

A consequence is that, on macOS, if Go code that uses the `net` package is built with `-buildmode=c-archive`, linking the resulting archive into a C program requires passing `-lresolv` when linking the C code.

On macOS, the race detector has been rewritten not to use cgo: race-detector-enabled programs can be built and run without Xcode. On Linux and other Unix systems, and on Windows, a host C toolchain is required to use the race detector.

Cover

Go 1.20 supports collecting code coverage profiles for programs (applications and integration tests), as opposed to just unit tests.

To collect coverage data for a program, build it with `go build`'s `-cover` flag, then run the resulting binary with the environment variable `GOCOVERDIR` set to an output directory for coverage profiles. See the 'coverage for integration tests' landing page for more on how to get started. For details on the design and implementation, see the proposal.

Vet

Improved detection of loop variable capture by nested functions

The `vet` tool now reports references to loop variables following a call to `T.Parallel()` within subtest function bodies. Such references may observe the value of the variable from a different iteration (typically causing test cases to be skipped) or an invalid state due to unsynchronized concurrent access.

The tool also detects reference mistakes in more places. Previously it would only consider the last statement of the loop body, but now it recursively inspects the last statements within if, switch, and select statements.

New diagnostic for incorrect time formats

The vet tool now reports use of the time format 2006-02-01 (yyyy-dd-mm) with `Time.Format` and `time.Parse`. This format does not appear in common date standards, but is frequently used by mistake when attempting to use the ISO 8601 date format (yyyy-mm-dd).

Runtime

Some of the garbage collector's internal data structures were reorganized to be both more space and CPU efficient. This change reduces memory overheads and improves overall CPU performance by up to 2%.

The garbage collector behaves less erratically with respect to goroutine assists in some circumstances.

Go 1.20 adds a new `runtime/coverage` package containing APIs for writing coverage profile data at runtime from long-running and/or server programs that do not terminate via `os.Exit()`.

Compiler

Go 1.20 adds preview support for profile-guided optimization (PGO). PGO enables the toolchain to perform application- and workload-specific optimizations based on run-time profile information. Currently, the compiler supports pprof CPU profiles, which can be collected through usual means, such as the `runtime/pprof` or `net/http/pprof` packages. To enable PGO, pass the path of a pprof profile file via the `–pgo` flag to `go build`, as mentioned above. Go 1.20 uses PGO to more aggressively inline functions at hot call sites. Benchmarks for a representative set of Go programs show enabling profile-guided inlining optimization improves performance about 3–4%. See the PGO user guide for detailed documentation. We plan to add more profile-guided optimizations in future releases. Note that profile-guided optimization is a preview, so please use it with appropriate caution.

The Go 1.20 compiler upgraded its front-end to use a new way of handling the compiler's internal data, which fixes several generic-types issues and enables type declarations within generic functions and methods.

The compiler now rejects anonymous interface cycles with a compiler error by default. These arise from tricky uses of embedded interfaces and have always had subtle correctness issues, yet we have no evidence that they're actually used in practice. Assuming no reports from users adversely affected by this change, we plan to update the language specification for Go 1.22 to formally disallow them so tools authors can stop supporting them too.

Go 1.18 and 1.19 saw regressions in build speed, largely due to the addition of support for generics and follow-on work. Go 1.20 improves build speeds by up to 10%, bringing it back in line with Go 1.17. Relative to Go 1.19, generated code performance is also generally slightly improved.

Linker

On Linux, the linker now selects the dynamic interpreter for `glibc` or `musl` at link time.

On Windows, the Go linker now supports modern LLVM-based C toolchains.

Go 1.20 uses `go:` and `type:` prefixes for compiler-generated symbols rather than `go.` and `type.`. This avoids confusion for user packages whose name starts with `go.`. The `debug/gosym` package understands this new naming convention for binaries built with Go 1.20 and newer.

Bootstrap

When building a Go release from source and `GOROOT_BOOTSTRAP` is not set, previous versions of Go looked for a Go 1.4 or later bootstrap toolchain in the directory `$HOME/go1.4` (`%HOMEDRIVE%%HOMEPATH%\go1.4` on Windows). Go 1.18 and Go 1.19 looked first for `$HOME/go1.17` or `$HOME/sdk/go1.17` before falling back to `$HOME/go1.4`, in anticipation of requiring Go 1.17 for use when bootstrapping Go 1.20. Go 1.20 does require a Go 1.17 release for bootstrapping, but we realized that we should adopt the latest point release of the bootstrap toolchain, so it requires Go 1.17.13. Go 1.20 looks for `$HOME/go1.17.13` or `$HOME/sdk/go1.17.13` before falling back to `$HOME/go1.4` (to support systems that hard-coded the path $HOME/go1.4 but have installed a newer Go toolchain there). In the future, we plan to move the bootstrap toolchain forward approximately once a year, and in particular we expect that Go 1.22 will require the final point release of Go 1.20 for bootstrap.

Standard library

New crypto/ecdh package

Go 1.20 adds a new `crypto/ecdh` package to provide explicit support for Elliptic Curve Diffie-Hellman key exchanges over NIST curves and Curve25519.

Programs should use `crypto/ecdh` instead of the lower-level functionality in `crypto/elliptic` for ECDH, and third-party modules for more advanced use cases.

Wrapping multiple errors

Go 1.20 expands support for error wrapping to permit an error to wrap multiple other errors.

An error `e` can wrap more than one error by providing an `Unwrap` method that returns a `[]error`.

The `errors.Is` and `errors.As` functions have been updated to inspect multiply wrapped errors.

The `fmt.Errorf` function now supports multiple occurrences of the `%w` format verb, which will cause it to return an error that wraps all of those error operands.

The new function `errors.Join` returns an error wrapping a list of errors.

HTTP ResponseController

The new `"net/http".ResponseController` type provides access to extended per-request functionality not handled by the `"net/http".ResponseWriter` interface.

Previously, we have added new per-request functionality by defining optional interfaces which a `ResponseWriter` can implement, such as `Flusher`. These interfaces are not discoverable and clumsy to use.

The `ResponseController` type provides a clearer, more discoverable way to add per-handler controls. Two such controls also added in Go 1.20 are `SetReadDeadline` and `SetWriteDeadline`, which allow setting per-request read and write deadlines. For example:

```
func RequestHandler(w ResponseWriter, r *Request) {
  rc := http.NewResponseController(w)
  rc.SetWriteDeadline(time.Time{}) // disable Server.WriteTimeout when sending a la
  io.Copy(w, bigData)
}
```

New ReverseProxy Rewrite hook

The `httputil.ReverseProxy` forwarding proxy includes a new `Rewrite` hook function, superseding the previous `Director` hook.

The `Rewrite` hook accepts a `ProxyRequest` parameter, which includes both the inbound request received by the proxy and the outbound request that it will send. Unlike `Director` hooks, which only operate on the outbound request, this permits `Rewrite` hooks to avoid certain scenarios where a malicious inbound request may cause headers added by the hook to be removed before forwarding. See issue #50580.

The `ProxyRequest.SetURL` method routes the outbound request to a provided destination and supersedes the `NewSingleHostReverseProxy` function. Unlike `NewSingleHostReverseProxy`, `SetURL` also sets the `Host` header of the outbound request.

The `ProxyRequest.SetXForwarded` method sets the `X-Forwarded-For`, `X-Forwarded-Host`, and `X-Forwarded-Proto` headers of the outbound request. When using a `Rewrite`, these headers are not added by default.

An example of a `Rewrite` hook using these features is:

```
proxyHandler := &httputil.ReverseProxy{
  Rewrite: func(r *httputil.ProxyRequest) {
    r.SetURL(outboundURL) // Forward request to outboundURL.
    r.SetXForwarded()     // Set X-Forwarded-* headers.
    r.Out.Header.Set("X-Additional-Header", "header set by the proxy")
  },
}
```

`ReverseProxy` no longer adds a `User-Agent` header to forwarded requests when the incoming request does not have one.

Minor changes to the library

As always, there are various minor changes and updates to the library, made with the Go 1 promise of compatibility in mind. There are also various performance improvements, not enumerated here.

archive/tar

When the `GODEBUG=tarinsecurepath=0` environment variable is set, `Reader.Next` method will now return the error `ErrInsecurePath` for an entry with a file name that is an absolute path, refers to a location outside the current directory, contains invalid characters, or (on Windows) is a reserved name such as `NUL`. A future version of Go may disable insecure paths by default.

archive/zip

When the `GODEBUG=zipinsecurepath=0` environment variable is set, `NewReader` will now return the error `ErrInsecurePath` when opening an archive which contains any file name that is an absolute path, refers to a location outside the current directory, contains invalid characters, or (on Windows) is a reserved names such as `NUL`. A future version of Go may disable insecure paths by default.

Reading from a directory file that contains file data will now return an error. The zip specification does not permit directory files to contain file data, so this change only affects reading from invalid archives.

bytes

The new `CutPrefix` and `CutSuffix` functions are like `TrimPrefix` and `TrimSuffix` but also report whether the string was trimmed.

The new `Clone` function allocates a copy of a byte slice.

context

The new `WithCancelCause` function provides a way to cancel a context with a given error. That error can be retrieved by calling the new `Cause` function.

crypto/ecdsa

When using supported curves, all operations are now implemented in constant time. This led to an increase in CPU time between 5% and 30%, mostly affecting P-384 and P-521.

The new `PrivateKey.ECDH` method converts an `ecdsa.PrivateKey` to an `ecdh.PrivateKey`.

crypto/ed25519

The `PrivateKey.Sign` method and the `VerifyWithOptions` function now support signing pre-hashed messages with Ed25519ph, indicated by an `Options.HashFunc` that returns `crypto.SHA512`. They also now support Ed25519ctx and Ed25519ph with context, indicated by setting the new `Options.Context` field.

crypto/rsa

The new field `OAEPOptions.MGFHash` allows configuring the MGF1 hash separately for OAEP decryption.

crypto/rsa now uses a new, safer, constant-time backend. This causes a CPU runtime increase for decryption operations between approximately 15% (RSA-2048 on amd64) and 45% (RSA-4096 on arm64), and more on 32-bit architectures. Encryption operations are approximately 20x slower than before (but still 5-10x faster than decryption). Performance is expected to improve in future releases. Programs must not modify or manually generate the fields of `PrecomputedValues`.

crypto/subtle

The new function `XORBytes` XORs two byte slices together.

crypto/tls

Parsed certificates are now shared across all clients actively using that certificate. The memory savings can be significant in programs that make many concurrent connections to a server or collection of servers sharing any part of their certificate chains.

For a handshake failure due to a certificate verification failure, the TLS client and server now return an error of the new type `CertificateVerificationError`, which includes the presented certificates.

crypto/x509

`ParsePKCS8PrivateKey` and `MarshalPKCS8PrivateKey` now support keys of type `*crypto/ecdh.PrivateKey`. `ParsePKIXPublicKey` and `MarshalPKIXPublicKey` now support keys of type `*crypto/ecdh.PublicKey`. Parsing NIST curve keys still returns values of type `*ecdsa.PublicKey` and `*ecdsa.PrivateKey`. Use their new `ECDH` methods to convert to the `crypto/ecdh` types.

The new `SetFallbackRoots` function allows a program to define a set of fallback root certificates in case an operating system verifier or standard platform root bundle is unavailable at runtime. It will most commonly be used with a new package, golang.org/x/crypto/x509roots/fallback, which will provide an up to date root bundle.

debug/elf

Attempts to read from a `SHT_NOBITS` section using `Section.Data` or the reader returned by `Section.Open` now return an error.

Additional `R_LARCH_*` constants are defined for use with LoongArch systems.

Additional `R_PPC64_*` constants are defined for use with PPC64 ELFv2 relocations.

The constant value for `R_PPC64_SECTOFF_LO_DS` is corrected, from 61 to 62.

debug/gosym

Due to a change of Go's symbol naming conventions, tools that process Go binaries should use Go 1.20's `debug/gosym` package to transparently handle both old and new binaries.

debug/pe

Additional `IMAGE_FILE_MACHINE_RISCV*` constants are defined for use with RISC-V systems.

encoding/binary

The `ReadVarint` and `ReadUvarint` functions will now return `io.ErrUnexpectedEOF` after reading a partial value, rather than `io.EOF`.

encoding/xml

The new `Encoder.Close` method can be used to check for unclosed elements when finished encoding.

The decoder now rejects element and attribute names with more than one colon, such as `<a:b:c>`, as well as namespaces that resolve to an empty string, such as `xmlns:a=""`.

The decoder now rejects elements that use different namespace prefixes in the opening and closing tag, even if those prefixes both denote the same namespace.

errors

The new `Join` function returns an error wrapping a list of errors.

fmt

The `Errorf` function supports multiple occurrences of the `%w` format verb, returning an error that unwraps to the list of all arguments to `%w`.

The new `FormatString` function recovers the formatting directive corresponding to a `State`, which can be useful in `Formatter`. implementations.

go/ast

The new `RangeStmt.Range` field records the position of the `range` keyword in a range statement.

The new `File.FileStart` and `File.FileEnd` fields record the position of the start and end of the entire source file.

go/token

The new `FileSet.RemoveFile` method removes a file from a `FileSet`. Long-running programs can use this to release memory associated with files they no longer need.

go/types

The new `Satisfies` function reports whether a type satisfies a constraint. This change aligns with the new language semantics that distinguish satisfying a constraint from implementing an interface.

html/template

Go 1.20.3 and later disallow actions in ECMAScript 6 template literals. This behavior can be reverted by the `GODEBUG=jstmpllitinterp=1` setting.

io

The new `OffsetWriter` wraps an underlying `WriterAt` and provides `Seek`, `Write`, and `WriteAt` methods that adjust their effective file offset position by a fixed amount.

io/fs

The new error `SkipAll` terminates a `WalkDir` immediately but successfully.

math/big

The math/big package's wide scope and input-dependent timing make it ill-suited for implementing cryptography. The cryptography packages in the standard library no longer call non-trivial Int methods on attacker-controlled inputs. In the future, the determination of whether a bug in math/big is considered a security vulnerability will depend on its wider impact on the standard library.

math/rand

The math/rand package now automatically seeds the global random number generator (used by top-level functions like `Float64` and `Int`) with a random value, and the top-level `Seed` function has been deprecated. Programs that need a reproducible sequence of random numbers should prefer to allocate their own random source, using `rand.New(rand.NewSource(seed))`.

Programs that need the earlier consistent global seeding behavior can set `GODEBUG=randautoseed=0` in their environment.

The top-level `Read` function has been deprecated. In almost all cases, `crypto/rand.Read` is more appropriate.

mime

The `ParseMediaType` function now allows duplicate parameter names, so long as the values of the names are the same.

mime/multipart

Methods of the `Reader` type now wrap errors returned by the underlying `io.Reader`.

In Go 1.19.8 and later, this package sets limits the size of the MIME data it processes to protect against malicious inputs. `Reader.NextPart` and `Reader.NextRawPart` limit the number of headers in a part to 10000 and `Reader.ReadForm` limits the total number of headers in all `FileHeaders` to 10000. These limits may be adjusted with the `GODEBUG=multipartmaxheaders` setting. `Reader.ReadForm` further limits the number of parts in a form to 1000. This limit may be adjusted with the `GODEBUG=multipartmaxparts` setting.

net

The `LookupCNAME` function now consistently returns the contents of a `CNAME` record when one exists. Previously on Unix systems and when using the pure Go resolver, `LookupCNAME` would return an error if a `CNAME` record referred to a name that with no `A`, `AAAA`, or `CNAME` record. This change modifies `LookupCNAME` to match the previous behavior on Windows, allowing `LookupCNAME` to succeed whenever a `CNAME` exists.

`Interface.Flags` now includes the new flag `FlagRunning`, indicating an operationally active interface. An interface which is administratively configured but not active (for example, because

the network cable is not connected) will have `FlagUp` set but not `FlagRunning`.

The new `Dialer.ControlContext` field contains a callback function similar to the existing `Dialer.Control` hook, that additionally accepts the dial context as a parameter. `Control` is ignored when `ControlContext` is not nil.

The Go DNS resolver recognizes the `trust-ad` resolver option. When `options trust-ad` is set in `resolv.conf`, the Go resolver will set the AD bit in DNS queries. The resolver does not make use of the AD bit in responses.

DNS resolution will detect changes to `/etc/nsswitch.conf` and reload the file when it changes. Checks are made at most once every five seconds, matching the previous handling of `/etc/hosts` and `/etc/resolv.conf`.

net/http

The `ResponseWriter.WriteHeader` function now supports sending `1xx` status codes.

The new `Server.DisableGeneralOptionsHandler` configuration setting allows disabling the default `OPTIONS *` handler.

The new `Transport.OnProxyConnectResponse` hook is called when a `Transport` receives an HTTP response from a proxy for a `CONNECT` request.

The HTTP server now accepts HEAD requests containing a body, rather than rejecting them as invalid.

HTTP/2 stream errors returned by `net/http` functions may be converted to a `golang.org/x/net/http2.StreamError` using `errors.As`.

Leading and trailing spaces are trimmed from cookie names, rather than being rejected as invalid. For example, a cookie setting of “name =value” is now accepted as setting the cookie “name”.

A `Cookie` with an empty Expires field is now considered valid. `Cookie.Valid` only checks Expires when it is set.

net/netip

The new `IPv6LinkLocalAllRouters` and `IPv6Loopback` functions are the `net/netip` equivalents of `net.IPv6loopback` and `net.IPv6linklocalallrouters`.

os

On Windows, the name `NUL` is no longer treated as a special case in `Mkdir` and `Stat`.

On Windows, `File.Stat` now uses the file handle to retrieve attributes when the file is a directory. Previously it would use the path passed to `Open`, which may no longer be the file represented by the file handle if the file has been moved or replaced. This change modifies `Open` to open directories without the `FILE_SHARE_DELETE` access, which match the behavior of regular files.

On Windows, `File.Seek` now supports seeking to the beginning of a directory.

os/exec

The new `Cmd` fields `Cancel` and `WaitDelay` specify the behavior of the `Cmd` when its associated `Context` is canceled or its process exits with I/O pipes still held open by a child process.

path/filepath

The new error `SkipAll` terminates a `Walk` immediately but successfully.

The new `IsLocal` function reports whether a path is lexically local to a directory. For example, if `IsLocal(p)` is `true`, then `Open(p)` will refer to a file that is lexically within the subtree rooted at the current directory.

reflect

The new `Value.Comparable` and `Value.Equal` methods can be used to compare two `Values` for equality. `Comparable` reports whether `Equal` is a valid operation for a given `Value` receiver.

The new `Value.Grow` method extends a slice to guarantee space for another `n` elements.

The new `Value.SetZero` method sets a value to be the zero value for its type.

Go 1.18 introduced `Value.SetIterKey` and `Value.SetIterValue` methods. These are optimizations: `v.SetIterKey(it)` is meant to be equivalent to `v.Set(it.Key())`. The implementations incorrectly omitted a check for use of unexported fields that was present in the unoptimized forms. Go 1.20 corrects these methods to include the unexported field check.

regexp

Go 1.19.2 and Go 1.18.7 included a security fix to the regular expression parser, making it reject very large expressions that would consume too much memory. Because Go patch releases do not introduce new API, the parser returned `syntax.ErrInternalError` in this case. Go 1.20 adds a more specific error, `syntax.ErrLarge`, which the parser now returns instead.

runtime/cgo

Go 1.20 adds new `Incomplete` marker type. Code generated by cgo will use `cgo.Incomplete` to mark an incomplete C type.

runtime/metrics

Go 1.20 adds new supported metrics, including the current `GOMAXPROCS` setting (`/sched/gomaxprocs:threads`), the number of cgo calls executed (`/cgo/go-to-c-calls:calls`), total mutex block time (`/sync/mutex/wait/total:seconds`), and various measures of time spent in garbage collection.

Time-based histogram metrics are now less precise, but take up much less memory.

runtime/pprof

Mutex profile samples are now pre-scaled, fixing an issue where old mutex profile samples would be scaled incorrectly if the sampling rate changed during execution.

Profiles collected on Windows now include memory mapping information that fixes symbolization issues for position-independent binaries.

runtime/trace

The garbage collector's background sweeper now yields less frequently, resulting in many fewer extraneous events in execution traces.

strings

The new `CutPrefix` and `CutSuffix` functions are like `TrimPrefix` and `TrimSuffix` but also report whether the string was trimmed.

sync

The new `Map` methods `Swap`, `CompareAndSwap`, and `CompareAndDelete` allow existing map entries to be updated atomically.

syscall

On FreeBSD, compatibility shims needed for FreeBSD 11 and earlier have been removed.

On Linux, additional `CLONE_*` constants are defined for use with the `SysProcAttr.Cloneflags` field.

On Linux, the new `SysProcAttr.CgroupFD` and `SysProcAttr.UseCgroupFD` fields provide a way to place a child process into a specific cgroup.

testing

The new method `B.Elapsed` reports the current elapsed time of the benchmark, which may be useful for calculating rates to report with `ReportMetric`.

Calling `T.Run` from a function passed to `T.Cleanup` was never well-defined, and will now panic.

time

The new time layout constants `DateTime`, `DateOnly`, and `TimeOnly` provide names for three of the most common layout strings used in a survey of public Go source code.

The new `Time.Compare` method compares two times.

`Parse` now ignores sub-nanosecond precision in its input, instead of reporting those digits as an error.

The `Time.MarshalJSON` method is now more strict about adherence to RFC 3339.

unicode/utf16

The new `AppendRune` function appends the UTF-16 encoding of a given rune to a uint16 slice, analogous to `utf8.AppendRune`.