The Go Programming Language

Go 1.9 Release Notes

- Introduction to Go 1.9
- Changes to the language
- Ports
  - ppc64x requires POWER8
  - FreeBSD
  - OpenBSD 6.0
  - Known Issues
- Tools
  - Parallel Compilation
  - Vendor matching with ./...
  - Moved GOROOT
  - Compiler Toolchain
  - Assembler
  - Doc
  - Env
  - Test
  - Pprof
  - Vet
  - Gccgo
- Runtime
  - Call stacks with inlined frames
- Performance
  - Garbage Collector
- Core library
  - Transparent Monotonic Time support
  - New bit manipulation package
  - Test Helper Functions
  - Concurrent Map
  - Profiler Labels
  - Minor changes to the library

Introduction to Go 1.9

The latest Go release, version 1.9, arrives six months after Go 1.8 and is the tenth release in the Go 1.x series. There are two changes to the language: adding support for type aliases and defining when implementations may fuse floating point operations. Most of the 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.

The release adds transparent monotonic time support, parallelizes compilation of functions within a package, better supports test helper functions, includes a new bit manipulation package, and has a new concurrent map type.

Changes to the language

There are two changes to the language.

Go now supports type aliases to support gradual code repair while moving a type between packages. The type alias design document and an article on refactoring cover the problem in detail. In short, a type alias declaration has the form:

```
type T1 = T2
```

This declaration introduces an alias name `T1`—an alternate spelling—for the type denoted by `T2`; that is, both `T1` and `T2` denote the same type.

A smaller language change is that the language specification now states when implementations are allowed to fuse floating point operations together, such as by using an architecture's "fused multiply and add" (FMA) instruction to compute `x*y + z` without rounding the intermediate result `x*y`. To force the intermediate rounding, write `float64(x*y) + z`.

Ports

There are no new supported operating systems or processor architectures in this release.

ppc64x requires POWER8

Both `GOARCH=ppc64` and `GOARCH=ppc64le` now require at least POWER8 support. In previous releases, only `GOARCH=ppc64le` required POWER8 and the big endian `ppc64` architecture supported older hardware.

FreeBSD

Go 1.9 is the last release that will run on FreeBSD 9.3, which is already unsupported by FreeBSD. Go 1.10 will require FreeBSD 10.3+.

OpenBSD 6.0

Go 1.9 now enables PT_TLS generation for cgo binaries and thus requires OpenBSD 6.0 or newer. Go 1.9 no longer supports OpenBSD 5.9.

Known Issues

There are some instabilities on FreeBSD that are known but not understood. These can lead to program crashes in rare cases. See issue 15658. Any help in solving this FreeBSD-specific issue would be appreciated.

Go stopped running NetBSD builders during the Go 1.9 development cycle due to NetBSD kernel crashes, up to and including NetBSD 7.1. As Go 1.9 is being released, NetBSD 7.1.1 is being released with a fix. However, at this time we have no NetBSD builders passing our test suite. Any help investigating the various NetBSD issues would be appreciated.

Tools

Parallel Compilation

The Go compiler now supports compiling a package's functions in parallel, taking advantage of multiple cores. This is in addition to the `go` command's existing support for parallel compilation of separate packages. Parallel compilation is on by default, but it can be disabled by setting the environment variable `GO19CONCURRENTCOMPILATION` to `0`.

Vendor matching with ./...

By popular request, `./...` no longer matches packages in `vendor` directories in tools accepting package names, such as `go test`. To match vendor directories, write `./vendor/...`.

Moved GOROOT

The go tool will now use the path from which it was invoked to attempt to locate the root of the Go install tree. This means that if the entire Go installation is moved to a new location, the go tool should continue to work as usual. This may be overridden by setting `GOROOT` in the environment, which should only be done in unusual circumstances. Note that this does not affect the result of the runtime.GOROOT function, which will continue to report the original installation location; this may be fixed in later releases.

Compiler Toolchain

Complex division is now C99-compatible. This has always been the case in gccgo and is now fixed in the gc toolchain.

The linker will now generate DWARF information for cgo executables on Windows.

The compiler now includes lexical scopes in the generated DWARF if the `-N -l` flags are provided, allowing debuggers to hide variables that are not in scope. The `.debug_info` section is now DWARF version 4.

The values of `GOARM` and `GO386` now affect a compiled package's build ID, as used by the `go` tool's dependency caching.

Assembler

The four-operand ARM `MULA` instruction is now assembled correctly, with the addend register as the third argument and the result register as the fourth and final argument. In previous releases, the two meanings were reversed. The three-operand form, in which the fourth argument is implicitly the same as the third, is unaffected. Code using four-operand `MULA` instructions will need to be updated, but we believe this form is very rarely used. `MULAWT` and `MULAWB` were already using the correct order in all forms and are unchanged.

The assembler now supports `ADDSUBPS/PD`, completing the two missing x86 SSE3 instructions.

Doc

Long lists of arguments are now truncated. This improves the readability of `go doc` on some generated code.

Viewing documentation on struct fields is now supported. For example, `go doc http.Client.Jar`.

Env

The new `go env -json` flag enables JSON output, instead of the default OS-specific output format.

Test

The `go test` command accepts a new `-list` flag, which takes a regular expression as an argument and prints to stdout the name of any tests, benchmarks, or examples that match it, without running them.

Pprof

Profiles produced by the `runtime/pprof` package now include symbol information, so they can be viewed in `go tool pprof` without the binary that produced the profile.

The `go tool pprof` command now uses the HTTP proxy information defined in the environment, using `http.ProxyFromEnvironment`.

Vet

The `vet` command has been better integrated into the `go` tool, so `go vet` now supports all standard build flags while `vet`'s own flags are now available from `go vet` as well as from `go tool vet`.

Gccgo

Due to the alignment of Go's semiannual release schedule with GCC's annual release schedule, GCC release 7 contains the Go 1.8.3 version of gccgo. We expect that the next release, GCC 8, will contain the Go 1.10 version of gccgo.

Runtime

Call stacks with inlined frames

Users of `runtime.Callers` should avoid directly inspecting the resulting PC slice and instead use `runtime.CallersFrames` to get a complete view of the call stack, or `runtime.Caller` to get information about a single caller. This is because an individual element of the PC slice cannot account for inlined frames or other nuances of the call stack.

Specifically, code that directly iterates over the PC slice and uses functions such as `runtime.FuncForPC` to resolve each PC individually will miss inlined frames. To get a complete view of the stack, such code should instead use `CallersFrames`. Likewise, code should not assume that the length returned by `Callers` is any indication of the call depth. It should instead count the number of frames returned by `CallersFrames`.

Code that queries a single caller at a specific depth should use `Caller` rather than passing a slice of length 1 to `Callers`.

`runtime.CallersFrames` has been available since Go 1.7, so code can be updated prior to upgrading to Go 1.9.

Performance

As always, the changes are so general and varied that precise statements about performance are difficult to make. Most programs should run a bit faster, due to speedups in the garbage collector, better generated code, and optimizations in the core library.

Garbage Collector

Library functions that used to trigger stop-the-world garbage collection now trigger concurrent garbage collection. Specifically, `runtime.GC`, `debug.SetGCPercent`, and `debug.FreeOSMemory`, now trigger concurrent garbage collection, blocking only the calling goroutine until the garbage collection is done.

The `debug.SetGCPercent` function only triggers a garbage collection if one is immediately necessary because of the new GOGC value. This makes it possible to adjust GOGC on-the-fly.

Large object allocation performance is significantly improved in applications using large (>50GB) heaps containing many large objects.

The `runtime.ReadMemStats` function now takes less than 100µs even for very large heaps.

Core library

Transparent Monotonic Time support

The `time` package now transparently tracks monotonic time in each `Time` value, making computing durations between two `Time` values a safe operation in the presence of wall clock adjustments. See the package docs and design document for details.

New bit manipulation package

Go 1.9 includes a new package, `math/bits`, with optimized implementations for manipulating bits. On most architectures, functions in this package are additionally recognized by the compiler and treated as intrinsics for additional performance.

Test Helper Functions

The new `(*T).Helper` and `(*B).Helper` methods mark the calling function as a test helper function. When printing file and line information, that function will be skipped. This permits writing test helper functions while still having useful line numbers for users.

Concurrent Map

The new `Map` type in the `sync` package is a concurrent map with amortized-constant-time loads, stores, and deletes. It is safe for multiple goroutines to call a `Map`'s methods concurrently.

Profiler Labels

The `runtime/pprof` package now supports adding labels to `pprof` profiler records. Labels form a key-value map that is used to distinguish calls of the same function in different contexts when looking at profiles with the `pprof` command. The `pprof` package's new `Do` function runs code associated with some provided labels. Other new functions in the package help work with labels.

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.

archive/zip

The ZIP `Writer` now sets the UTF-8 bit in the `FileHeader.Flags` when appropriate.

crypto/rand

On Linux, Go now calls the `getrandom` system call without the `GRND_NONBLOCK` flag; it will now block until the kernel has sufficient randomness. On kernels predating the `getrandom` system call, Go continues to read from `/dev/urandom`.

crypto/x509

On Unix systems the environment variables `SSL_CERT_FILE` and `SSL_CERT_DIR` can now be used to override the system default locations for the SSL certificate file and SSL certificate files directory, respectively.

The FreeBSD file `/usr/local/etc/ssl/cert.pem` is now included in the certificate search path.

The package now supports excluded domains in name constraints. In addition to enforcing such constraints, `CreateCertificate` will create certificates with excluded name constraints if the provided template certificate has the new field `ExcludedDNSDomains` populated.

If any SAN extension, including with no DNS names, is present in the certificate, then the Common Name from `Subject` is ignored. In previous releases, the code tested only whether DNS-name SANs were present in a certificate.

database/sql

The package will now use a cached `Stmt` if available in `Tx.Stmt`. This prevents statements from being re-prepared each time `Tx.Stmt` is called.

The package now allows drivers to implement their own argument checkers by implementing `driver.NamedValueChecker`. This also allows drivers to support `OUTPUT` and `INOUT` parameter types. `Out` should be used to return output parameters when supported by the driver.

`Rows.Scan` can now scan user-defined string types. Previously the package supported scanning into numeric types like `type Int int64`. It now also supports scanning into string types like `type String string`.

The new `DB.Conn` method returns the new `Conn` type representing an exclusive connection to the database from the connection pool. All queries run on a `Conn` will use the same underlying connection until `Conn.Close` is called to return the connection to the connection pool.

encoding/asn1

The new `NullBytes` and `NullRawValue` represent the ASN.1 NULL type.

encoding/base32

The new Encoding.WithPadding method adds support for custom padding characters and disabling padding.

encoding/csv

The new field `Reader.ReuseRecord` controls whether calls to `Read` may return a slice sharing the backing array of the previous call's returned slice for improved performance.

fmt

The sharp flag ('#') is now supported when printing floating point and complex numbers. It will always print a decimal point for `%e`, `%E`, `%f`, `%F`, `%g` and `%G`; it will not remove trailing zeros for `%g` and `%G`.

hash/fnv

The package now includes 128-bit FNV-1 and FNV-1a hash support with `New128` and `New128a`, respectively.

html/template

The package now reports an error if a predefined escaper (one of "html", "urlquery" and "js") is found in a pipeline and does not match what the auto-escaper would have decided on its own. This avoids certain security or correctness issues. Now use of one of these escapers is always either a no-op or an error. (The no-op case eases migration from text/template.)

image

The `Rectangle.Intersect` method now returns a zero `Rectangle` when called on adjacent but non-overlapping rectangles, as documented. In earlier releases it would incorrectly return an empty but non-zero `Rectangle`.

image/color

The YCbCr to RGBA conversion formula has been tweaked to ensure that rounding adjustments span the complete [0, 0xffff] RGBA range.

image/png

The new `Encoder.BufferPool` field allows specifying an `EncoderBufferPool`, that will be used by the encoder to get temporary `EncoderBuffer` buffers when encoding a PNG image. The use of a `BufferPool` reduces the number of memory allocations performed while encoding multiple images.

The package now supports the decoding of transparent 8-bit grayscale ("Gray8") images.

math/big

The new `IsInt64` and `IsUint64` methods report whether an `Int` may be represented as an `int64` or `uint64` value.

mime/multipart

The new `FileHeader.Size` field describes the size of a file in a multipart message.

net

The new `Resolver.StrictErrors` provides control over how Go's built-in DNS resolver handles temporary errors during queries composed of multiple sub-queries, such as an A+AAAA address lookup.

The new `Resolver.Dial` allows a `Resolver` to use a custom dial function.

`JoinHostPort` now only places an address in square brackets if the host contains a colon. In previous releases it would also wrap addresses in square brackets if they contained a percent ('%') sign.

The new methods `TCPConn.SyscallConn`, `IPConn.SyscallConn`, `UDPConn.SyscallConn`, and `UnixConn.SyscallConn` provide access to the connections' underlying file descriptors.

It is now safe to call `Dial` with the address obtained from `(*TCPListener).String()` after creating the listener with `Listen("tcp", ":0")`. Previously it failed on some machines with half-configured IPv6 stacks.

net/http

The `Cookie.String` method, used for `Cookie` and `Set-Cookie` headers, now encloses values in double quotes if the value contains either a space or a comma.

Server changes:

- `ServeMux` now ignores ports in the host header when matching handlers. The host is matched unmodified for `CONNECT` requests.
- `Server.WriteTimeout` now applies to HTTP/2 connections and is enforced per-stream.
- HTTP/2 now uses the priority write scheduler by default. Frames are scheduled by following HTTP/2 priorities as described in RFC 7540 Section 5.3.
- The HTTP handler returned by `StripPrefix` now calls its provided handler with a modified clone of the original `*http.Request`. Any code storing per-request state in maps keyed by `*http.Request` should use `Request.Context`, `Request.WithContext`, and `context.WithValue` instead.
- `LocalAddrContextKey` now contains the connection's actual network address instead of the interface address used by the listener.

Client & Transport changes:

- The `Transport` now supports making requests via SOCKS5 proxy when the URL returned by `Transport.Proxy` has the scheme `socks5`.

net/http/fcgi

The new `ProcessEnv` function returns FastCGI environment variables associated with an HTTP request for which there are no appropriate `http.Request` fields, such as `REMOTE_USER`.

net/http/httptest

The new `Server.Client` method returns an HTTP client configured for making requests to the test server.

The new `Server.Certificate` method returns the test server's TLS certificate, if any.

net/http/httputil

The `ReverseProxy` now proxies all HTTP/2 response trailers, even those not declared in the initial response header. Such undeclared trailers are used by the gRPC protocol.

os

The `os` package now uses the internal runtime poller for file I/O. This reduces the number of threads required for read/write operations on pipes, and it eliminates races when one goroutine closes a file while another is using the file for I/O.

On Windows, `Args` is now populated without `shell32.dll`, improving process start-up time by 1-7 ms.

os/exec

The `os/exec` package now prevents child processes from being created with any duplicate environment variables. If `Cmd.Env` contains duplicate environment keys, only the last value in the slice for each duplicate key is used.

os/user

`Lookup` and `LookupId` now work on Unix systems when `CGO_ENABLED=0` by reading the `/etc/passwd` file.

`LookupGroup` and `LookupGroupId` now work on Unix systems when `CGO_ENABLED=0` by reading the `/etc/group` file.

reflect

The new `MakeMapWithSize` function creates a map with a capacity hint.

runtime

Tracebacks generated by the runtime and recorded in profiles are now accurate in the presence of inlining. To retrieve tracebacks programmatically, applications should use `runtime.CallersFrames` rather than directly iterating over the results of `runtime.Callers`.

On Windows, Go no longer forces the system timer to run at high resolution when the program is idle. This should reduce the impact of Go programs on battery life.

On FreeBSD, `GOMAXPROCS` and `runtime.NumCPU` are now based on the process' CPU mask, rather than the total number of CPUs.

The runtime has preliminary support for Android O.

runtime/debug

Calling `SetGCPercent` with a negative value no longer runs an immediate garbage collection.

runtime/trace

The execution trace now displays mark assist events, which indicate when an application goroutine is forced to assist garbage collection because it is allocating too quickly.

"Sweep" events now encompass the entire process of finding free space for an allocation, rather than recording each individual span that is swept. This reduces allocation latency when tracing allocation-heavy programs. The sweep event shows how many bytes were swept and how many were reclaimed.

sync

`Mutex` is now more fair.

syscall

The new field `Credential.NoSetGroups` controls whether Unix systems make a `setgroups` system call to set supplementary groups when starting a new process.

The new field `SysProcAttr.AmbientCaps` allows setting ambient capabilities on Linux 4.3+ when creating a new process.

On 64-bit x86 Linux, process creation latency has been optimized with use of `CLONE_VFORK` and `CLONE_VM`.

The new `Conn` interface describes some types in the `net` package that can provide access to their underlying file descriptor using the new `RawConn` interface.

testing/quick

The package now chooses values in the full range when generating `int64` and `uint64` random numbers; in earlier releases generated values were always limited to the [-2^{62}, 2^{62}) range.

In previous releases, using a nil `Config.Rand` value caused a fixed deterministic random number generator to be used. It now uses a random number generator seeded with the current time. For the old behavior, set `Config.Rand` to `rand.New(rand.NewSource(0))`.

text/template

The handling of empty blocks, which was broken by a Go 1.8 change that made the result dependent on the order of templates, has been fixed, restoring the old Go 1.7 behavior.

time

The new methods `Duration.Round` and `Duration.Truncate` handle rounding and truncating durations to multiples of a given duration.

Retrieving the time and sleeping now work correctly under Wine.

If a `Time` value has a monotonic clock reading, its string representation (as returned by `String`) now includes a final field "`m=±value`", where `value` is the monotonic clock reading formatted as a decimal number of seconds.

The included `tzdata` timezone database has been updated to version 2017b. As always, it is only used if the system does not already have the database available.