Engineer

Go

lenguaje compilado con tipado fuerte y recolector de basura

Qué es

Go es un lenguaje de programación de propósito general nacido en Google, diseñado pensando en la programación de sistemas y con la concurrencia incorporada al propio lenguaje, en vez de añadida como biblioteca. A menudo se llama golang, por la primera dirección web del proyecto, golang.org; las preguntas frecuentes del proyecto insisten en que el lenguaje se llama simplemente Go, escrito Go y nunca GO.

Robert Griesemer, Rob Pike y Ken Thompson esbozaron sus objetivos en una pizarra el 21 de septiembre de 2007. El 10 de noviembre de 2009, el lenguaje se publicó como proyecto de código abierto. Go 1 llegó el 28 de marzo de 2012 con la promesa de que los programas escritos para él seguirían compilándose y ejecutándose, sin cambios, en todas las versiones posteriores de Go 1. La promesa se mantiene: la especificación describe ahora la versión del lenguaje go1.27 y Go 1.27.0 se publicó el 19 de agosto de 2026. Los genéricos llegaron en Go 1.18 y los diseñadores los calificaron como el mayor cambio desde la primera publicación de código abierto.

Las preguntas frecuentes explican que el lenguaje surgió de una elección que sus autores estaban cansados de hacer: entre compilación eficiente, ejecución eficiente y facilidad de programación. Go pretende ofrecer las tres. Un programa se construye con paquetes y la cadena de herramientas se maneja con una orden, `go`, que compila, prueba, formatea y recupera dependencias. La palabra clave `go` delante de una llamada a función inicia una goroutine, una función que se ejecuta junto al resto del programa. Son tan ligeras que, según las preguntas frecuentes, cientos de miles pueden compartir un espacio de direcciones.
Los canales transportan valores entre ellas y Effective Go resume el estilo buscado con un lema: «No se comunique compartiendo memoria; comparta memoria comunicándose».

Dos hábitos distinguen el código Go dondequiera que aparezca. El programa `gofmt` presenta cada archivo fuente con una disposición estándar, por lo que el formato deja de discutirse en las revisiones. Además, el compilador rechaza un programa con una variable o una importación sin usar: la primera puede ocultar un fallo y la segunda ralentiza todas las compilaciones posteriores.

Utilice el concepto en una decisión

Aplique «¿Qué es Go?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

La decisión que suele resolver Go es con qué escribir un servicio de red, una herramienta de línea de órdenes o una pieza de infraestructura. Las alternativas quedan a ambos lados: un lenguaje de scripting por uno, y C, C++ o Rust por el otro.

Frente a un lenguaje de scripting, Go ofrece despliegue sencillo y detección temprana de errores. El enlazador produce binarios enlazados estáticamente de forma predeterminada: el resultado es un archivo que lleva su propio entorno de ejecución y funciona en un servidor sin nada más instalado. El precio en espacio, según las preguntas frecuentes, es de un par de megabytes para un programa que imprime “hello, world”, frente a unos 750 kB del mismo programa en C. Los errores de tipos aparecen al compilar, en lugar de cuando se ejecuta en producción una rama poco usada. Uno de los objetivos iniciales era que incluso un ejecutable grande se compilara en pocos segundos en un solo ordenador.

Frente a C, C++ o Rust, Go ofrece sencillez y seguridad de memoria a cambio de control. Un recolector de basura recupera la memoria y libera al código habitual de toda una clase de errores de uso después de liberar la memoria. A cambio, el programa no decide exactamente cuándo se libera y cada binario incorpora el entorno de ejecución que lo hace. Para la mayoría de los servidores, el intercambio es adecuado; para un kernel, un dispositivo embebido pequeño o un bucle crítico que no tolera un recolector, no lo es.

El argumento menos llamativo es el tiempo. En 2023, Russ Cox escribió que un Go 2 que rompiera los programas antiguos «no va a ocurrir nunca» y que el proyecto, en cambio, «redoblaría su apuesta por la compatibilidad». La promesa se paga con actualizaciones y no con reescrituras. Cada seis meses, en febrero y agosto, sale una nueva versión principal, y cada una recibe soporte hasta que existen dos posteriores. Un equipo que quiera correcciones de seguridad avanza, por tanto, aproximadamente una vez al año.

La sencillez del lenguaje también obliga a escribir algunas cosas con más extensión. Los errores son valores ordinarios, que se devuelven y comprueban donde ocurren: resulta explícito y repetitivo casi a partes iguales.

Cómo se aplica en la práctica

Iniciar una goroutine cuesta poco y perderla es igual de fácil. La página sobre trasladar el trabajo lento a una cola de tareas River lo dice claramente: si el trabajo lento se hace en una goroutine, desaparece en cuanto se reinicia el proceso, en pleno despliegue, sin constancia de que debía ocurrir. La empresa lo trasladó a River, que conserva la cola en PostgreSQL, de modo que una tarea y las filas que modifica se confirman o revierten juntas. Detrás hay 15 módulos de trabajadores y 8 tareas programadas, con 20 tareas pg_cron por debajo. Una goroutine aporta concurrencia; la durabilidad debe proceder de otro lugar.

Las pruebas vienen con la cadena de herramientas, y la batería de pruebas por capas muestra a qué equivale eso. Su API de Go contiene 981 funciones de prueba en 310 archivos: una de cuatro capas probadas cada una donde reside su comportamiento, junto a las especificaciones de prueba del frontend y las pruebas SQL de la propia base de datos.

El programa continuo de seguridad también se apoya en las herramientas del lenguaje. Las dependencias de Go se comprueban con govulncheck, y una vulnerabilidad reconocida por las comprobaciones configuradas puede hacer fallar la compilación. La página identifica el volumen de hallazgos como el coste real del análisis. govulncheck está diseñado para reducir ese coste: solo comunica una vulnerabilidad cuando las funciones del propio código llaman a la función afectada, directa o indirectamente.

Dónde surgen los problemas

Una función comunica un error sin que nada haya fallado. Un valor de interfaz en Go contiene un tipo y un valor, y solo es igual a nil si ambos están sin establecer. Una función que devuelve un puntero nil de un tipo de error concreto mediante la interfaz `error` entrega al llamador una interfaz que contiene un tipo y, por tanto, no es nil:

```
func save() error {
    var err *ValidationError // a nil pointer
    if invalid() {
        err = &ValidationError{}
    }
    return err // never equal to nil: the interface holds a type
}
```

Devuelva un `nil` literal cuando haya éxito y declare cada resultado de error como `error`, no como un tipo concreto.

Un servicio que superó todas las pruebas termina abruptamente con tráfico real. Las operaciones sobre mapas se dejan deliberadamente sin sincronización: bloquear cada acceso ralentizaría la mayoría de los programas. Las preguntas frecuentes advierten de que «el acceso no controlado a un mapa puede provocar la terminación del programa». La causa es que una goroutine escribe en un mapa mientras otra lo lee o escribe; las lecturas por sí solas son seguras. Proteja el mapa con un mutex, asígnelo a una sola goroutine a la que las demás accedan por un canal o utilice `sync.Map` para los casos para los que se diseñó. Después ejecute las pruebas con `go test -race`. En un programa típico, el detector puede multiplicar el consumo de memoria entre cinco y diez veces y el tiempo de ejecución entre dos y veinte. Solo encuentra carreras en rutas de código que realmente se ejecutan, por lo que un binario compilado con `-race` y sometido a carga realista encuentra más.

Las goroutines iniciadas en un bucle trabajan todas sobre el último elemento. Antes de Go 1.22, un bucle `for` tenía una sola variable compartida por todas las iteraciones: un cierre que la capturaba veía el valor que tuviera cuando por fin se ejecutaba la goroutine. Go 1.22 da a cada iteración su propia variable, pero solo en módulos cuyo go.mod declara `go 1.22` o posterior. El código antiguo mantiene su significado incluso si lo compila una cadena de herramientas más

reciente. Suba la versión de la línea `go` y ejecute `go vet` y las pruebas. Una prueba que empieza a fallar tras el cambio a menudo llevaba todo el tiempo comprobando un valor equivocado.

La compilación en una máquina nueva se detiene para descargar otro Go. Desde Go 1.21, la línea `go` es un requisito. Cuando la cadena instalada es anterior a la exigida por un módulo, la orden `go` recupera esa versión más nueva como módulo y la ejecuta desde la caché de módulos. En un portátil resulta cómodo; en una compilación aislada de la red o una imagen de contenedor fijada resulta una sorpresa. Instale la cadena de herramientas que indica go.mod y establezca la variable de entorno `GOTOOLCHAIN` cuando una compilación deba usar una versión exacta.

Lecturas adicionales

- Especificación del lenguaje de programación Go — El manual de referencia de cada construcción del lenguaje, versionado con cada publicación.
- Preguntas frecuentes — Por qué se creó el lenguaje, sus decisiones de diseño y los problemas de errores nil y mapas descritos antes.
- Go 1 y el futuro de los programas Go — La promesa de compatibilidad y las excepciones que permite.
- Historial de versiones — Todas las versiones desde Go 1, con la política de soporte al principio.
- Detector de carreras de datos — Cómo ejecutarlo, qué detecta y qué cuesta.
- Corrección de los bucles for en Go 1.22 — El cambio de las variables de bucle, por qué se vincula a go.mod y cómo probar el código antiguo frente a él.



Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es Go?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/go/