Engineer

Go

∞ コンパイル型で、強い型付けと自動メモリー回収を持つ言語

どのようなものか

GoはGoogleで生まれた汎用プログラミング言語です。システムプログラミングを念頭に設計され、並行処理はライブラリーとして後から加えたものではなく、言語自体に組み込まれています。プロジェクトの最初のウェブアドレスgolang.orgにちなみ、golangともよく呼ばれます。ただし、公式FAQは、言語名はあくまでGo、表記もGoであり、GOではないと明言しています。

Robert Griesemer、Rob Pike、Ken Thompsonが2007年9月21日にホワイトボードへ最初の目標を書き、2009年11月10日にオープンソースのプロジェクトとして公開しました。2012年3月28日のGo 1では、その版向けに書いたプログラムは、以後のすべてのGo 1でも変更せずにコンパイルでき、動き続けると約束しました。この約束は今も続きます。現在の仕様が記す言語バージョンはgo1.27で、Go 1.27.0は2026年8月19日に公開されました。ジェネリクスはGo 1.18で入り、設計者たちは最初のオープンソース公開以来、最大の変更と呼びました。

FAQによれば、Goは、作者たちが繰り返すのにうんざりしていた選択から生まれました。効率よくコンパイルできること、効率よく実行できること、書きやすいことの、どれを選ぶかという選択です。Goは、この三つの両立を目指します。プログラムはパッケージで構成し、ツールチェーンは `go` という一つのコマンドで、ビルド、テスト、整形、依存関係の取得を行います。関数呼び出しの前へキーワード `go` を置くと、プログラムの残りと並行して動く関数、ゴルーチンを起動します。FAQによれば、数十万個が一つのアドレス空間を共有できるほど、負担は小さいものです。チャネルがゴルーチンの間で値を運び、Effective Goは、意図する書き方を「メモリーを共有して通信してはいけない。代わりに、通信によってメモリーを共有しよう」という標語で表します。

どこで見ても、Goのコードには二つの習慣があります。 `gofmt` はすべてのソースファイルを一つの標準形式で出力するため、レビューで整形を議論しません。また、未使用の変数や未使用のインポートがあるプログラムは、コンパイラーが

拒否します。前者はバグを隠す可能性があり、後者は以後のすべてのビルドを遅くするためです。

概念を判断に生かす

「Goとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 データの保存先、処理のルール、他のシステムとの接続方法など、利用者の目に見えない部分をつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

Goを検討するのは、ネットワークサービス、コマンドラインツール、基盤の一部を何で書くか決めるときが普通です。選択肢の一方にはスクリプト言語、もう一方にはC、C++、Rustがあります。

スクリプト言語と比べると、Goにはデプロイの単純さと、早くエラーが分かる利点があります。リンカーは既定で静的リンクしたバイナリーを作るため、ビルド結果はランタイムも内包する一ファイルになり、ほかに何も導入していないサーバーで動きます。FAQによれば、代償として「hello, world」を表示するだけで二メガバイトほどになり、Cの同じプログラムの約750 kBより大きくなります。型のエラーは、本番でめったに通らない分岐を実行したときではなく、コンパイル時に分かります。大きい実行ファイルでも、一台のコンピューターで数秒のうちにビルドできるようにすることも、当初の目標でした。

C、C++、Rustと比べると、Goは制御の細かさと引き換えに、単純さとメモリー安全性を提供します。ガベージコレクターがメモリーを回収するため、普通のコードでは、解放後の使用という種類の誤りをまとめて避けられます。一方、いつ解放するかをプログラムが正確には決められず、すべてのバイナリーに、その処理を行うランタイムが付きます。大半のサーバーでは妥当な交換条件です。カーネル、小さい組み込み機器、コレクターの動作を許容できない高頻度のループには向きません。

目立たないもう一つの理由は、時間です。Russ Coxは2023年、古いプログラムを壊すGo 2は「決して実現しない」とし、代わりに「互換性をさらに重視する」と書きました。この約束のおかげで、書き直しではなくアップグレードに労力を

使えます。新しいメジャーリリースは半年ごと、2月と8月に出て、それぞれ、より新しい版が二つ出るまでサポートされます。セキュリティ修正を受けたいチームは、年に一度ほど更新することになります。

言語仕様は小さく、明示的に詳しく書く必要のある部分もあります。エラーは普通の値で、発生した場所で返し、検査します。明示的であるのと同じくらい、繰り返しも多くなります。

実際の活用例

ゴルーチンは簡単に起動でき、同じくらい簡単に失われます。遅い処理をRiverのジョブキューへ移した事例は、はっきり説明します。遅い処理をゴルーチンで行うと、デプロイ中にプロセスを再起動した瞬間、実行すべきだったという記録も残さず消えます。自社はその処理を、PostgreSQLにキューを置くRiverへ移し、ジョブと、それが触る行を一緒にコミット、またはロールバックするようにしました。その後ろには15のワーカーモジュールと8つの定期タスクがあり、さらに下で20のpg_cronジョブが動きます。ゴルーチンが与えるのは並行処理で、永続性は別の場所から得る必要があります。

テスト機能はツールチェーンに含まれ、層ごとのテストスイートが、その積み重ねを示します。Go APIは310ファイルに計981のテスト関数を持ち、フロントエンドの仕様テストやデータベース自体のSQLテストと並ぶ、四つの層の一つです。どの層も、振る舞いがある場所でテストします。

継続的なセキュリティ検査も、言語の道具に支えられています。Goの依存関係をgovulncheckで調べ、設定した検査で認識できる脆弱性は、ビルドを失敗させることがあります。事例は検出結果の量こそ検査の本当の負担とします。govulncheckも、その負担を意識した設計で、コード自身の関数が、直接またはほかの関数を介して、影響を受ける関数を呼ぶ場合だけ報告します。

問題が起きるところ

何も問題がないのに、関数がエラーを返します。 Goのインターフェース値は型と値を持ち、両方とも設定されていない場合だけnilと等しくなります。具体的なエラー型のnilポインターを `error` インターフェース経由で返すと、呼び出し元は型を持つインターフェースを受け取り、それはnilではありません。

```
func save() error {
    var err *ValidationError // nilポインター
    if invalid() {
        err = &ValidationError{}
    }
    return err // nilとは等しくならない：インターフェースが型を保持する
}
```

成功時にはリテラルの `nil` を返し、すべてのエラーの戻り値は、具体的な型ではなく `error` として宣言してください。

すべてのテストを通ったサービスが、実際のトラフィックで落ちます。 マップの操作は、すべてのアクセスをロックすると大半のプログラムが遅くなるため、意図的に同期しません。FAQは「制御されていないマップへのアクセスは、プログラムを異常終了させる可能性がある」と警告します。一つのゴルーチンが書く間に、別のものが読むか書くことが原因です。読むだけなら安全です。ミューテックスで守るか、マップを一つのゴルーチンだけに管理させ、ほかからはチャネルでアクセスするか、設計された用途に合う場合に `sync.Map` を使います。そのうえで `go test -race` でテストします。典型的なプログラムでは、競合検出によってメモリー使用量が五倍から十倍、実行時間が二倍から二十倍になる場合があります。また、実際に通ったコード経路の競合しか見つけないため、 `-race` でビルドしたバイナリーを現実的な負荷で動かすと、より多くを見つけられます。

ループ内で起動したゴルーチンが、すべて最後の項目を処理します。 Go 1.22より前の `for` ループは、すべての反復で一つの変数を共有しました。それを捕捉したクロージャーは、ゴルーチンが実際に動いた時点の値を読みます。Go 1.22では反復ごとに変数を持ちますが、go.modで `go 1.22` 以降を宣言するモジュールに限ります。古いコードは、新しいツールチェーンでビルドしても、古い意味を保ちます。 `go` 行の版を上げ、 `go vet` とテストを実行してください。変更後に失敗し始めるテストは、以前から誤った値を検査していた場合がよくあります。

新しいマシンのビルドが、別のGoを取得するために止まります。 Go 1.21以降、 `go` 行は必須のバージョンを表します。導入済みのツールチェーンがモジュールの求める版より古ければ、 `go` コマンドは、新しいものをモジュールとして取得し、モジュールキャッシュから動かします。ノートPCでは便利でも、外部ネットワークから隔離したビルドや、版を固定したコンテナーイメージでは不意を突かれます。go.modが指定するツールチェーンを導入し、指定した版だけを使う必要があるビルドでは、環境変数 `GOTOOLCHAIN` を設定してください。

参考資料

- Go言語仕様 — 言語のすべての構文についての参照マニュアルで、リリースごとに版があります。
- よくある質問（FAQ） — 言語を作った理由と設計上の決定、および上で扱ったnilエラーとマップの問題です。
- Go 1とGoプログラムの将来 — 互換性の約束と、認める例外です。
- リリース履歴 — Go 1以降の全リリースで、冒頭にサポート方針があります。
- データ競合検出器 — 実行方法、検出するもの、必要な負担です。
- Go 1.22でのforループの修正 — ループ変数の変更、go.modと結び付ける理由、古いコードへの影響をテストする方法です。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「Goとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/go/