Effective Go - Tóm tắt các kỹ thuật và ví dụ

Introduction

- Go có đặc điểm và idiom riêng, không nên dịch máy từ C++/Java sang Go.
- Hiểu rõ idiom, convention về đặt tên, format, cấu trúc chương trình giúp code dễ đọc, dễ bảo trì.

Formatting

- Sử dụng `gofmt` để format code tự động, thống nhất style.
- Dùng tab để thụt lề, không giới hạn độ dài dòng, ít dùng dấu ngoặc đơn hơn C/Java.

Commentary

- Dùng `//` cho comment dòng, `/* */` cho block comment.
- Comment trước khai báo top-level sẽ thành doc comment cho package.

Names

- Tên package: ngắn, thường là chữ thường, không dùng underscore hay mixedCaps.
- Getter: Không dùng tiền tố `Get`, chỉ cần viết hoa để export.
- Interface một hàm: Đặt tên bằng hậu tố -er (Reader, Writer...).
- Dùng MixedCaps hoặc mixedCaps cho tên nhiều từ.

Semicolons

- Go tự động chèn dấu `;` khi cần, không viết tường minh.
- Không được đặt `{` ở dòng mới sau control structure.

Control structures

- Không có `do`/`while`, chỉ có `for` (3 dạng: C-style, while, for(; ;)).
- `if`, `switch` có thể có init statement.
- `switch` không tự động fallthrough, có thể dùng nhiều giá trị cho một case.
- Có type switch để kiểm tra kiểu động của interface.

Functions

- Hỗ trợ trả về nhiều giá trị, thường dùng để trả về kết quả và lỗi.
- Có thể đặt tên cho giá trị trả về (named result parameters).
- `defer` dùng để hoãn thực thi hàm cho đến khi hàm hiện tại return (thường dùng để đóng file, unlock mutex...).
- Deferred function thực thi theo thứ tự LIFO.

Data

Allocation with `new`

- `new(T)` cấp phát vùng nhớ zero-value cho kiểu T, trả về con trỏ `*T`.
- Nên thiết kế struct sao cho zero-value dùng được luôn.

Constructors and composite literals

- Dùng composite literal để khởi tạo struct, array, map, slice.
- Có thể trả về địa chỉ biến local (Go sẽ giữ vùng nhớ).

Allocation with `make`

- `make` chỉ dùng cho slice, map, channel. Khởi tạo và trả về giá trị đã sẵn sàng dùng.

Arrays

- Array là value, truyền vào hàm sẽ copy toàn bộ.
- Kích thước là một phần của kiểu.
- Nếu muốn truyền tham chiếu, dùng con trỏ.

Slices

- Slice là wrapper cho array, truyền slice sẽ chia sẻ underlying array.
- Có thể thay đổi độ dài slice trong capacity.
- Dùng built-in `append` để thêm phần tử.

Two-dimensional slices

- Slice 2 chiều là slice của slice.
- Có thể cấp phát từng dòng riêng hoặc một block lớn rồi chia nhỏ.

Maps

- Map là built-in, key phải so sánh được.
- Truy cập key không tồn tại trả về zero value.
- Dùng `comma ok` để kiểm tra tồn tại key.
- Dùng `delete` để xóa key.

Printing

- Dùng package `fmt` với các hàm Printf, Println, Sprintf...
- `%v`, `%+v`, `%#v`, `%T` để in giá trị, kiểu, struct...
- Định nghĩa method `String() string` để custom in struct.

Append

- `append(slice, elems...)` thêm phần tử vào slice, trả về slice mới.
- Dùng `...` để append một slice vào slice khác.

Initialization

Constants

- Hằng số được tạo lúc compile, dùng `iota` để tạo enum.
- Có thể định nghĩa method cho type mới để custom in hằng số.

Variables

- Biến có thể khởi tạo bằng biểu thức runtime.

The init function

- Mỗi file có thể có hàm `init` để khởi tạo trạng thái trước khi chạy main.

Methods

Pointers vs. Values

- Method có thể gắn cho value hoặc pointer receiver.
- Method pointer receiver chỉ gọi được trên pointer.
- Nếu method cần thay đổi giá trị receiver, dùng pointer receiver.

Interfaces and other types

Interfaces

- Interface là tập hợp các method, type nào implement đủ method sẽ thỏa mãn interface.
- Có thể implement nhiều interface.
- Interface thường đặt tên bằng hậu tố -er.

Conversions

- Có thể convert giữa các type có cùng underlying type.
- Dùng type assertion hoặc type switch để lấy giá trị cụ thể từ interface.

Interface conversions and type assertions

- Dùng type assertion để kiểm tra và lấy giá trị cụ thể từ interface.
- Dùng `comma ok` để kiểm tra an toàn.

Generality

- Nếu type chỉ để implement interface, chỉ export interface, không export type.

Interfaces and methods

- Có thể gắn method cho bất kỳ type nào (trừ pointer, interface).
- Có thể dùng function, channel, int... làm HTTP handler nhờ interface.

The blank identifier

- `_` dùng để bỏ qua giá trị không cần thiết (vd: for range, multiple return).
- Dùng để tránh lỗi unused import/variable khi đang phát triển.
- Import package chỉ để side effect: `import _ "pkg"`
- Dùng để kiểm tra compile-time một type có implement interface không.

Embedding

- Go không có subclassing, nhưng có thể embed type vào struct hoặc interface.
- Embed interface: interface mới là union của các interface con.
- Embed struct: struct ngoài sẽ có luôn method của struct được embed.
- Nếu trùng tên field/method, field/method ở ngoài sẽ che field/method ở trong.

Concurrency

Share by communicating

- Go khuyến khích chia sẻ dữ liệu qua channel, không chia sẻ bộ nhớ trực tiếp.

Goroutines

- Goroutine là hàm chạy song song, rất nhẹ, tạo bằng từ khóa `go`.
- Closure trong goroutine sẽ giữ biến ngoài scope.

Channels

- Channel là kiểu dữ liệu truyền thông tin giữa goroutine.
- Channel có thể buffered hoặc unbuffered.
- Dùng channel như semaphore để giới hạn số goroutine chạy song song.

Channels of channels

- Channel là giá trị first-class, có thể truyền channel qua channel.
- Dùng để xây dựng hệ thống RPC song song, không cần mutex.

Parallelization

- Dùng goroutine và channel để chia nhỏ công việc, tận dụng nhiều CPU core.
- Dùng `runtime.NumCPU()` hoặc `runtime.GOMAXPROCS(0)` để lấy số core.

A leaky buffer

- Dùng channel buffered để quản lý pool/buffer, tận dụng GC để thu hồi bộ nhớ.

Errors

- Lỗi thường trả về qua giá trị `error` (interface có method `Error() string`).
- Có thể định nghĩa struct lỗi riêng để bổ sung thông tin.
- Dùng type assertion để kiểm tra loại lỗi cụ thể.

Panic

- Dùng `panic` khi gặp lỗi không thể phục hồi.
- `panic` sẽ unwind stack, chạy các deferred function.

Recover

- Dùng `recover` trong deferred function để bắt panic, tránh crash toàn bộ chương trình.
- Chỉ có tác dụng khi gọi trực tiếp trong deferred function.
- Pattern: Dùng panic để báo lỗi nội bộ, recover ở ngoài để chuyển thành error trả về.

A web server

- Ví dụ hoàn chỉnh về web server tạo QR code.
- Sử dụng package `net/http`, `html/template`, flag, log.
- Handler nhận request, render template với dữ liệu từ form.
- Template HTML dùng `{{if .}}`, `{{.}}` để hiển thị dữ liệu