Go Learning Session

Tom Arrell

Tue 1st Dec, 2020

Overview

1. History of Go
2. Idiomatic Go
3. Go and DDD
4. Takeaways
5. Q&A

Brief history

- Created by Robert Griesemer, Rob Pike, and Ken Thompson
- Early release in late 2009
- Frustration with existing languages and environments at Google
- Choose either efficient compilation, efficient execution, or ease of programming
- Working with Go is intended to be fast

Guiding principles

- Reduce typing in both senses (no forward declaration, header files)
- Reduce clutter and complexity
- Syntax is light on keywords
- No type heirarchy
- No explicit relationships
- Keep concepts orthogonal (clear separation of concerns)
  - Early contracts proposal not orthogonal enough to interfaces

Sounds neat!

Effective Go

So how can we write idiomatic and effective Go?

Effective Go

Firstly, you need to try to forget some of the patterns that you may have used in the past. Go is *not* an object oriented language. There is no support for inheritance in Go.

There are also other principles, such as DRY, which are also challenged when writing Go.

Writing effective Go is ultimatley about being **pragmatic**.

Composition over inheritance

Go does not support inheritance. This is intentional, as inheritance commonly causes tight coupling, unused methods and ultimately confusion.

Instead, Go favours composition. Where a type is made up of the sum of its parts.

Composition over inheritance

An example of this is an embedded interface.

```
type Reader interface {
    Read(p []byte) (n int, err error)
}

type Writer interface {
    Write(p []byte) (n int, err error)
}

type ReadWriter interface {
    Reader
    Writer
}
```

The same basic idea applies to structs as well.

DRY

Dry principles have in the past been used to justify adding new abstractions around existing mechanisms.

These abstractions hurt the readability of the code.

In Go, you should focus on minimising the indirection in your code, this means *duplicating where reasonable*.

SOLID Principles

A very prominent figure in the Go community has already put together a much better resource than I would be able to on how you can apply the SOLID principles to Go.

We'll take a brief look at the points he makes.

https://dave.cheney.net/2016/08/20/solid-go-design

DDD Structure

- Doesn't make a lot of sense in Go
- Quite a bit of boilerplate
- Use domain package, no dependencies
- Group package by dependencies
- Keep it simple!

Fin.

Useful resources:

- Effective Go: link
- Dave Cheney's Blog: link
- Standard package layout: link
- Go By Example: link