Go Academy

#4 DI, context and in-memory database

www.3fs.si

Dependency injection / management

- How to manage when multiple dependencies are required to solve a problem?
- And still be able to properly unit test all possible flows in your unit of code?
- Interfaces can be used to reduce the effort required to mock / stub external components.
- Methods can be used to ease handling a set of dependencies.
- Beware of unit of code having access to too many components. Can get very hard to change.

Dependency injection / management

- `./01-initial`
- `./02-attributes`
- `./03-struct`
- `./04-interfaces`

Context

- "A frame that surrounds the event and provides resources for its appropriate interpretation." (source)
- Can carry cancelation signal to goroutines.
- Goroutine can still handle the cancelation message as it pleases (not a kill switch).
- Supports simple key / value storage to allow us to pass data between different units inside a single event.

Context (`./05-context`)

- `context.Background()` creates an empty context object
- `context.TODO()` notes that the correct context still has to be defined
- Each call to `context.With*` creates a new context object linked to parent
  - Cancellation signal is propagated downwards

Context - WithCancel (`./06-cancel, ./07-cautious`)

- Allows one process (goroutine) to signal another that it should stop its work
- Recipient goroutine is not obliged to respect this signal
- Sender should not expect the recipient to handle it perfectly
- Passed through with a channel available at `"ctx.Done() chan struct{}"`
- Once `Cancel()` is called `<- ctx.Done()` will no longer block
- Better to call `Cancel()` multiple times then never

Context - withDeadline, withTimeout (`./08-timeout`)

- Adds a time component to the cancelation process
- Context will cancel itself automatically after a specified period of time.
- `WithDeadline()` accepts `time.Time`
- `WithTimeout()` accepts `time.Duration`
- Both also return a `Cancel()` method that can cancel the context before the specified time span.

Context - WithValue (`./09-withValue`)

- Useful to pass around event specific data
- In theory it should only be used for smaller / scalar values
- Can make code error prone as compiler can't catch possible issues
- Custom type should be used for keys to remove possibility of key collision with other packages

In-memory database (`./10-bucket, ./11-in-memory`)

- Database is an organised collection of data
- Golang process can hold data in its memory stack
- In-memory database can be used to hold internal state
- Easily replaceable with use of interface

Race condition (`./12-mutex`)

- When the outcome depends on the timing of uncontrollable events
- Fatal error is thrown when multiple goroutines try to modify a map
- `go run -race` detects possible race conditions
- `sync.Mutex` provides a locking mechanism to allow us to control the flow

Race conditions - others

- A builtin `sync.Map` can replace your own usage of mutex
  - Has limited applicability (see video)
- `sync.Mutex` can be replaced with `channels`
  - Unbuffered channels will automatically block when multiple values are being sent through it

Bolt (`./13-bolt`)

- `go get github.com/coreos/bbolt`
- A simple, fast, and reliable database for projects that don't require a full database server
- Stable API and file format
- Supports transactions and batches

Homework

- Implement in memory storage following the `./08-bucket` interface in your calculator
- Add support for following scenario:
  - `#1:  1 +  1 -> 2`
  - `#2: 18 - 12 -> 6`
  - `#3: $1 + $2 -> 8`