What’s context package ?

sivchari

context package

contextパッケージの中で定義されているContext型は、デッドライン、キャンセルシグナル、その他のリクエストに応じた値をAPI境界やプロセス間で伝達します

FYI

context

ex

```go
func main() {
    h1 := func(w http.ResponseWriter, _ *http.Request) {
        io.WriteString(w, "ping")
    }
    h2 := func(w http.ResponseWriter, _ *http.Request) {
        io.WriteString(w, "ping2")
    }

    http.HandleFunc("/ping1", h1)
    http.HandleFunc("/ping2", h2)

    // サーバー起動
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

ListenAndServeはリクエストごとにgoroutineを生成する

実践的に考えるとそれぞれの goroutineがさらにgoroutineを生成する可能性がある
(DB, 外部APIへのリクエスト)

-> 意識せず多くのgoroutineを生成している可能性がある

-> goroutineは勝手に消滅する？

-> goroutine leak

goroutine leak

```go
package main

import (
    "fmt"
    "runtime"
    "time"
)

func main() {
    fmt.Printf("before leak:\t%d\n", runtime.NumGoroutine())

    leak(nil)

    time.Sleep(3 * time.Second)
    fmt.Printf("after leak:\t%d\n", runtime.NumGoroutine())
}

func leak(c <-chan string) {
    go func() {
        for cc := range c {
            fmt.Println(cc)
        }
    }()

    fmt.Printf("in leak:\t%d\n", runtime.NumGoroutine())
}
```

out put

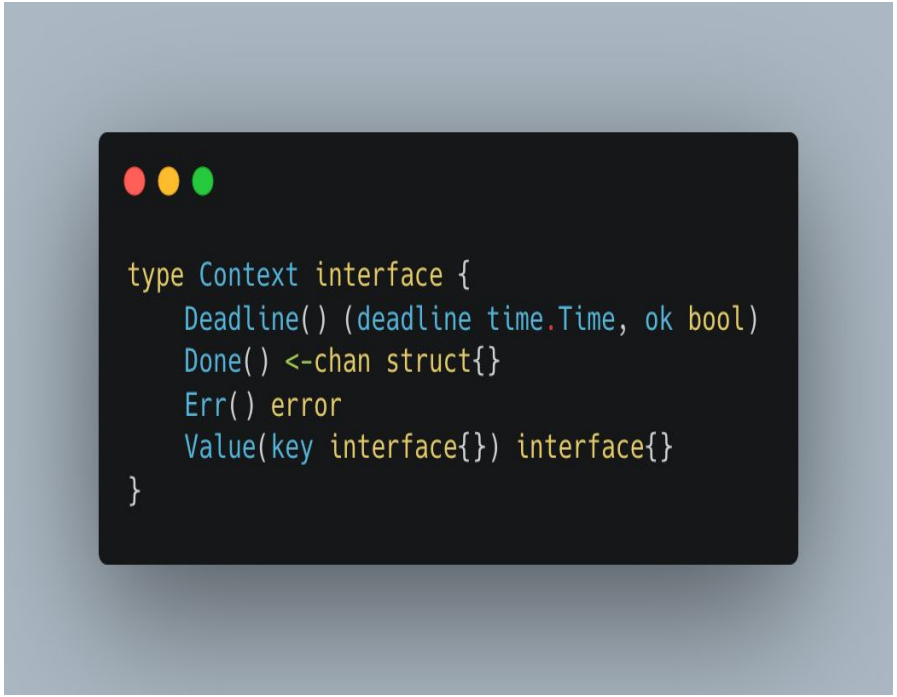
1 before
2 in leak
2 after leak

この場合はnil chanをしっかりmakeしてcloseまでする必要がある
nilでもchannelはcompile timeでerrorを発見できない(可能性としてはruntime error)

problem

- goroutine leakがより複雑になり10個などになった時
- タイムアウトした時
- 使っているライブラリでgoroutineが生成されていたら？
- ある関数はcloseしたいが、ある関数は継続させたい処理にどう対応する？
    - チャネルを複数用意する方法もあるがチャネルの生成にはコストがかかる

context



```
type Context interface {
    Deadline() (deadline time.Time, ok bool)
    Done() <-chan struct{}
    Err() error
    Value(key interface{}) interface{}
}
```

context.Contextはinterface
BackgroundやWithCancelなどユースケースによる実装をinterfaceで抽象化できる

しない場合
context.WithCancelなど限定的な型が必要

Background

contextの生成にはBackgroundを使う

内部実装はContext interfaceを満たすprivate pointer intが生成される

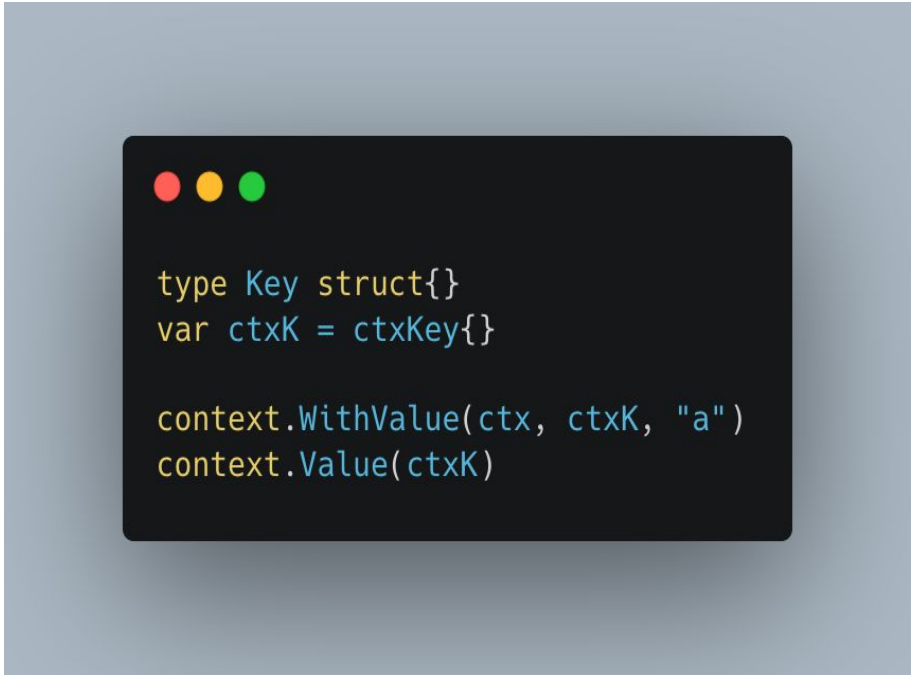
contextを使うか微妙な場合はcontext.TODOを引数に取る

contextの関係

- contextが親子関係にある
- 親のcontextのキャンセルは親から派生したcontextも同じ挙動をとる

- contextが兄弟関係にある(parentからctx1, ctx2が生成される)
- 親がキャンセルされるとどちらも終了する
- 言い換えると親は子の影響を受けない
- ある関数はcloseしたいが、ある関数は継続させたい処理にどう対応する？という問題が解決される

Request Scope of context



```
type Key struct{}
var ctxK = ctxKey{}

context.WithValue(ctx, ctxK, "a")
context.Value(ctxK)
```

Keyを指定してcontextにセットする
同じcontextに対してkeyでsetした値を取得する

type structの理由は一意性を型として担保するため

ex)
別packageごとに同じkeyをセットした場合を考える
(hogeとfugaで同じ文字列をkeyにする)

パッケージに閉じた(privateな)keyを使うようにする

Q. 閉じていればintなどでもいいのでは？
- 空のstructはメモリサイズが0

FYI
https://go.dev/play/p/TYZV5GPrIWm

Don’t use context like this

structの中にcontextを含めない

- contextにはスコープごとの値がsetされるかもしれない(JWT, Token etc..)
- structに含めるとどこで参照されるかわからない

実際にBad PracticeとGo Teamが言っている

Use context Values only for request-scoped data that transits processes and

APIs, not for passing optional parameters to functions.

context

-> 第一引数で渡すようにする

Why?

- スコープがinterface(API)として明確になる

- 例外はnet/http
- 後方互換性(contextは1.7から登場した)
- database/sqlはcontext対応のメソッドを生やした

FYI

https://go.dev/blog/context-and-structs

https://github.com/golang/go/blob/ecf6b52b7f4ba6e8c98f25adf9e83773fe908829/src/net/http/request.go#L319-L323

まとめ

- 自分のライブラリを作る場合はcontextを渡すようなinterfaceを実装する

- クライアント(使う側)から明示的にストップさせる以外はcontextを使う方がいい
  (streamなどはchannelを使う方がいい)

- 逆説としてchannelを使うときはgoroutine leakが起きないか注意する