image

Go 103

Pallat Anchaleechamaikorn

yod.pallat@gmail.com
https://github.com/pallat
https://dev.to/pallat

https://go.dev/tour (Thai)
https://github.com/uber-go/guide (Thai)

defer

A defer statement defers the execution of a function until the surrounding function returns.

The deferred call's arguments are evaluated immediately, but the function call is not executed until the surrounding function returns.

```go
res, _ := http.DefaultClient.Do(req)

defer res.Body.Close()
body, _ := ioutil.ReadAll(res.Body)
```

a defer

```go
defer fmt.Println("end")

fmt.Println("Hello, Gophers")
```

defer is stack

```go
defer fmt.Println("defer first")
defer fmt.Println("defer second")
defer fmt.Println("defer third")

fmt.Println("Hello, Gophers")
```

Questions? What is the result

```go
func doSomething(n int) {
    defer fmt.Println(n)
    fmt.Println(n)
}

func main() {
    doSomething(4)
}
```

Questions? One more

```go
func doSomething(n int) {
    defer fmt.Println(n)
    defer func() {
        fmt.Println(n)
    }()
    n = n * n
    fmt.Println(n)
}

func main() {
    doSomething(4)
}
```

defer for recovering

```go
func catchMe() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println(r)
        }
    }()

    s := []int{}

    fmt.Println(s[1])
}
```

image

defer: exercise

> log name and age ไม่ว่าจะ return ด้วยเหตุผลอะไรก็ตาม

```go
func API(name string, age int) error {
    if name != "Peter" {
        return errors.New("not acceptable")
    }
    if name == "" {
        name = "Michael"
    }
    if age < 60 {
        return errors.New("you are too young")
    }

    return nil
}
```

image

Composition with Struct Embedding

```go
type Card struct {
    HolderName string
    IssuedAt  time.Time
    ExpiredAt time.Time
}

type DrivingCard struct {
    DriverLicense   string
    Class           string
    Card
}

type IDCard struct {
    ID      string
    Address string
    Card
}
```

Composition (2)

```go
type Card struct {
    HolderName string
    IssuedAt  time.Time
    ExpiredAt time.Time
}

func (c Card) IsExpire() bool {
    return time.Now().After(c.ExpiredAt)
}

type DrivingCard struct {
    Card
}

dc := DrivingCard{}
dc.IsExpire()
```

Composition (3)

```
type ReadWriter interface {
    Reader
    Writer
}
```

Generic

```go
func min(x, y float64) float64 {
  if x < y {
    return x
  }
  return y
}
```

image

Generic: type parameter

```go
func min[T constraints.Ordered](x, y T) T {
  if x < y {
    return x
  }
  return y
}
```

> instantiation

```go
m := min[int](2, 3)
fmin := min[float64]
m := fmin(2.1, 2.0)
```

image

Parameter Type

```go
type Tree[T interface{}] struct {
  left, right *Tree[T]
  data        T
}

func (t *Tree[T]) Lookup(x T) *Tree[T]

var stringTree Tree[string]
```

image

Type constraint

```
interface {
  int|string|bool
}
```

```
package constraints

type Ordered interface {
  Integer|Float|~string
}
```

image

![image]

Play with Generic

```
func main() {
    a, b := "1", "2"
    // a, b := 1, 2
    // a, b := 1.0, 2.0
    c := add(a, b)
    fmt.Println(c)
}
```

![image]

First-Class Function

```
var add = func(a, b int) int {
    return a + b
}

fmt.Println(add(1, 2))
```

image

Test FCF

```go
func lampda(a int) int {
    x := 0
    for i:= 1; i <=a; i++ {
        x+=i
    }
    return x
}

func main() {
    fn := lampda
    fmt.Println(fn(3))
}
```

image

image

Higher-Order Function

```go
func hof(fn func(string) string) {
    ...
}

func hof() func(string) string {
    ...
}
```

image

HOF Example

```go
func operation(patient string, doctor func(string) bool) {
    if done := doctor(patient); done {
        fmt.Printf("patient %s is safe\n", patient)
    } else {
        fmt.Printf("we're sorry for %s\n", patient)
    }
}

func main() {
    operation("too", func(name string) bool {
        if name == "too" {
            return false
        }
        return true
    })
}
```

Function type

```go
type doctorFunc func(string) bool

func operation(patient string, doctor doctorFunc) {
    if done := doctor(patient); done {
        fmt.Printf("patient %s is safe\n", patient)
    } else {
        fmt.Printf("we're sorry for %s\n", patient)
    }
}

func main() {
    operation("Peter", func(name string) bool {
        if name == "Peter" {
            return false
        }
        return true
    })
}
```

Test HOF

```go
type pFunc func() int

func factory(a, b pFunc) pFunc {
    return func() int {
        return a() + b()
    }
}

func one() int { return 1 }
func Peter() int { return one() + one() }

func main() {
    fmt.Println(factory(one, Peter)())
}
```

Closure Function

```go
func newFunc() func() int {
    a := 0
    return func() int {
        a++
        return a
    }
}
```

Test Closure Function

```go
func main() {
    fn1, fn2 := factory()
    fn1()
    fn1()
    fmt.Println(fn2())

    fn1()
    fmt.Println(fn2())
}

func factory() (func(), func() int) {
    var i int
    return func() {
            i++
        },
        func() int {
            return i
        }
}
```

image

method on function

```
type IntnFunc func(int) int

func (fn IntnFunc) Intn(n int) int {
    return fn(n)
}
```

image

image

goroutine

```go
func main() {
    total := 10
    now := time.Now()
    for i := 0; i < total; i++ {
        go printout(i)
    }
    fmt.Println(time.Now().Sub(now))
}

func printout(i int) {
    fmt.Println(i)
}
```

image

image

goroutine waiting

```go
var wg = sync.WaitGroup{}

func main() {
    total := 10
    wg.Add(total)
    now := time.Now()
    for i := 0; i < total; i++ {
        go printout(i)
    }
    wg.Wait()
    fmt.Println(time.Now().Sub(now))
}

func printout(i int) {
    fmt.Println(i)
    wg.Done()
}
```

image

Excercise - Count down 1 min

2 goroutine

1. print `+` every 1 sec
2. print `-` every 5 sec

program end after 1 minute pass

channel

keyword `chan`

- no buffered channel
- buffered channel

buffered channel simple

```go
func main() {
    ch := make(chan int, 2)
    ch <- 1
    ch <- 2
    fmt.Println(<-ch)
    fmt.Println(<-ch)
}
```

image

buffered channel

```go
total := 10
ch := make(chan int, total)
for i := total; i > 0; i-- {
    ch <- i
}
close(ch)

for i := range ch {
    fmt.Println(i)
}
```

image

image

buffered channel range

```
func fibonacci(n int, c chan int) {
 x, y := 0, 1
 for i := 0; i < n; i++ {
  c <- x
  x, y = y, x+y
 }
 close(c)
}

func main() {
 c := make(chan int, 10)
 go fibonacci(cap(c), c)
 for i := range c {
  fmt.Println(i)
 }
}
```

image

image

no buffered channel

```
func main() {
    total := 10
    ch := make(chan struct{})
    now := time.Now()
    for i := 0; i < total; i++ {
        go printout(i, ch)
    }
    for i := 0; i < total; i++ {
        <-ch
    }
    fmt.Println(time.Now().Sub(now))
}

func printout(i int, ch chan struct{}) {
    fmt.Println(i)
    ch <- struct{}{}
}
```

image

Goroutine exercise

```go
import "github.com/pallat/force"

force.Decrypt
force.Validate()
```

brute force all encrypted

select

```
select {
    case <- ch1:
    case <- ch2:
}
```

select example

```
func fibonacci(c, quit chan int) {
 x, y := 0, 1
 for {
  select {
  case c <- x:
   x, y = y, x+y
  case <-quit:
   fmt.Println("quit")
   return
  }
 }
}

func main() {
 c := make(chan int)
 quit := make(chan int)
 go func() {
  for i := 0; i < 10; i++ {
   fmt.Println(<-c)
  }
  quit <- 0
 }()
 fibonacci(c, quit)
}
```

Keywords: 22/25

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

break

```go
i := 0

for {
    if i > 100 {
        break
    }
    i++
}
```

continue

```go
s := []int{1, 4, 5, 3, -2, 0, 8, -1}
for i, v := range s {
    if v < 0 {
        continue
    }
    fmt.Printf("index: %d, value: %d\n", i, v)
}
```

goto

```go
func main() {
    check("First")
}

func check(class string) {
    if class == "First" {
        goto welcome
    }
    fmt.Println("Please show me the passport")

welcome:
    fmt.Println("Welcome")
}
```

Keywords: 25/25

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

built-in function

```
new         make            append          delete
len         cap             recover         panic
```

time

net/http