Goを嫌いにならないためのメンタルモデル

2024/06/28 Yamanaka Junichi

自己紹介

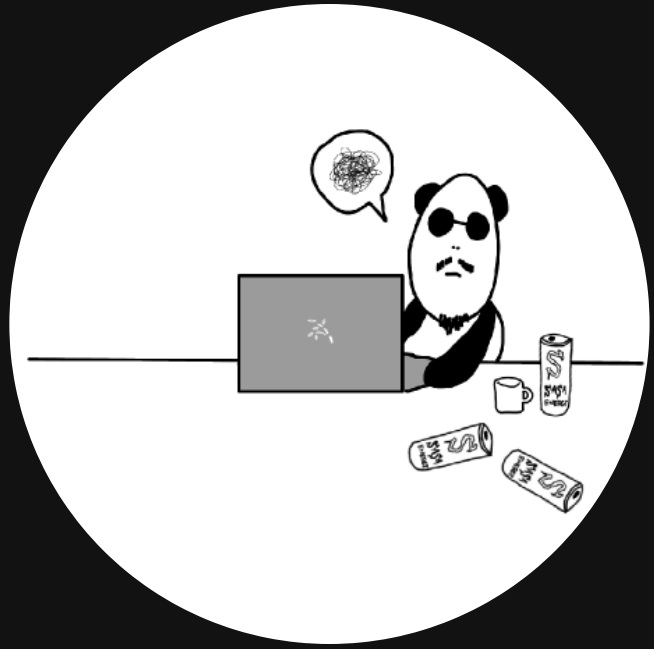
Yamanaka Junichi

経歴

- 2013年 ~ 2020年　Jcomで営業
- 2020年 ~ 2022年 Javaで受託開発
- 2022年 ~ 2024年 Coconeでブロックチェーンゲームの開発をKotlinやGoで
- 2024年 ~ フリーランス

好きな技術

Go, Protobuf, Unit Test



今日話したいこと

- 🚀 Goを好きになるには今までの言語からのパラダイムシフトが必要(かもしれない)
- 🥑 Goの言語仕様が少ないことには理由が必ずある
- 🐢 Goは独自の文化やルールを強制しているわけではない

Goが合わないとならない!!とならないようにこれらのことを知っておきましょう😇

Java脳でGoを書くとどうなるか

オブジェクトのカプセル化ができなくて悩む

Java

```java
public class User {

    private String name;
    private int age;

    // コンストラクタ
    public User(String name, int age) {
        this.name  = name;
        this.age = age;
    }

    // getter
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

Go

```go
package user

type User struct {
  name string
  age  int
}

func NewUser(name string, age int) User {
  return User{name: name, age: age}
}

func (u User) Name() string {
  return u.name
}

func (u User) Age() int {
  return u.age
}

// 同じパッケージからだと
// プライベートなフィールドにもアクセスできてしまう
```

nullチェックがしたくなる

Java

```java
public class Item {
    private final String id;

    private Item(String id) {
        this.id = id;
    }

    public static Item createItem(String id) {
        if (id.startsWith("d-")) {
            return null;
        }
        return new Item(id);
    }
}

// Main.java
var item = Item.createItem("d-1111");
if(item == null) {
    // 何か処理
}
```

Go

```go
func NewItem(id string) (Item, error) {
  if strings.HasPrefix(id, "d-") {
    return Item{}, fmt.Errorf("specific item not starting `
  }
  return Item{Id: id}, nil
}

func main() {
  item, err := NewItem("d-111")
  if err ≠ nil {
    panic(err)
  }
}
```

ここまでのまとめ

Goはオブジェクト指向の言語ではない

- 👉 Goはもともと**システムプログラミングを書く言語**

Goでは変数に値があるかは気にしなくてもいい

- 👉 ポインタ以外はデフォルト値が使われるためpanicしない
- 👉 ポインタ型はnilになり得るがerrorを返すことでnil参照が起こらない
- 👉 Goではnillチェックの代わりにerrorチェックをする

Goにないもの

配列・List操作

Kotlin

```kotlin
data class Student(val math: Int, val english: Int)

fun main(args: Array<String>) {
  val students = listOf(
  Student(math = 70, english = 81),
  Student(math = 70, english = 81),
  Student(math = 70, english = 81),
  )

    students.filter { it.math > 70 }
        .filter { it.english > 80 }
        .map { it.math + it.english }
        .forEach { println("total: $it") }

}
```

Go

```go
type Student struct {
  math    int
  english int
}

func main() {
  students := []Student{
    {math: 71, english: 81},
    {math: 71, english: 81},
    {math: 71, english: 81},
  }

  for _, s := range students {
    if s.math ⩽ 70 || s.english ⩽ 80 {
      continue
    }
    sum := s.math + s.english
    fmt.Printf("total: %d\n", sum)
```

enum

Kotlin

```kotlin
enum class Status{
    Success,
    Failed
    ;
}

fun main() {
    // code 0 name Success
    // code 1 name Failed
    for (status in Status.entries) {
        println("code ${status.ordinal} name ${status.name}
    }
}
```

Go

```go
type status int
const (
  _ status = iota
  success
  failed
)

func main() {
  // 1 2
  fmt.Printf("%v %v", success, failed)
}
```

他の言語にあるようなenumの機能が欲しければstringerやenumerというモジュールもある

ここまでのまとめ

Goにはなぜmapやfilterがないのか

- 👉 遅くなるから
- 👉 Goは他の言語にあるからといってパフォーマンスを落とすような機能を取り入れない

Goにはなぜenumがないのか

- 👉 Goにある既存の機能で十分enumとしての機能を提供することができる
- 👉 enumがもたらす複雑さに比べて得られるものが少ない

Goはベストよりもベターを採用する言語😇

Goの短い変数名について

Go

```go
func TestExample(t *testing.T) {
  tests := map[string]struct {
    x        int
    y        int
    expected int
  }{
    "1 + 1 = 2": {
      x:        1,
      y:        1,
      expected: 2,
    },
    "0 + (-1) = -1": {
      x:        0,
      y:        -1,
      expected: -1,
    },
  }

  for name, tt := range tests {
    name, tt := name, tt
    t.Run(name, func(t *testing.T) {
      if tt.x+tt.y ≠ tt.expected {
```

```
        t.Errorf("expected %d but %d", tt.expected, tt.x+tt.y)
      }
    })
  }
}
```

パッケージ名を使った命名

syncパッケージの例

```
var once sync.Once

func Hello(ch chan struct{}) {
  time.Sleep(time.Second)
  ch←struct{}{}
  close(ch)
}

func main() {
  ch := make(chan struct{})
  defer func ()  {
    once.Do(func() {
      close(ch)
    })
  }()

  go Hello(ch)

  ←ch
}
```

まとめ

Goはシステムプログラミングのための言語

- 👉 Goが合わないと思ったのなら今までの言語のメンタルモデルを一度壊しましょう😇

Goはベストよりもベターを選択する言語

- 👉 なぜ？と思うものにはだいたい理由があるので調べてみるといいです🔍

Goは命名についてどの言語よりも真剣に考えている

- 👉 Goの短い命名は独自の文化でもルールでもないです

END

参考

- Goのなぜ問答