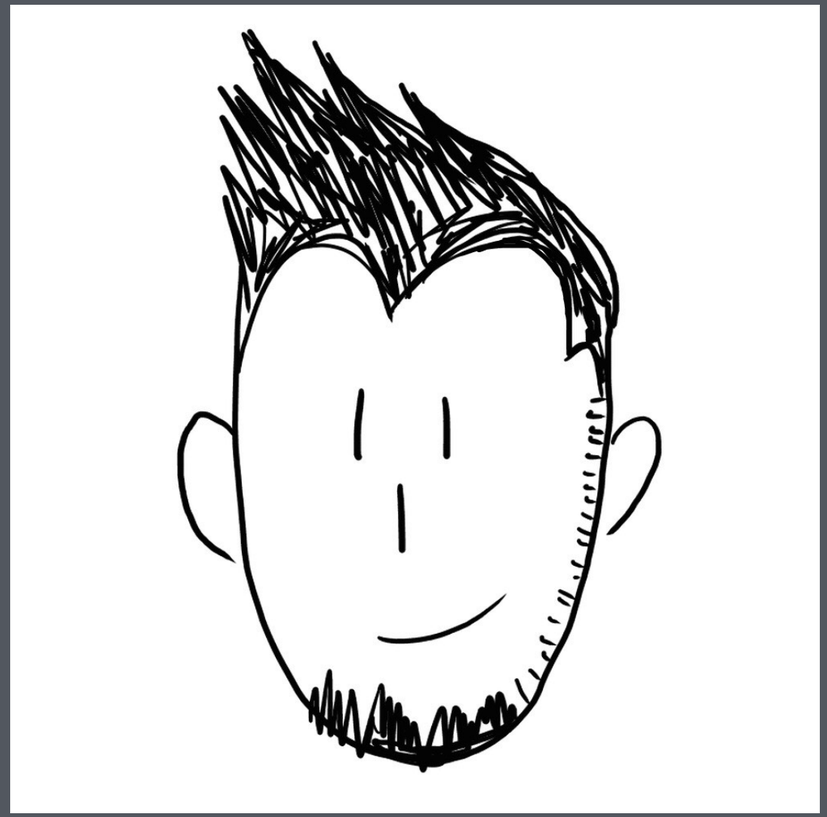
DIコンテナを使わないDI

golang.tokyo#11 2017/12/11

自己紹介

- Name: 森國 泰平 (Morikuni Taihei)
- GitHub: @morikuni
- Twitter: @inukirom
- 所属
  - 株式会社メルカリ/ソウゾウ
  - メルカリ カウル



本日の内容

❌ DIできるようにコードを書く方法

⭕ DIコンテナを使わずに依存関係を解決する方法

どうやって依存関係を解決する？

- mainに書く
- DIコンテナを使う
- DI用の関数を定義する

例題

以下の3つの関数を使ってServiceのインスタンスを作れ。

```
func NewService(repo Repository, mailer Mailer) Service { ... }

func NewRepository(db *sql.DB) Repository { ... }

func NewMailer() Mailer { ... }
```

ServiceはRepositoryとMailerに依存している。

Repositoryは*sql.DBに依存している。

mainに書く

```
func main() {
    db, err := sql.Open("db", "dsn")
    if err != nil { ... }
    repo := NewRepository(db)
    mailer := NewMailer()
    service := NewService(repo, mailer)
}
```

オブジェクトが増える度にmainが肥大化していき可読性が低い

DIコンテナを使う (goldi[*1])

- yamlに依存関係を記述する
- DIコンテナ用のコードを生成する
- DIコンテナから必要なインスタンスを取得する

DIコンテナの使い方を覚える必要がある

[*1] 実際にgoldiを使ったことはないので間違っている可能性があります

```yaml
types:
    db:
        package: database/sql
        type: *DB
        factory: Open
        arguments:
            - "db"
            - "dsn"
    repository:
        package: github.com/morikuni/hoge
        type: Repository
        factory: NewRepository
        arguments:
            - "@db"
    mailer:
        package: github.com/morikuni/hoge
        type: Mailer
        factory: NewMailer
    service:
        package: github.com/morikuni/hoge
        type: Service
        factory: NewService
        arguments:
            - "@repository"
            - "@mailer"
```

DI用の関数を定義する(Inject関数)

- オブジェクトが引数0個で取得できるようにする関数
- あるオブジェクトについて、依存先が引数0個で取得できればそのオブジェクトも引数0個で取得できる
- Inject関数を組み合わせることでInject関数を作る

Inject関数の実装

InjectDBは*sql.DBが引数0個で取得できるようにする

```
func InjectDB() *sql.DB {
    db, err := sql.Open("db", "dsn")
    if err != nil {
        panic(err)
    }
    return db
}
```

Inject関数の実装

*sql.DBが引数0個で取得できるのでRepositoryも引数0個で取得できる

```
func InjectRepository() Repository {
    return NewRepository(
        InjectDB(),
    )
}

func InjectMailer() Mailer {
    return NewMailer()
}
```

Inject関数の実装

RepositoryとMailerが引数0個で取得できるので、Serviceも引数0個で取得できるようになる

```
func InjectService() Service {
    return NewService(
        InjectRepository(),
        InjectMailer(),
    )
}
```

Inject関数を使う利点

- 依存先が増減したとしても影響範囲はInject関数内に収まる
- 実装が書かれるpackageが変わっても影響範囲はInject関数内に収まる
- Goのコードで書けるので構文を新しく覚える必要がない

詳しくはWebで！

明日(12/12)のQiita Advent Calender Go4に

もう少し詳しい記事を書きます！

https://qiita.com/advent-calendar/2017/go4