Please visit website: http://cxyroad.com

Go项目结构整洁实现 | GitHub 3.5k

一、前言

hi，大家好，这里是白泽。今天给大家分享一个GitHub 3.5k 的 Go项目：go-backend-clean-arch

[github.com/amitshekhar…](http://cxyroad.com/"https://github.com/amitshekhariitbhu/go-backend-clean-architecture")

这个项目是一位老外写的，通过一个 HTTP demo 介绍了一个优雅的项目结构。

我也在b站出了一期30多分钟的视频，讲解了这个仓库，欢迎你的 B站：白泽talk，qq交流群：622383022。

`![image-20240401202825006](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/7d5beb04b3ca442cac47837b59157232~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1816&h=362&s=582318&e=png&b=f9f6f6)`

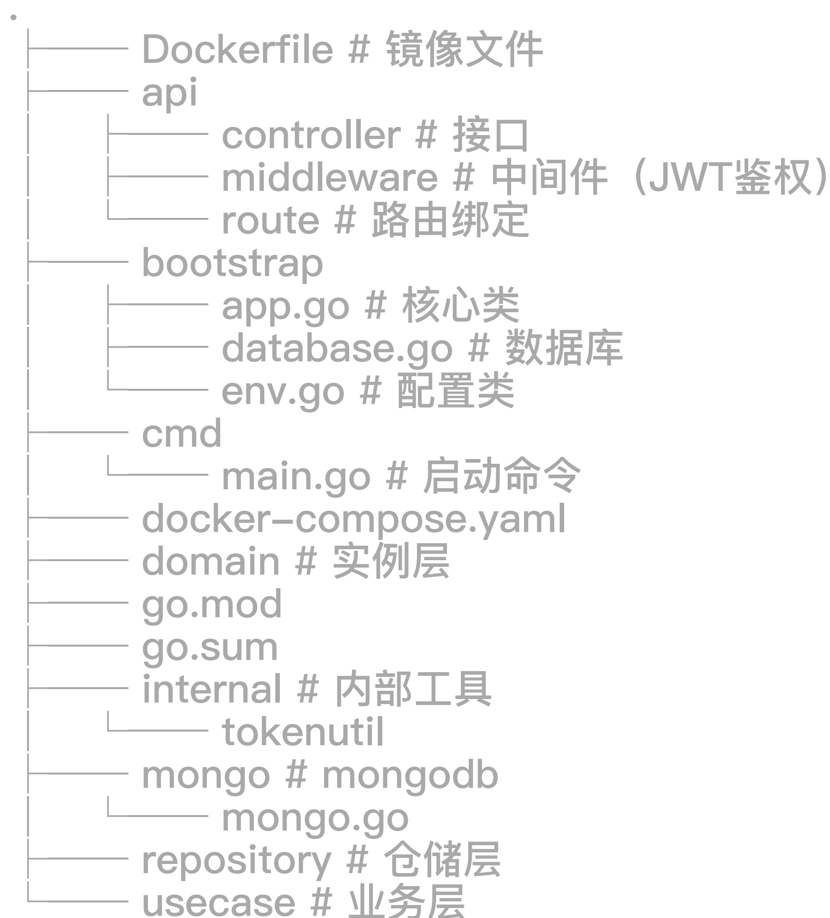
> 当然，如果您是一位 Go 学习的新手，您可以在我开源的学习仓库：[github.com/BaiZe1998/g…](http://cxyroad.com/"https://github.com/BaiZe1998/go-learning") 中，找到我往期翻译的英文书籍，或者Go学习路线。

`![image-20240401222006030](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/48c4bfdbd0fb457ebfd154b9fc6d8a1b~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=3820&h=1732&s=572143&e=png&b=ffffff)`

二、项目架构

```
![image-20240401202921385](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/76e1d593da404baa9cbfe361ca8fefbe~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1550&h=832&s=167676&e=png&b=fdfdfd)
```

三、目录详解



```
.
├── Dockerfile # 镜像文件
├── api
│   ├── controller # 接口
│   ├── middleware # 中间件（JWT鉴权）
│   └── route # 路由绑定
├── bootstrap
│   ├── app.go # 核心类
│   ├── database.go # 数据库
│   └── env.go # 配置类
├── cmd
│   └── main.go # 启动命令
├── docker-compose.yaml
├── domain # 实例层
├── go.mod
├── go.sum
├── internal # 内部工具
│   └── tokenutil
├── mongo # mongodb
│   └── mongo.go
├── repository # 仓储层
└── usecase # 业务层
```

3.1 参数配置 & 项目启动

./cmd/main.go

```
```

```
type Application struct {
    Env   *Env
    Mongo mongo.Client
}

type Env struct {
    AppEnv                string `mapstructure:"APP_ENV"`
    ServerAddress         string `mapstructure:"SERVER_ADDRESS"`
    ContextTimeout        int    `mapstructure:"CONTEXT_TIMEOUT"`
    DBHost                string `mapstructure:"DB_HOST"`
    DBPort                string `mapstructure:"DB_PORT"`
    ...
}

func main() {
    // app 是整个应用的实例，管理生命周期中的重要资源
    app := bootstrap.App()
    // 配置变量
    env := app.Env
    // 数据库实例
    db := app.Mongo.Database(env.DBName)
    defer app.CloseDBConnection()

    timeout := time.Duration(env.ContextTimeout) * time.Second
    // gin 实例创建
    gin := gin.Default()
    // 路由绑定
    route.Setup(env, timeout, db, gin)
    // 运行服务
    gin.Run(env.ServerAddress)
}
```

接下来的讲解将以登陆逻辑为例，讲解三层架构。

3.2 接口层

./api/controller/login_controller.go

LoginController 持有配置类，以及 LoginUsecase 接口（定义了业务层的行为）

```
// 业务层接口
type SignupUsecase interface {
    Create(c context.Context, user *User) error
    GetUserByEmail(c context.Context, email string) (User, error)
    CreateAccessToken(user *User, secret string, expiry int)
(accessToken string, err error)
    CreateRefreshToken(user *User, secret string, expiry int)
(refreshToken string, err error)
}

type LoginController struct {
   LoginUsecase domain.LoginUsecase
   Env          *bootstrap.Env
}

func (lc *LoginController) Login(c *gin.Context) {
   var request domain.LoginRequest

   err := c.ShouldBind(&request)
   if err != nil {
      c.JSON(http.StatusBadRequest, domain.ErrorResponse{Message:
err.Error()})
      return
   }

   user, err := lc.LoginUsecase.GetUserByEmail(c, request.Email)
   if err != nil {
      c.JSON(http.StatusNotFound, domain.ErrorResponse{Message:
”User not found with the given email”})
      return
   }

   if bcrypt.CompareHashAndPassword([]byte(user.Password),
[]byte(request.Password)) != nil {
      c.JSON(http.StatusUnauthorized, domain.ErrorResponse{Message:
”Invalid credentials”})
      return
   }

   accessToken, err := lc.LoginUsecase.CreateAccessToken(&user,
lc.Env.AccessTokenSecret, lc.Env.AccessTokenExpiryHour)
   if err != nil {
      c.JSON(http.StatusInternalServerError,
domain.ErrorResponse{Message: err.Error()})
      return
   }
```

```
    refreshToken, err := lc.LoginUsecase.CreateRefreshToken(&user,
lc.Env.RefreshTokenSecret, lc.Env.RefreshTokenExpiryHour)
    if err != nil {
        c.JSON(http.StatusInternalServerError,
domain.ErrorResponse{Message: err.Error()})
        return
    }

    loginResponse := domain.LoginResponse{
        AccessToken:  accessToken,
        RefreshToken: refreshToken,
    }

    c.JSON(http.StatusOK, loginResponse)
}
```

3.3 业务层

./usecase/login_usecase.go

loginUsecase 结构实现 LoginUsecase 接口，同时在 loginUsecase 结构中，持有了 UserRepository 接口（定义了仓储层的行为）。

```
// 数据防腐层接口
type UserRepository interface {
    Create(c context.Context, user *User) error
    Fetch(c context.Context) ([]User, error)
    GetByEmail(c context.Context, email string) (User, error)
    GetByID(c context.Context, id string) (User, error)
}

type loginUsecase struct {
    userRepository domain.UserRepository
    contextTimeout time.Duration
}

func NewLoginUsecase(userRepository domain.UserRepository, timeout
time.Duration) domain.LoginUsecase {
    return &loginUsecase{
        userRepository: userRepository,
        contextTimeout: timeout,
```

```
    }
}

func (lu *loginUsecase) GetUserByEmail(c context.Context, email string) (domain.User, error) {
   ctx, cancel := context.WithTimeout(c, lu.contextTimeout)
   defer cancel()
   return lu.userRepository.GetByEmail(ctx, email)
}

func (lu *loginUsecase) CreateAccessToken(user *domain.User, secret string, expiry int) (accessToken string, err error) {
   return tokenutil.CreateAccessToken(user, secret, expiry)
}

func (lu *loginUsecase) CreateRefreshToken(user *domain.User, secret string, expiry int) (refreshToken string, err error) {
   return tokenutil.CreateRefreshToken(user, secret, expiry)
}

```

3.4 防腐层

./repository/user_repository.go

userRepository 结构实现了 UserRepository 接口，内部持有 mongo.Database 接口（定义数据层行为），以及 collection 实例的名称。

```
// 数据操作层接口
type Database interface {
    Collection(string) Collection
    Client() Client
}

type userRepository struct {
   database   mongo.Database
   collection string
}

func NewUserRepository(db mongo.Database, collection string) domain.UserRepository {
   return &userRepository{
```

```
      database:   db,
      collection: collection,
   }
}

func (ur *userRepository) Create(c context.Context, user *domain.User)
error {
   collection := ur.database.Collection(ur.collection)

   _, err := collection.InsertOne(c, user)

   return err
}

func (ur *userRepository) Fetch(c context.Context) ([]domain.User, error)
{
   collection := ur.database.Collection(ur.collection)

   opts := options.Find().SetProjection(bson.D{{Key: ”password”, Value:
0}})
   cursor, err := collection.Find(c, bson.D{}, opts)

   if err != nil {
      return nil, err
   }

   var users []domain.User

   err = cursor.All(c, &users)
   if users == nil {
      return []domain.User{}, err
   }

   return users, err
}

func (ur *userRepository) GetByEmail(c context.Context, email string)
(domain.User, error) {
   collection := ur.database.Collection(ur.collection)
   var user domain.User
   err := collection.FindOne(c, bson.M{”email”: email}).Decode(&user)
   return user, err
}

func (ur *userRepository) GetByID(c context.Context, id string)
(domain.User, error) {
   collection := ur.database.Collection(ur.collection)
```

```
    var user domain.User

    idHex, err := primitive.ObjectIDFromHex(id)
    if err != nil {
        return user, err
    }

    err = collection.FindOne(c, bson.M{"_id": idHex}).Decode(&user)
    return user, err
}
```

3.5 数据层

./mongo/mongo.go

实现了 mongo.Database 接口，通过 mongoDatabase 结构体的两个方法可以获取对应的 Client 实例和 Collection 实例，从而操作数据库。

```
type mongoDatabase struct {
    db *mongo.Database
}

func (md *mongoDatabase) Collection(colName string) Collection {
     collection := md.db.Collection(colName)
     return &mongoCollection{coll: collection}
}

func (md *mongoDatabase) Client() Client {
     client := md.db.Client()
     return &mongoClient{cl: client}
}
```

四、单例与封装

查看 ./cmd/main.go 的路由绑定逻辑：route.Setup(env, timeout, db, gin)。

```
func Setup(env *bootstrap.Env, timeout time.Duration, db mongo.Database, gin *gin.Engine) {
   publicRouter := gin.Group(””)
   // All Public APIs
   NewSignupRouter(env, timeout, db, publicRouter)
   NewLoginRouter(env, timeout, db, publicRouter)
   NewRefreshTokenRouter(env, timeout, db, publicRouter)

   protectedRouter := gin.Group(””)
   // Middleware to verify AccessToken

protectedRouter.Use(middleware.JwtAuthMiddleware(env.AccessTokenSecret))
   // All Private APIs
   NewProfileRouter(env, timeout, db, protectedRouter)
   NewTaskRouter(env, timeout, db, protectedRouter)
}
```

进一步查看 NewLoginRouter，会发现，在注册路由触发的 controller 方法的时候，已经将所需要的 db 创建出来，并且在数据层共享，同时防腐层、业务层、控制层的实例，在服务启动前创建，依次嵌套持有，因此所有的结构都是单例，且类似树形结构，依次串联。

```
func NewLoginRouter(env *bootstrap.Env, timeout time.Duration, db mongo.Database, group *gin.RouterGroup) {
   ur := repository.NewUserRepository(db, domain.CollectionUser)
   lc := &controller.LoginController{
      LoginUsecase: usecase.NewLoginUsecase(ur, timeout),
      Env:          env,
   }
   group.POST(”/login”, lc.Login)
}
```

通过这种方式，实现了资源的约束，使得开发者无法跨模块调用实例，导致循环依赖等安全问题。

原文链接: https://juejin.cn/post/7352789840352755721