

数据库大作业报告

学号：2022300004054


参与部分：后端的架构设计，接口设计，接口文档编写，后端实现。

成果：完成接口文档的设计。完成后端大部分代码的编写。


项目地址：


GitHub - Aliancn/LibSystem

Contribute to Aliancn/LibSystem development by creating an account on GitHub.


 <https://github.com/aliancn/libsystem>

Aliancn/**LibSystem**




 2


Contributors

 0


Issues

 2

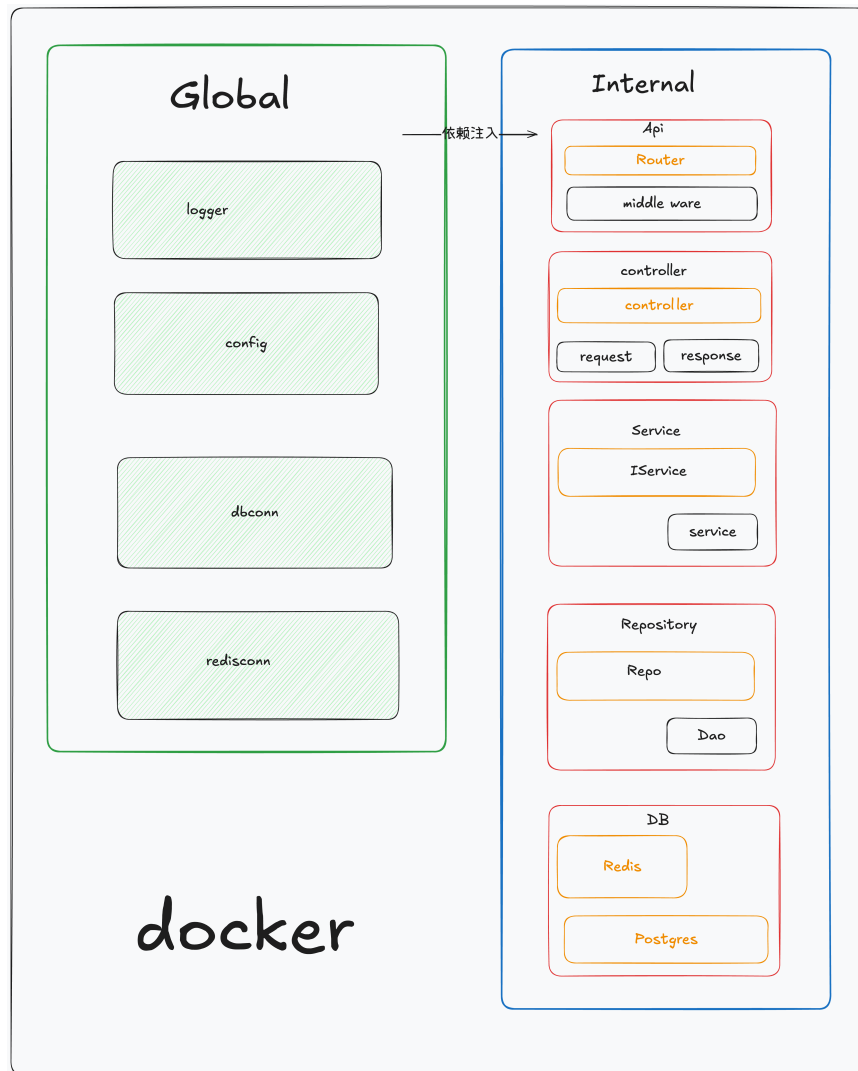
Stars

 0

Forks



项目架构：



项目结构

```

.
├── common
│   ├── enum.go
│   └── utils
│       ├── encrypt.go
│       ├── generice.go
│       ├── jwt.go
│       └── oss.go
├── config
│   ├── application-dev.yaml
│   ├── application-release.yaml
│   └── config.go
└── data

```

```

|   |   ├── postgres
|   |   |   ├── config
|   |   |   ├── data
|   |   |   └── logs
|   |   ├── redis
|   |   |   ├── conf
|   |   |   ├── data
|   |   |   └── logs
|   |   ├── running
|   |   |   └── logs
|   |   └── uploads
|   ├── docker
|   |   └── etcd
|   |       └── dockerfile
|   ├── docker-compose.distribute.yaml
|   ├── docker-compose.yaml
|   ├── dockerfile
|   ├── global
|   |   ├── global.go
|   |   └── tx
|   |       ├── gormTx.go
|   |       └── transactionManager.go
|   ├── go.mod
|   ├── go.sum
|   ├── initialize
|   |   ├── enter.go
|   |   ├── gorm.go
|   |   ├── redis.go
|   |   ├── router.go
|   |   └── table_create.go
|   ├── internal
|   |   ├── api
|   |   |   ├── controller
|   |   |   ├── request
|   |   |   └── response
|   |   ├── model
|   |   |   ├── Book.go
|   |   |   └── Borrow.go

```

```
| | | └─ Info.go
| | | └─ Paper.go
| | | └─ User.go
| | | └─ file.go
| | | └─ model.go
| | └─ repository
| | | └─ book_repo.go
| | | └─ borrow_repo.go
| | | └─ dao
| | | └─ info_repo.go
| | | └─ paper_repo.go
| | | └─ user_repo.go
| | └─ router
| | | └─ admin
| | | └─ common
| | | └─ router.go
| | | └─ user
| └─ service
| | └─ book.go
| | └─ borrow.go
| | └─ info.go
| | └─ paper.go
| | └─ user.go
└─ logger
| | └─ log.go
| | └─ slog.go
└─ main.go
└─ middle
| | └─ jwt_middle.go
└─ readme.md
└─ script
| | └─ db.sql
| | └─ run.bash
| | └─ run.sh
└─ uploads
| | └─ papers
```

设计方案：

- 遵循传统的MVC架构设计。
- 遵循传统SOLID设计原则，遵循接口隔离和依赖倒置的设计方法。
- 采用多级缓存。

完成接口

根目录（接口 29 个）

输入关键字进行搜索

Q

⚙️ ↗️

<input type="checkbox"/> 接口名称	请求类型 ▾	接口路径	接口分组 : ▾	接口状态 : ▾	标签
<input type="checkbox"/> get all	GET	/admin/user	根目录/admin/user	● 开发中	-
<input type="checkbox"/> getbyid	GET	/admin/user/:id	根目录/admin/user	● 开发中	-
<input type="checkbox"/> getbyname	GET	/admin/user/userna...	根目录/admin/user	● 开发中	-
<input type="checkbox"/> adduser	POST	/admin/user	根目录/admin/user	● 开发中	-
<input type="checkbox"/> editpassword	PUT	/admin/user/editPa...	根目录/admin/user	● 开发中	-
<input type="checkbox"/> updateuser	PUT	/admin/user	根目录/admin/user	● 开发中	-
<input type="checkbox"/> deleteuser	DELETE	/admin/user	根目录/admin/user	● 开发中	-
<input type="checkbox"/> edit paper	PUT	/admin/papers	根目录/admin/paper	● 开发中	-
<input type="checkbox"/> delete paper	DELETE	/admin/papers/{id}	根目录/admin/paper	● 开发中	-
<input type="checkbox"/> add book	POST	/admin/books	根目录/admin/book	● 开发中	-
<input type="checkbox"/> delete book	DELETE	/admin/books/{id}	根目录/admin/book	● 开发中	-
<input type="checkbox"/> edit book	PUT	/admin/books	根目录/admin/book	● 开发中	-
<input type="checkbox"/> delete log	DELETE	/admin/borrows/{id}	根目录/admin/borrow	● 开发中	-
<input type="checkbox"/> get all	GET	/admin/borrows	根目录/admin/borrow	● 开发中	-

根目录 (接口 29 个)						输入关键字进行搜索	Q	⚙
<input type="checkbox"/> 接口名称	请求类型	接口路径	接口分组	接口状态	标签			
<input type="checkbox"/> download paper	GET	/papers/download/{...	根目录/user/paper	• 开发中	-			
<input type="checkbox"/> get by id	GET	/papers/{id}	根目录/user/paper	• 开发中	-			
<input type="checkbox"/> get all papers	GET	/papers	根目录/user/paper	• 开发中	-			
<input type="checkbox"/> search by title	GET	/papers/title	根目录/user/paper	• 开发中	-			
<input type="checkbox"/> get all	GET	/books	根目录/user/book	• 开发中	-			
<input type="checkbox"/> get by id	GET	/books/{id}	根目录/user/book	• 开发中	-			
<input type="checkbox"/> search by title	GET	/books/title	根目录/user/book	• 开发中	-			
<input type="checkbox"/> borrow book	POST	/borrows	根目录/user/borrow	• 开发中	-			
<input type="checkbox"/> return book	PUT	/borrows/{id}	根目录/user/borrow	• 开发中	-			
<input type="checkbox"/> 获得用户借书情况	GET	/borrows	根目录/user/borrow	• 开发中	-			
<input type="checkbox"/> login	POST	/login	根目录/common	• 开发中	-			
<input type="checkbox"/> logout	POST	/logout	根目录/common	• 开发中	-			
<input type="checkbox"/> register	POST	/register	根目录/common	• 开发中	-			
<input type="checkbox"/> 获得统计信息	GET	/info	根目录/common	• 开发中	-			

代码说明

对部分代码按照项目的文件结构进行简单的说明。

common

本文件夹包含通用代码，主要包含通用工具函数和全局常量。

常量具体包含错误码定义、状态码定义、枚举量定义等。

工具函数主要包含加密函数、Jwt生成和效验工具

config

本文件夹主要包含配置文件和config结构体。

global

本文件夹主要包含全局依赖

```

var (
    Config *config.AllConfig // 全局Config
    Log     logger.ILog // 全局日志器
    DB      *gorm.DB // db连接
    Redis   *redis.Client // redis连接
)

```

middle

本文件夹主要包含Jwt中间件实现

```

func VerifyJWT() gin.HandlerFunc {
    return func(c *gin.Context) {
        code := common.SUCCESS
        token := c.Request.Header.Get(global.Config.Jwt.Name)
        // 解析获取用户载荷信息
        token = token[7:]
        payLoad, err := utils.ParseJwtToken(global.Config.Jwt
        if err != nil {
            code = common.UNKNOW_IDENTITY
            global.Log.Error("jwt parse error: ", err)
            c.JSON(http.StatusUnauthorized, common.Result{Cod
            c.Abort()
            return
        }
        // 在上下文设置载荷信息
        c.Set(common.CurrentID, payLoad["uid"])
        c.Set(common.CurrentName, payLoad["role"])

        // 这里是否要通知客户端重新保存新的Token
        c.Next()
    }
}

```

本Jwt中间件使用gin框架实现。用于在用户发出请求后对请求中的令牌解析，并将解析出的信息添加到上下文中，为api层提供便捷的参数获取方法。

internal

本文件夹包含项目的核心业务逻辑

api 文件夹中包含Restful API接口的具体实现，通过对请求参数的结构体和答复参数的结构的定义，规范的进行请求的参数解析和回复封装。

示例如下：

```
type BookController struct {
    service service.IBookService
}

func NewBookController(bookService service.IBookService) *BookController {
    return &BookController{service: bookService}
}
```

service 文件夹中主要包含业务的主要逻辑处理代码。通过对 **interface**（go语言中的抽象类型）的设计，完成抽象接口的设计，符合依赖倒置原则。

示例如下：

```
type IBookService interface {
    GetBookList(ctx *gin.Context, pageID, pageSize int) (response.BookVO, error)
    GetBookById(ctx *gin.Context, id int) (response.BookVO, error)
    GetBookByTitle(ctx *gin.Context, title string) (response.BookVO, error)
    AddBook(ctx *gin.Context, add request.BookDTO) error
    UpdateBook(ctx *gin.Context, update request.BookDTO) error
    DeleteBook(ctx *gin.Context, id int) error
}

type BookService struct {
    repo repository.BookRepo
}
```


repository 文件夹中主要包含Model层代码。用于充当业务层与数据库之间的桥梁。在repo文件中定义对应数据库对象需要操作的 **interface** 接口，并在 **dao** 文件中具体实现，符合依赖倒置的原则。

示例如下：

```
type BookRepo interface {
    GetAll(ctx context.Context, pageID, pageSize int) ([]model.Book, error)
    GetByID(ctx context.Context, id int) (model.Book, error)
    GetByTitle(ctx context.Context, title string) ([]model.Book, error)
    Create(ctx context.Context, book model.Book) error
    Update(ctx context.Context, book model.Book) error
    Delete(ctx context.Context, id int) error
    GetNum(ctx context.Context) (int, error)
}

type BookDao struct {
    db *gorm.DB
}
```

router 文件夹包含所有的路由定义。用于实现路由的身份管理和效验规则的定义。同时为路由对应的处理器注入数据库依赖，完成依赖注入。

示例如下：

```
type UBookRouter struct {
}

func (br *UBookRouter) InitApiRouter(router *gin.RouterGroup) {
    bookCtl := controller.NewBookController(service.NewBookService())
    publicBookRouter := router.Group("/books")
    privateBookRouter := router.Group("/books")
    privateBookRouter.Use(middleware.VerifyJWT())
    {
        publicBookRouter.GET("", bookCtl.GetBookList)
        privateBookRouter.GET("/:id", bookCtl.GetBookById)
        privateBookRouter.GET("/title", bookCtl.GetBookByTitle)
    }
}
```

```
}  
}
```