

Tencent 腾讯

GO DDD领域驱动设计架构思考与实践

daheige
腾讯应用开发工程师

目录

- 01 自我介绍
- 02 架构设计相关理论
- 03 领域驱动设计相关理论
- 04 GO DDD领域驱动设计实践
- 05 总结 & 收益

资深GO开发者（布道者）、资深PHP架构师、资深Nodejs开发者、Rust语言布道者

工作经历：

- 2014-2015 sass邮件系统和电商平台 PHP+Nodejs
- 2016-2017 金融平台和游戏平台 PHP+Nodejs
- 2018年 加入腾讯 sass应用开发 GO+Rust

主要工作：

GO微服务架构设计、接口性能优化、DDD领域驱动设计

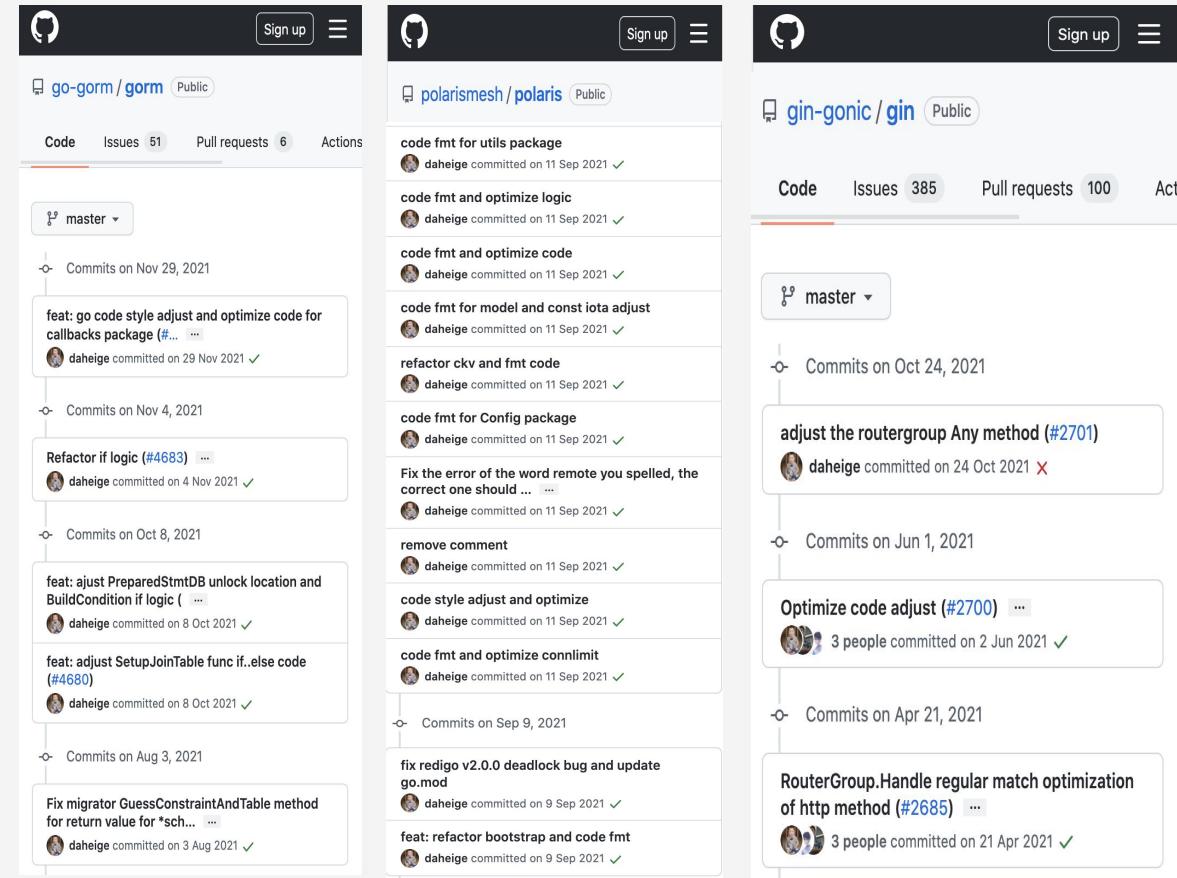
- 负责团队GO技术栈引入，基础组件把关，GO BFF设计
- 推动团队GO微服务架构设计和领域驱动设计落地
- 制定团队GO代码规范和落地，并形成良好的CR/MR习惯



专业影响力：

- 腾讯代码规范OTeam GO代码规范小组成员之一
- 腾讯北极星polaris开源项目GO/Rust语言核心committer
- 开源项目go gin框架代码贡献者
- 开源项目go-gorm/gorm框架代码贡献者
- 开源项目cloudwego/volo rust微服务框架代码贡献者
- 开源项目go-god/broker committer
- 开源项目rs-god/rs-infras rcron committer

个人主页：github.com/daheige



什么是架构设计？

Robert C. Martin 罗伯特·C·马丁在《[架构整洁之道](#)》提到过：

- 软件架构是系统设计过程中的重要设计决定的集合，可以通过变更成本来衡量每个设计决定的重要程度。
- 所谓架构：就是“[用最小的人力成本来满足构建和维护系统需求](#)”的设计行为。
- 架构设计的本质：[识别并解决软件复杂度](#)。

如何识别架构优劣：

- ✓ 一个软件架构的优劣，可以用它满足用户需求所用的成本来衡量。
- ✓ 一个好的架构需满足[易开发、易理解、易维护、可拓展、轻松部署和快速发布](#)。

软件复杂度主要包含4部分：

- ◆ [高并发场景下的对软件高性能的要求](#)
- ◆ [复杂多变的业务场景，对软件高可用的要求](#)
- ◆ [持续变化的业务以及业务扩张和需求变更，对拓展性的要求](#)
- ◆ [软件开发的人力成本、时间成本、安全、软件规模等，对软件维护和拓展的要求](#)



解决软件复杂度的方法：

高性能：

多进程、多线程、多协程编程、异步处理

计算和存储分离，关注点分离（es搜索，mysql数据持久化，kibana日志搜索）

缓存（redis/内存缓存）降低数据库和后台的负载，扛并发

消息队列MQ（kafka、pulsar、rbmq）对业务进行解耦和削峰填谷

高可用：

服务器集群分布式处理、多数据中心，支持垂直和水平拓展

同城双活（多活）、跨地复制、异地容灾，保证服务高可用

redis哨兵/集群/主从、mysql读写分离和主从机制，保证数据高可用

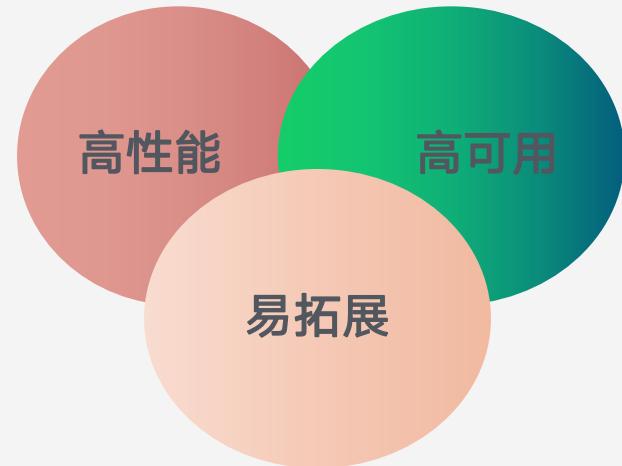
完善的服务监控系统、性能分析、日志服务、trace链路追踪服务

可拓展性和可维护性：

划分边界：分层设计、模块化设计，使得组件之间相互独立，具有边界感。--《软件架构：架构模式 特征及实践指南》

策略与层次：面向接口编程，低层策略指向高层策略，提升代码的易读性、可拓展性、复用性。--《架构整洁之道》

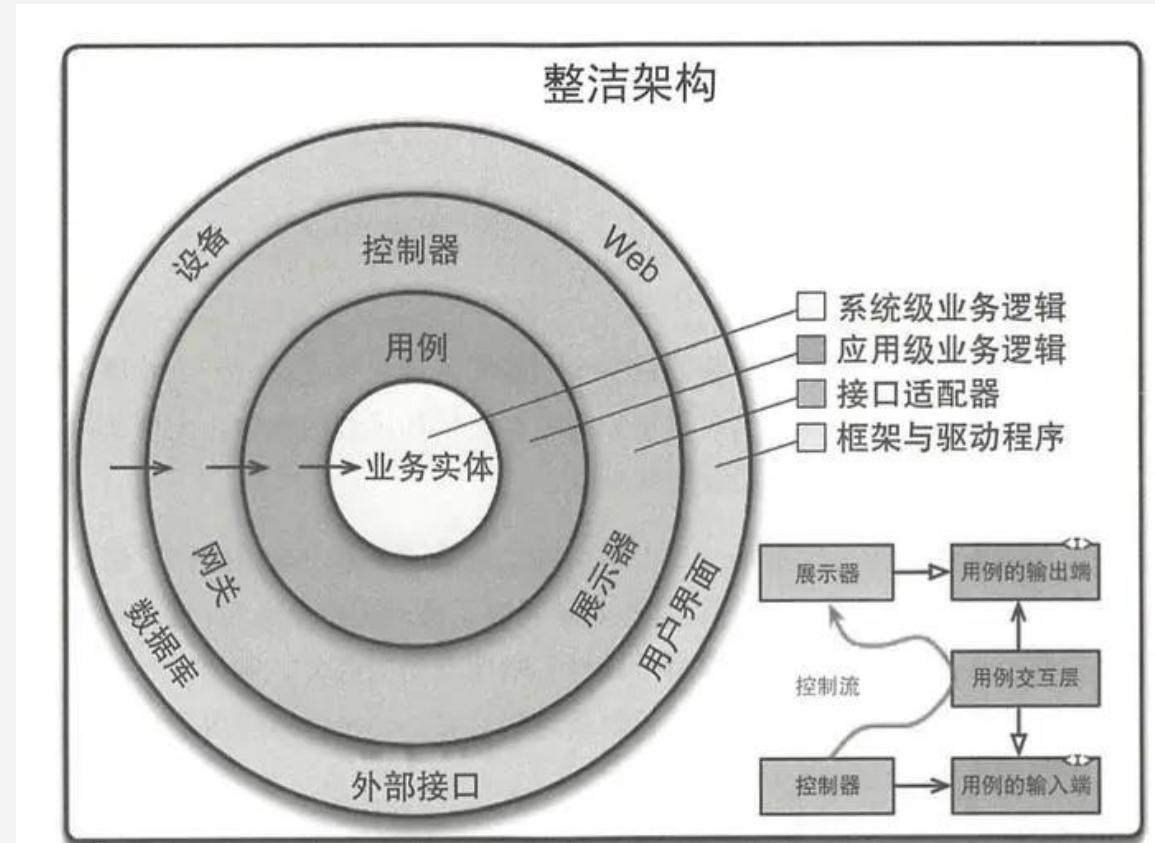
设计原则：一言概括就是最小kiss原则，简单优于复杂，做一件事情并做好它。--《unix编程艺术》



2 架构设计相关理论

整洁架构：按照不同关注点对软件服务进行切割。

- ✓ 框架、web层、数据库、用户界面等都是实现细节
- ✓ 一个系统的架构应该着重于展示系统本身的设计，而并非该系统所使用的框架
- ✓ 架构可被测试，这些系统的业务逻辑可以脱离UI、数据库
- ✓ 用例包含各种特定场景下的业务逻辑，这一层变动不应影响内层业务实体，也不应受外层细节影响而变动，内外都需要进行隔离
- ✓ 一个业务实体既可以是一个带有方法的对象，也可以是一组数据结构和函数的集合
- ✓ 最内层的圆中包含的是最通用、最高层的策略，最外层的圆包含的是最具体的实现细节
- ✓ 源码中的依赖关系必须只指向同心圆的内层，即由低层机制指向高层策略
- ✓ 对于外部细节，Rob推荐把utils, DB, UI, web框架等统统放到外层



3

DDD领域驱动设计相关理论

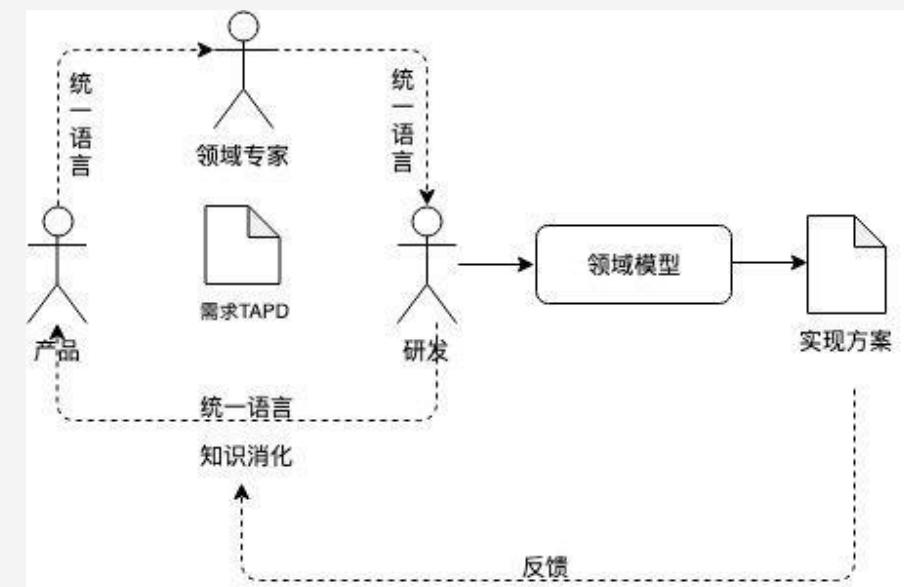
什么是DDD?

DDD是一种处理高度复杂领域的设计思想，试图分离技术实现的复杂性，同时围绕业务概念构建领域模型，提出的一种软件架构设计的方法论。

DDD核心思想：通过领域驱动设计方法，定义领域模型，从而界定业务和应用的领域边界，保证业务模型于代码模型一致。

DDD主要由2部分组成：

- 战略设计：重业务建模。从业务视角出发，分析问题，拆分领域，通过事件风暴去识别并建立业务领域模型。根据领域实体间的业务关联形成聚合，并在各个聚合之间建立起边界。根据业务和语义边界，将一个或数个聚合分配在不同的限界上下文中。它涉及到的人员：领域专家、开发人员、项目经理、产品人员、测试、架构师等多个角色。
- 战术设计：重落地。从技术视角出发，侧重于领域模型的技术实现，完成软件开发和落地生根，包括聚合根、实体、值对象、领域服务、应用服务和资源库等代码逻辑的设计和实现。它涉及到的人员：架构师、开发人员等等多个角色。



3

DDD领域驱动设计相关理论

DDD名称简述：

- ◆ 领域、子域、核心域、通用域、支撑域

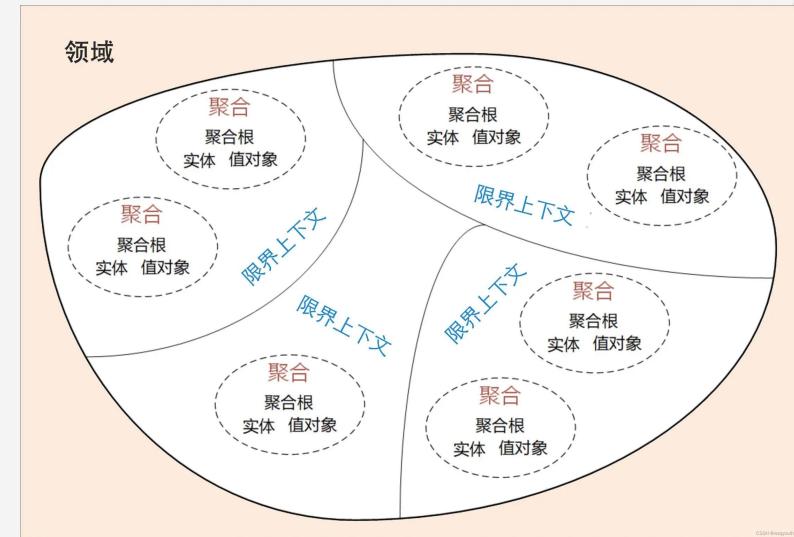
领域与子域用来划分业务域范围

核心域是划分服务核心关注点

通用域是多个领域通用范畴，被多个子域使用的通用功能子域

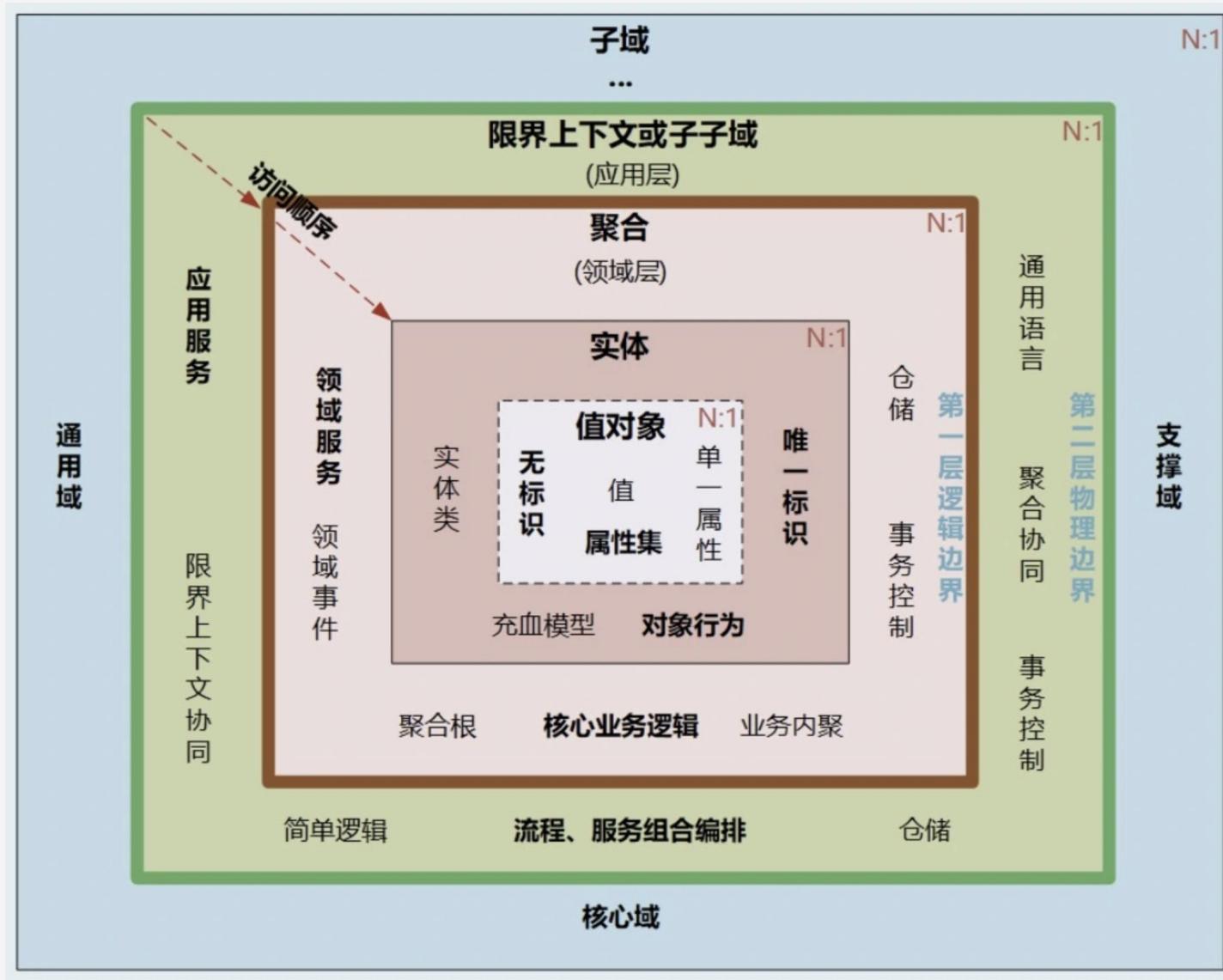
支撑域是对领域起支撑范畴

- ◆ 限定上下文：区分领域边界，确保每个上下文含义在它特定的边界内都具有唯一的含义
- ◆ 实体：领域内容的核心成员，领域模型中的实体是多个属性、操作或行为的载体
- ◆ 值对象：属性集合，值对象能降低实体数量，对实体状态和状态进行描述（比如用户地址、邮箱、电话等值对象打包，放入订单物流单中），是一个没有标识符的对象
- ◆ 聚合：单个实体的聚合采用充血模式（业务逻辑都在一个类或包内实现），将实体和实体相关的业务包裹起来，实现业务逻辑的高内聚操作
- ◆ 领域服务：跨多个实体的不同业务操作，通过对领域的领域服务进行服务编排和组合。比如订单业务，涉及到用户信息，商品信息，订单物流信息等，就需要跨多个领域进行编排和组合业务逻辑
- ◆ 聚合根：聚合内多个实体对外以统一的标识ID对外提供服务



3

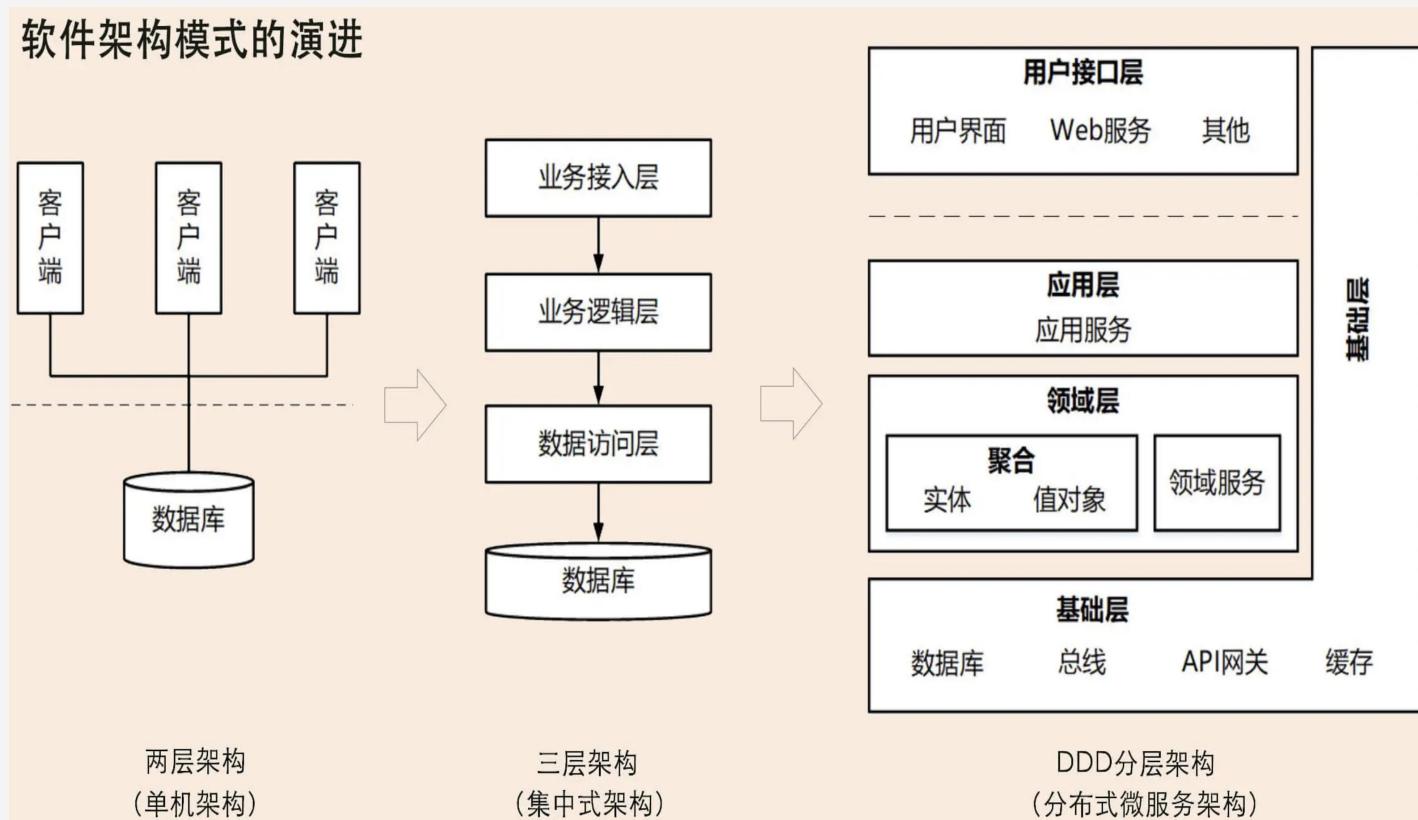
DDD领域驱动设计相关理论



3 DDD领域驱动设计相关理论

DDD与微服务关系：

- ◆ DDD是一种架构设计的方法论，而微服务是一种架构思想和架构设计实现，两者本质都是追求对业务变化的快速响应，分别从不同的业务视角去解决软件的复杂度。
- ◆ DDD主要从业务领域角度划分服务边界，构建通用语言进行高效沟通，抽象业务，建立模型，维持业务和技术实现代码一致性。
- ◆ 微服务主要从技术出发，关注服务运行时服务通信，编码/解码、容错、故障隔离、服务发现与注册、服务治理、服务开发和部署、服务监控、devops CI/CD等关键内容。
- ◆ DDD方法论，可以更好地指导微服务落地生根，让服务具有清晰的边界感。

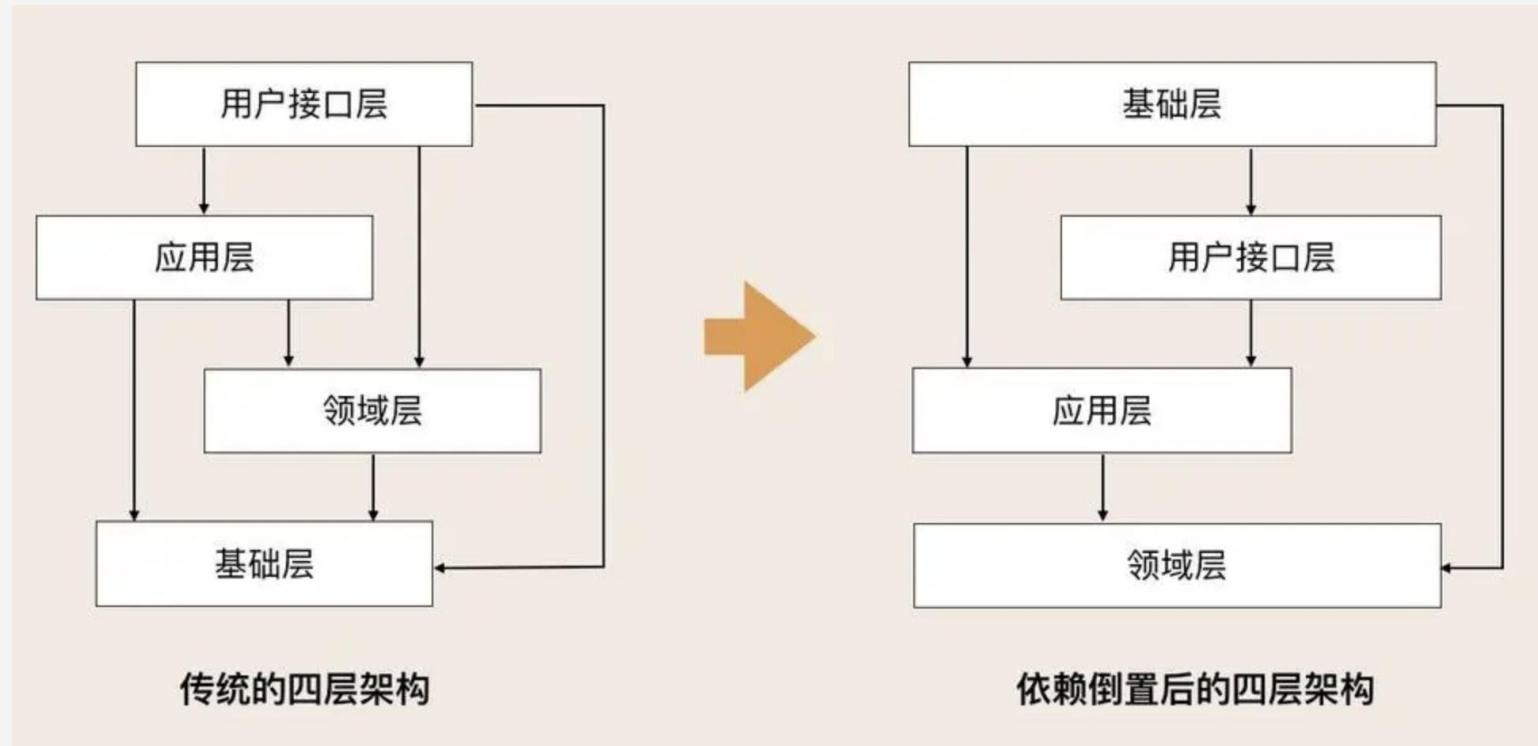


4

GO DDD领域驱动设计实践

DDD分层架构+面向接口编程：

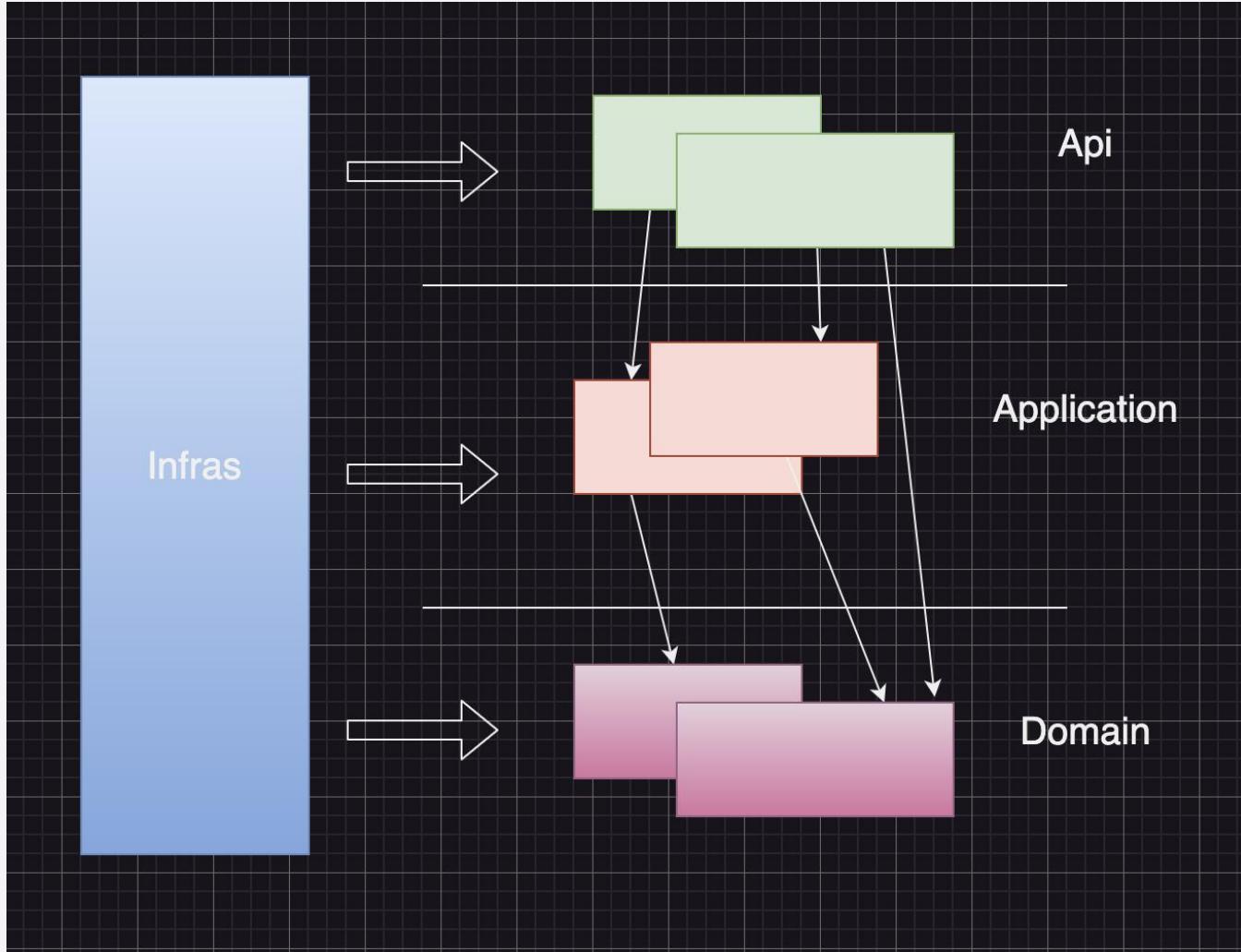
- ◆ DDD分层架构实际还是分而治之，化繁为简的思想，将复杂的大问题拆分为小问题解决，以及将复杂大领域简化为多个子领域服务设计，再通过业务编排和组合解决问题。
- ◆ 分层架构设计本质：**关注点分离**，层与层之间具有清晰的边界感与职责分明，相互独立
- ◆ 通过**依赖倒置**的设计思想，可以把业务领域聚焦起来，通过接口设计暴露领域服务能力
- ◆ **面向接口编程，底层策略指向高层策略**，用户接口层对应用层依赖、应用层对领域层依赖、基础设施层对其他各层依赖



4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/duheige/go-ddd-api>



DDD分层架构设计：

领域层Domain: 负责news新增、查询、删除、更新等操作，实体需要通过聚合，对外提供能力给到应用层使用，同时通过仓库

Repository接口提供实体持久化存储

基础设施层Infras: 包含配置文件读取、数据migration、mysql持久化存储功能、项目基础组件等

接口层Api: 负责news列表接口、发布、修改以及topic相关处理

应用层Application: 负责业务逻辑的组合和编排处理

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

分层目录划分：

The screenshot shows the file structure of the `go-ddd-api` project. The `internal` folder contains several sub-folders: `api`, `application`, `domain`, `infras`, and `providers`. The `api` folder is highlighted with a red box and labeled "接口层". The `application` folder is highlighted with a red box and labeled "应用层". The `domain` folder is highlighted with a red box and labeled "领域层". The `infras` and `providers` folders are highlighted with red boxes and labeled "基础设施层" and "依赖注入处理" respectively. The `main.go` file is highlighted with a blue box and labeled "入口文件".

```
1 package main
2
3 import (
4     "log"
5
6     "github.com/daheige/go-ddd-api/internal/api"
7     "github.com/daheige/go-ddd-api/internal/providers"
8 )
9
10 func main() {
11     var app api.NewsHandler
12     // dependency injection
13     if err := providers.Inject(&app); err != nil {
14         log.Fatalf("provier error:%s\n", err)
15     }
16
17     // app service run
18     app.Run()
19 }
20
```



以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

Design

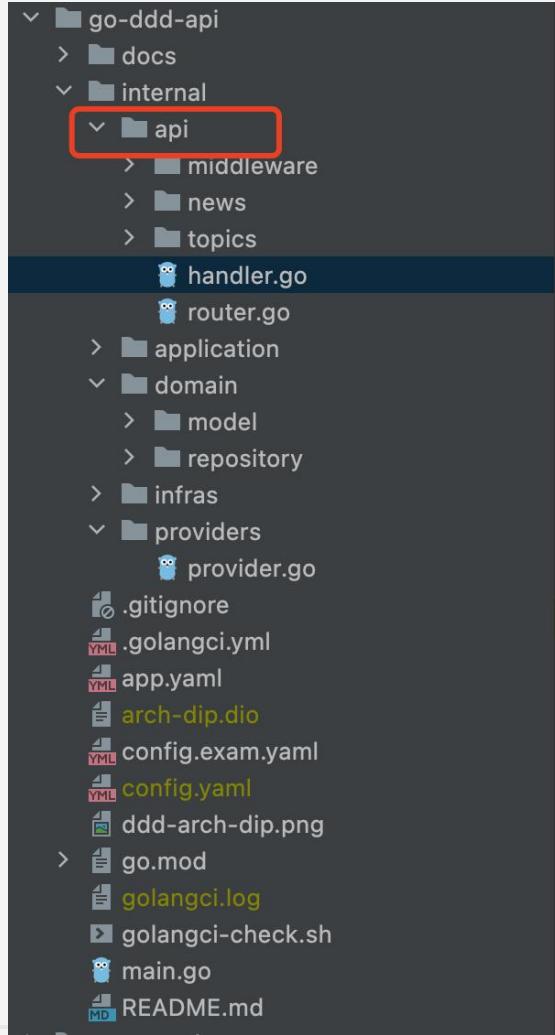
- Application
 - Write business logic
 - news.go (GetNews, GetAllNews, &...)
 - topic.go (GetTopic, GetAllTopic, &...)
- Domain
 - Define interface
 - repository interface for infrastructure
 - Define struct
 - Entity struct that represent mapping to data model
 - news.go
 - topic.go
- Infrastructure
 - Implements repository interface
 - news_repository.go
 - topic_repository.go
- Interfaces
 - HTTP handler

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

接口层api：负责请求的参数接收处理、对外接口服务、中间件、日志记录等功能



The image shows a code editor displaying a Go file. The code defines a `NewsHandler` struct with `AppConfig` and `RouterHandler` fields. It also defines a `RouterHandler` struct with `TopicHandler`, `NewsHandler`, and `MigrateAction` fields. The `Run` method creates a mux router, registers it, and starts an HTTP server. A red arrow points from the `RouterHandler` field in the `NewsHandler` struct to the `RouterHandler` struct definition below it. The code uses annotations like `'inject:'''` for dependency injection.

```
1 package api
2
3 import (
4     "context"
5     "fmt"
6     "log"
7     "net/http"
8     "os"
9     "os/signal"
10    "syscall"
11    "time"
12
13    "github.com/daheige/go-ddd-api/internal/infras/config"
14    "github.com/daheige/go-ddd-api/internal/infras/utils"
15 )
16
17 // NewsHandler service handler
18 type NewsHandler struct {
19     AppConfig *config.AppConfig `inject:'"`
20     RouterHandler *RouterHandler `inject:'"`
21 }
22
23 // Run start services
24 func (s *NewsHandler) Run() {
25     addr := fmt.Sprintf("0.0.0
26 // log.Println("AppConfig:
27 log.Printf("Server running
28
29 // register mux router
30 router := s.RouterHandler.
31
32 // create http services
33 server := &http.Server{
34     // Handler: http.TimeoutHandler(router, time.Second*6, `{code:503,"message":"services timeout"}`),
35     Handler: router,
36     Addr:    addr,
37 }
```

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

应用层application：负责news/topic业务逻辑的编排和组合

```
1 package application
2
3 import ...
4
5 // NewsService news service
6 type NewsService struct {
7     DB *gorm.DB `inject:""`
8     NewsRepo repository.NewsRepository `inject:"-"`
9 }
10
11 // GetNews returns domain.news
12 func (s *NewsService) GetNews(id int) (*model.News, error) {
13     return s.NewsRepo.Get(id)
14 }
15
16 // GetAllNews return all domain.news
17 func (s *NewsService) GetAllNews() ([]model.News, error) {
18     return s.NewsRepo.GetAll()
19 }
20
21 // AddNews saves new news
22 func (s *NewsService) AddNews(p *model.News) error {
23     return s.NewsRepo.Save(p)
24 }
25
26 // RemoveNews do remove news by id
27 func (s *NewsService) RemoveNews(id int) error {
28     return s.NewsRepo.Remove(id)
29 }
30
31 // UpdateNews do remove news by id
32 func (s *NewsService) UpdateNews(p model.News, id int) error {
33     return s.NewsRepo.Update(p, id)
34 }
35
36 // GetAllNewsByFilter return all model.News by filter
37 func (s *NewsService) GetAllNewsByFilter(status string) ([]model.News, error) {
38     return s.NewsRepo.GetAllByStatus(status)
39 }
40
41 // GetNewsByTopic returns []model.News by topic.slug
42 func (s *NewsService) GetNewsByTopic(slug string) ([]model.News, error) {
43     return s.NewsRepo.GetBySlug(slug)
44 }
```

NewsRepository represent repository of the news Expect implementation by the infras layer
'NewsRepository' on pkg.go.dev ↗

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

领域层domain：负责实体定义和repo资源接口设计

```
1 package model
2
3 // News represent entity of the news
4 ↑ type News struct {
5     BaseModel
6     Title string `json:"title"`
7     Slug  string `json:"slug"`
8     Content string `json:"content" gorm:"text"`
9     Status string `json:"status"`
10    Topic []Topic `gorm:"many2many:news_topics;"`  

11 }
12
13 // TableName table name
14 ↑ func (News) TableName() string {
15     return "news"
16 }
```

```
1 package repository
2
3 import (
4     "github.com/daheige/go-ddd-api/internal/domain/model"
5 )
6
7 / NewsRepository represent repository of the news
8 // Expect implementation by the infras layer
9 ↓ type NewsRepository interface {
10     // Get obtain news by id
11     Get(id int) (*model.News, error)
12     // GetAll obtain all news
13     GetAll() ([]model.News, error)
14     // GetBySlug obtain news list by slug
15     GetBySlug(slug string) ([]model.News, error)
16     // GetAllByStatus obtain news by status
17     GetAllByStatus(status string) ([]model.News, error)
18     // Save news save
19     Save(*model.News) error
20     // Remove news remove by id
21     Remove(id int) error
22     // Update news update by entity
23     Update(*model.News) error
24 }
```

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

基础设施层infras：负责news应用的config配置读取、数据库处理、分页组件、项目组件等功能

The image shows a file browser and a code editor side-by-side. The file browser on the left lists the project structure of 'go-ddd-api'. A red box highlights the 'infras' folder under 'domain', which contains 'confia' and 'news_config.go'. Another red box highlights the 'news_repository.go' file under 'persistence'. The code editor on the right displays the content of 'news_repository.go'. A red box highlights the declaration of the NewsRepository interface and its implementation. Another red box highlights several methods: GetAll(), Save(news *model.News), Remove(id int), and Update(news *model.News). A callout box provides a detailed explanation of the NewsRepository interface.

```
1 package persistence
2
3 import ...
4
5 var _ repository.NewsRepository = (*NewsRepositoryImpl)(nil)
6
7 type NewsRepository interface {
8     Get(id int) (*model.News, error)
9     GetAll() ([]model.News, error)
10    GetBySlug(slug string) ([]model.News, error)
11    GetAllByStatus(status string) ([]model.News, error)
12    Save(*model.News) error
13    Remove(id int) error
14    Update(*model.News) error
15 }
16
17 // Get news by id return domain.news
18 func (r *NewsRepositoryImpl) Get(id int) (*model.News, error) {
19     news := &model.News{}
20     if err := r.DB.Preload("news").Where("id = ?", id).First(&news); err != nil {
21         return nil, err
22     }
23     return news, nil
24 }
25
26
27 // GetAll News return all domain.news
28 func (r *NewsRepositoryImpl) GetAll() ([]model.News, error) {
29     var news []model.News
30     r.DB.Preload("news").All(&news)
31     return news, nil
32 }
33
34 // Save to add news
35 func (r *NewsRepositoryImpl) Save(news *model.News) error {
36     if err := r.DB.Create(news).Error; err != nil {
37         return err
38     }
39     return nil
40 }
41
42 // Remove to delete news by id
43 func (r *NewsRepositoryImpl) Remove(id int) error {
44     var news model.News
45     if err := r.DB.Where("id = ?", id).Delete(&news).Error; err != nil {
46         return err
47     }
48     return nil
49 }
50
51 // Update is update news
52 func (r *NewsRepositoryImpl) Update(news *model.News) error {
53     if err := r.DB.Model(news).Update("content", news.Content).Error; err != nil {
54         return err
55     }
56     return nil
57 }
58
59 // GetAll News return all domain.news
60 func (r *NewsRepositoryImpl) GetAllByStatus(status string) ([]model.News, error) {
61     var news []model.News
62     r.DB.Preload("news").Where("status = ?", status).All(&news)
63     return news, nil
64 }
65
66 // GetBySlug News return all []model.News by topic.slug
67 func (r *NewsRepositoryImpl) GetBySlug(slug string) ([]model.News, error) {
68     var news []model.News
69     r.DB.Preload("news").Where("slug = ?", slug).All(&news)
70     return news, nil
71 }
```

NewsRepository represent repository of the news Expect implementation by the infras layer
'NewsRepository' on pkg.go.dev ↗

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

providers: 负责news应用的依赖注入，处理组件的初始化，接口绑定等功能

```

1 package providers
2
3 import ...
16
17 // Inject dependency injection
18 func Inject(app *api.NewsHandler) error {
19     conf := config.NewConfig()
20     di := factory.CreateDI(factory.FbInject) // create a di container
21     err := di.Provide(
22         &gdi.Object{Value: app},           →  创建di容器
23         &gdi.Object{Value: conf.AppConfig()}, // app section inject
24         &gdi.Object{Value: conf.InitDB()},   // db inject
25         &gdi.Object{Value: &migration.MigrateAction{}},
26         &gdi.Object{Value: &news.NewsHandler{}},
27         &gdi.Object{Value: &topics.TopicHandler{}},
28         &gdi.Object{Value: &application.TopicService{}},
29         &gdi.Object{Value: &application.NewsService{}},
30         &gdi.Object{Value: &persistence.NewsRepositoryImpl{}},
31         &gdi.Object{Value: &persistence.TopicRepositoryImpl{}},
32         &gdi.Object{Value: &api.RouterHandler{}},
33     )
34
35     if err != nil {
36         return fmt.Errorf("provide error:%s", err.Error())
37     }
38
39     // invoke object
40     err = di.Invoke()
41     if err != nil {
42         return fmt.Errorf("invoke error:%s", err.Error())
43     }
44
45     return nil
}

```

```

1 package gdi
2
3 // Injector di inject interface
4 type Injector interface {
5     // Provide objects to the Graph. The Object documentation describes
6     // the impact of various fields.
7     Provide(objects ...*Object) error
8
9     // Invoke the incomplete Objects.
10    Invoke(values ...interface{}) error
11
12    // String return inject type name.
13    String() string
14
15
16    // Object an Object in the Graph.
17    type Object struct {
18        Value interface{}
19        Name  string // di name optional
20        Group string // di group name optional
21    }
}

```

github.com/go-god/gdi

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

入口文件main.go：负责news应用的启动、平滑退出等功能

```
1 package main
2
3 import (
4     "log"
5
6     "github.com/daheige/go-ddd-api/internal/api"
7     "github.com/daheige/go-ddd-api/internal/providers"
8 )
9
10 func main() {
11     var app api.NewsHandler
12     // dependency injection
13     if err := providers.Inject(&app); err != nil {
14         log.Fatalf("provider error:%s\n", err)
15     }
16
17     // app service run
18     app.Run()
19 }
```

```
import ...
// NewsHandler service handler
type NewsHandler struct {
    AppConfig *config.AppConfig `inject:""`
    RouterHandler *RouterHandler `inject:""`
}
// Run start services
func (s *NewsHandler) Run() {
    addr := fmt.Sprintf("0.0.0.0:{s.AppConfig.Port}")
    // log.Println("AppConfig: ", s.AppConfig)
    log.Printf("Server running on:#{addr}")

    // register mux router
    router := s.RouterHandler.Router()

    // create http services
    server := &http.Server{
        // Handler: http.TimeoutHandler(router, time.Second*6, '{code:500, Handler: router, Addr: addr, ReadTimeout: 5 * time.Second, WriteTimeout: 10 * time.Second,}')
    }

    // run http services in goroutine
    go func() {
        defer utils.Recover()

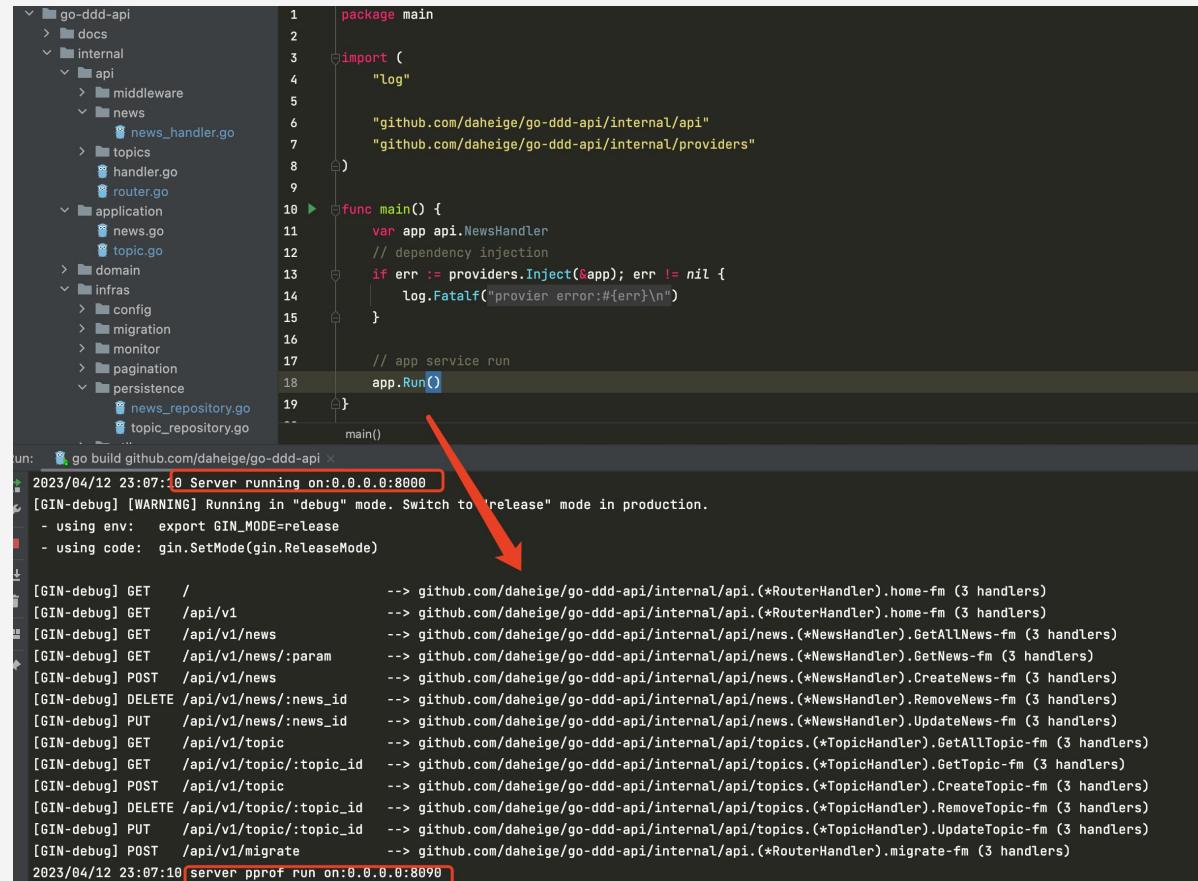
        if err := server.ListenAndServe(); err != nil {
            if err != http.ErrServerClosed {
                log.Println("services listen error:", err)
            }
        }
    }()
}
```

4

GO DDD领域驱动设计实践

以news应用为例子，实现news发布和管理 <https://github.com/daheige/go-ddd-api>

运行效果：

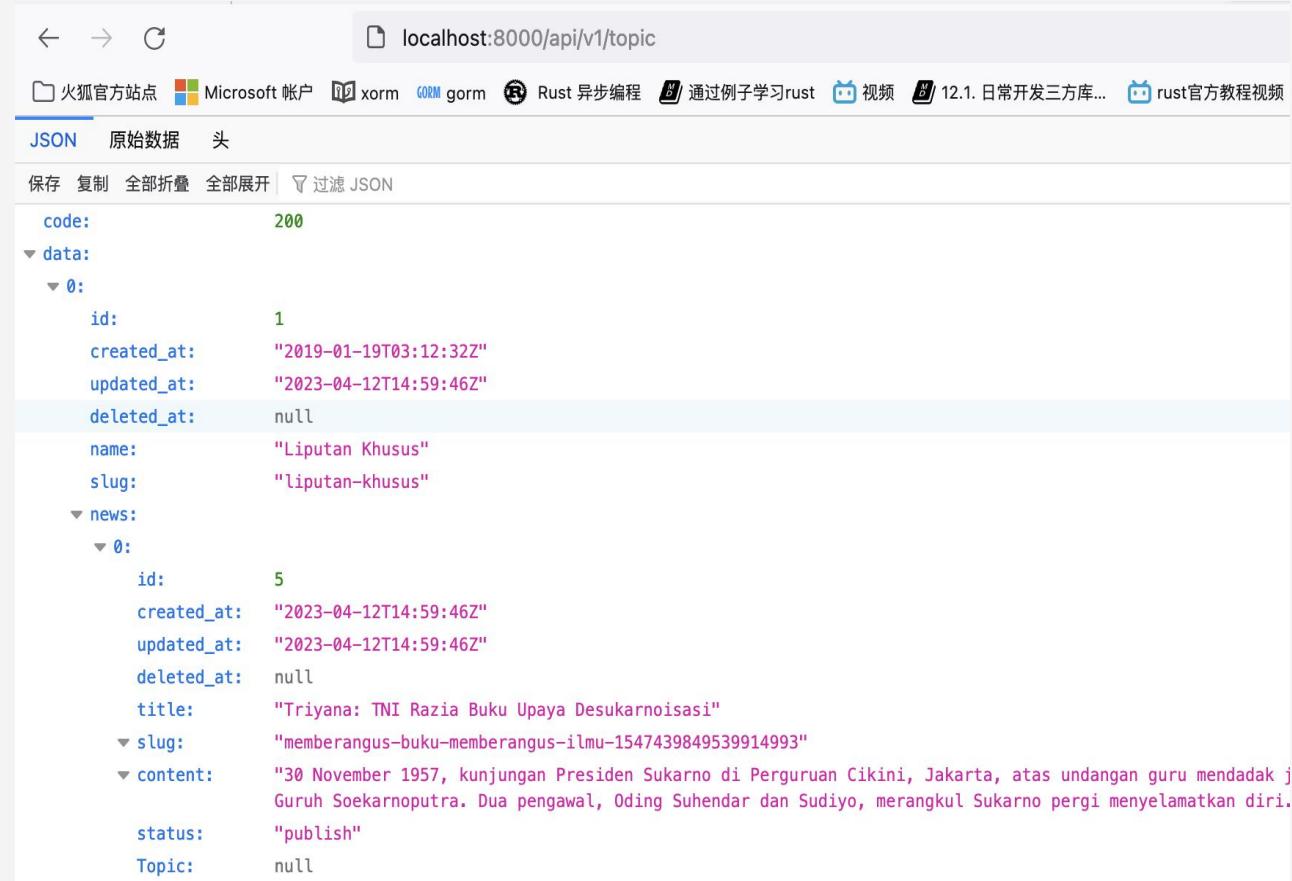


```

go-ddd-api
├── docs
└── internal
    ├── api
    │   ├── middleware
    │   └── news
    │       └── news_handler.go
    ├── topics
    │   └── handler.go
    └── router.go
├── application
│   ├── news.go
│   └── topic.go
└── domain
    ├── infra
    │   ├── config
    │   ├── migration
    │   ├── monitor
    │   └── pagination
    └── persistence
        ├── news_repository.go
        └── topic_repository.go
un: go build github.com/daheige/go-ddd-api
2023/04/12 23:07:10 Server running on:0.0.0.0:8000
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env: export GIN_MODE=release
- using code: gin.SetMode(gin.ReleaseMode)

[GIN-debug] GET    /          --> github.com/daheige/go-ddd-api/internal/api.(*RouterHandler).home-fm (3 handlers)
[GIN-debug] GET    /api/v1           --> github.com/daheige/go-ddd-api/internal/api.(*RouterHandler).home-fm (3 handlers)
[GIN-debug] GET    /api/v1/news        --> github.com/daheige/go-ddd-api/internal/api.news.(*NewsHandler).GetAllNews-fm (3 handlers)
[GIN-debug] GET    /api/v1/news/:param  --> github.com/daheige/go-ddd-api/internal/api.news.(*NewsHandler).GetNews-fm (3 handlers)
[GIN-debug] POST   /api/v1/news        --> github.com/daheige/go-ddd-api/internal/api.news.(*NewsHandler).CreateNews-fm (3 handlers)
[GIN-debug] DELETE  /api/v1/news/:news_id --> github.com/daheige/go-ddd-api/internal/api.news.(*NewsHandler).RemoveNews-fm (3 handlers)
[GIN-debug] PUT    /api/v1/news/:news_id --> github.com/daheige/go-ddd-api/internal/api.news.(*NewsHandler).UpdateNews-fm (3 handlers)
[GIN-debug] GET    /api/v1/topic         --> github.com/daheige/go-ddd-api/internal/api.topics.(*TopicHandler).GetAllTopic-fm (3 handlers)
[GIN-debug] GET    /api/v1/topic/:topic_id --> github.com/daheige/go-ddd-api/internal/api.topics.(*TopicHandler).GetTopic-fm (3 handlers)
[GIN-debug] POST   /api/v1/topic         --> github.com/daheige/go-ddd-api/internal/api.topics.(*TopicHandler).CreateTopic-fm (3 handlers)
[GIN-debug] DELETE  /api/v1/topic/:topic_id --> github.com/daheige/go-ddd-api/internal/api.topics.(*TopicHandler).RemoveTopic-fm (3 handlers)
[GIN-debug] PUT    /api/v1/topic/:topic_id --> github.com/daheige/go-ddd-api/internal/api.topics.(*TopicHandler).UpdateTopic-fm (3 handlers)
[GIN-debug] POST   /api/v1/migrate      --> github.com/daheige/go-ddd-api/internal/api.(*RouterHandler).migrate-fm (3 handlers)
2023/04/12 23:07:10 server pprof run on:0.0.0.0:8090

```



localhost:8000/api/v1/topic

JSON 原始数据 头
保存 复制 全部折叠 全部展开 过滤 JSON

```

code: 200
data:
  0:
    id: 1
    created_at: "2019-01-19T03:12:32Z"
    updated_at: "2023-04-12T14:59:46Z"
    deleted_at: null
    name: "Liputan Khusus"
    slug: "liputan-khusus"
  news:
    0:
      id: 5
      created_at: "2023-04-12T14:59:46Z"
      updated_at: "2023-04-12T14:59:46Z"
      deleted_at: null
      title: "Triyana: TNI Razia Buku Upaya Desukarnoasisi"
      slug: "memberangus-buku-memberangus-ilmu-1547439849539914993"
      content: "30 November 1957, kunjungan Presiden Sukarno di Perguruan Cikini, Jakarta, atas undangan guru mendadak j Guruuh Soekarnoputra. Dua pengawal, Oding Suhendar dan Sudijo, merangkul Sukarno pergi menyelamatkan diri."
      status: "publish"
      Topic: null

```

5 总结 & 收益



- ✓ DDD领域驱动设计能解决软件的复杂度，适合偏复杂业务的编排和组合。
- ✓ DDD领域驱动设计不是万能的，非银弹，不同的业务场景需根据实际情况选择不同的架构设计。
- ✓ GO DDD分层建议采用严格的分层设计，不跨层调用，组件的依赖关系通过依赖注入的方式解决。
- ✓ GO DDD目录layout并不是固定不变的，不同的团队使用的layout不一样。
- ✓ GO DDD分层设计的接口比较多的话，接口设计命名尽量遵循简洁的命名方式。
- ✓ GO DDD领域驱动设计在做战术落地时，尽量按照domain->infras->application->api的步骤展开，层与层之间的职责要分明，组件相互独立。

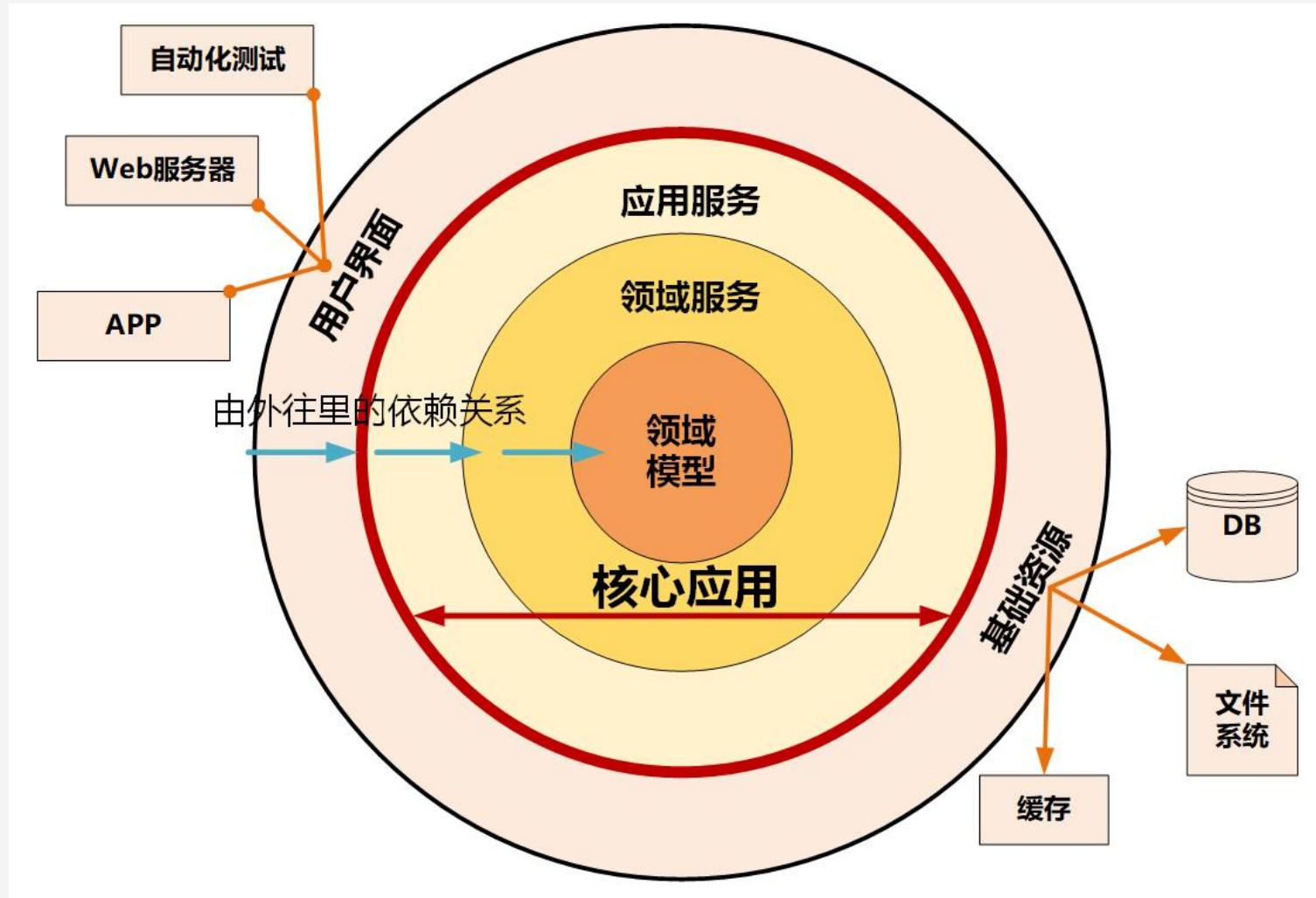


Q & A

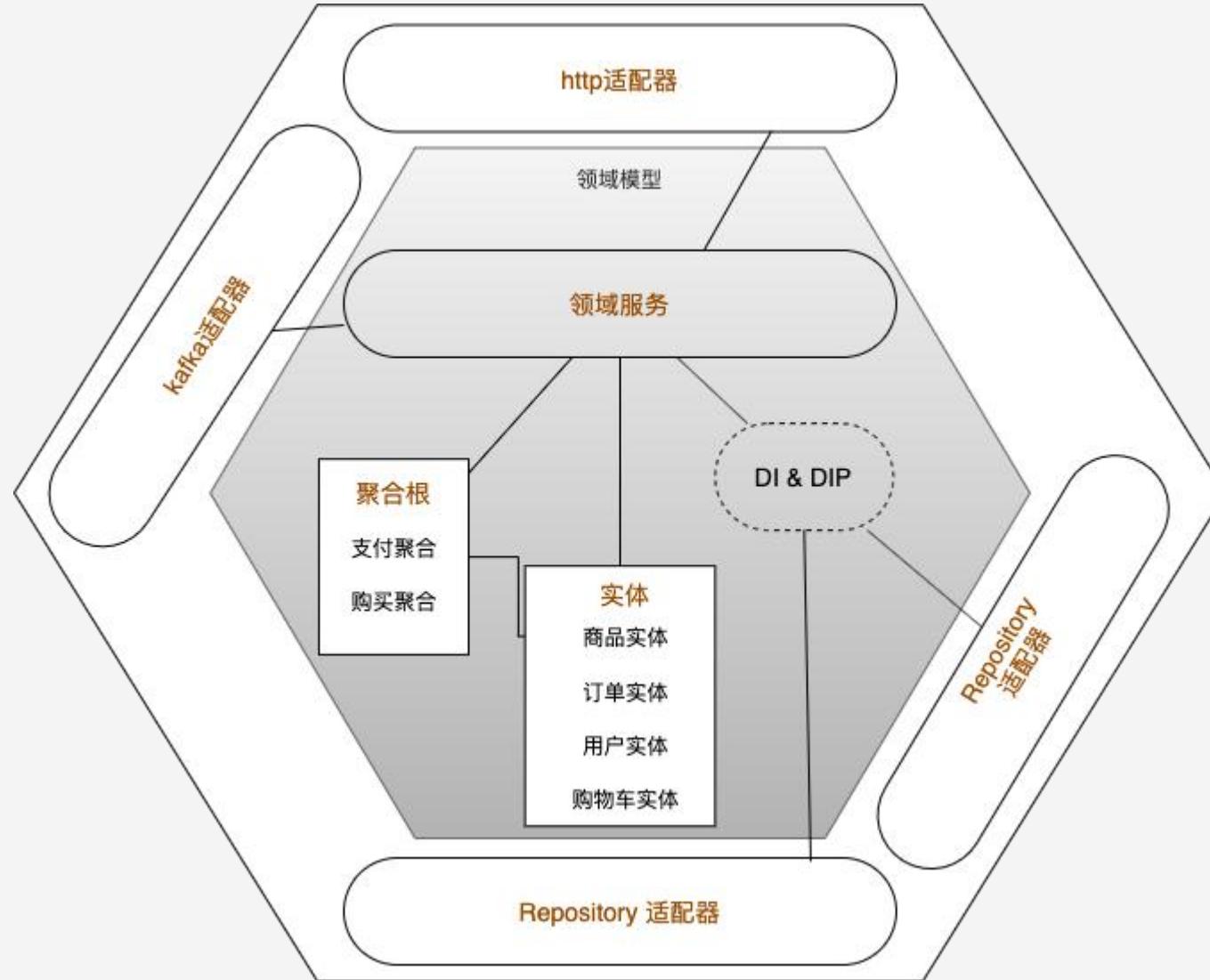
Thanks



附录1-洋葱架构模型



附录2-六边形架构模型



附录3-参考资料

《架构整洁之道》--Robert C. Martin (罗伯特 C. 马丁)

《实现领域驱动设计》--Aughn Vernon (沃恩.弗农)

《软件架构：架构模式 特征及实践指南》--Mark Richards (马克)

《领域驱动设计--软件核心复杂性应对之道》--Eric Evans (埃文斯)

