



下载APP



## 12 | 结构：如何系统设计框架的整体目录？

2021-10-13 叶剑峰

《手把手带你写一个Web框架》

课程介绍 &gt;



讲述：叶剑峰

时长 17:32 大小 16.07M



你好，我是轩脉刃。

到现在，我们已经将 Gin 集成到框架 hade 中，同时又引入了服务容器和服务提供者，明确框架的核心思想是面向服务编程，一切皆服务，所有服务都是基于协议。后续也会以服务的形式，封装一个个的服务，让我们的框架越来越丰富。

但是服务文件代码应该存放在哪里呢？这是我们会最先遇到的问题。业务的目录结构是否应该有规范，应该如何规范？所以今天这节课我们就来讨论这个问题，从系统的角度考虑框架的整体目录设计。



### 框架和业务区分

有人可能会问了，我们写的不是一个 Web 框架么，为什么要规范业务的目录结构？像 Gin 框架，就没有规范业务应用的目录结构，业务方根据自己的需要，想怎么组织业务目录就怎么组织，不好么？作为一个框架，想规范业务的目录结构，这样做是不是有点越俎代庖？

这个问题其实很有意义，我们需要搞清楚目录结构到底是用来干什么的。

**业务代码的目录结构是一种工程化的规范。**所谓工程化，简单来说就是希望不管是谁，在一个工程项目中，都按照一种做法来完成某个事情。而目录结构，就是项目工程化的一个起点。

在一个公司或者一个部门中，如果有架构团队，基本上要做的第一个事情就是，规范公司或者部门的代码目录结构。整体目录结构不仅仅代表着分层、归纳，也包含着很多架构的思想。

而对于 hade 框架而言，因为目标是将框架应用于实际生产，工程化使用，所以对业务的目录结构，**制定最小化的工程化规范，并且提供默认的整体目录结构**是很有必要的。

不过请注意，这里是希望制定最小化的工程化规范，就是说，对于这个框架，规范某些最小化的功能性目录是必要的，如果没有这些功能性目录，我们会认为目录设计是不合理的。

下面就来讨论下如何制定目录结构。

## 如何设计

在制定具体的目录结构之前，我们要明确一点：**业务的目录结构也是一个服务，是一个应用目录服务。**在这个服务中，我们制定框架要求的最小化的工程化规范，即框架要求业务至少有哪些目录结构。

而在其他服务中，一旦需要用到某个目录，我们能从目录结构服务中，查找出对应的结构。比如后续要创建配置服务，它需要去某个配置目录文件中读取配置，而去哪个配置目录呢，只需要去服务容器中获取这个应用目录服务就能知道了。


所以按照面向接口编程的思想，先为“应用目录服务”定义一下服务接口协议，也就是应用要提供哪些接口。

在今后的章节中，我们会定义许多框架级别的基础服务，这些服务我们都存放在框架目录中。**服务的接口协议，统一放在框架目录下的 framework/contract 中，而对应的服务提供者和服务实现，统一存放在框架目录下的 framework/provider 中。**

这里将所有框架级别的接口协议放在 framework/contract 中的设计有两个好处。

一是框架协议的关键字，我希望使用 contract.xxx 这个语义来区分，比如 App 服务的接口为 contract.App、日志服务的接口为 contract.Log，它们的 namespace 都是 contract，这样在使用的时候记忆成本会比较低。另外将框架提供的所有接口协议都放在一个文件夹中，在阅读框架提供哪些服务的时候，也更清晰明了。

以“应用目录服务”为例，我们在 framework/contract 目录中创建 app.go 存放服务协议。在 framework/contract/app.go 中，我们定义应用目录服务接口名称为 App，代表整个应用。它对应的服务字符串凭证为“hade:app”。

 复制代码

```
1 const AppKey = "hade:app"
```

这个接口的抽象，我们按照创建应用需要哪些目录的顺序来思考。

首先创建一个应用需要明确应用所在的根目录，这个就是根目录 BaseFolder；其次，我们需要一个目录保存配置文件，所以要有一个配置文件的目录 ConfigFolder；应用要输出日志，日志输出的存放路径也是需要设置的，所以我们再创建一个 LogFolder 来设置日志存放。

再考虑下，业务的服务提供者 and 对应接口需要有一个地方存放，我们将其命名为 ProviderFolder。同时，我们业务也有可能创建自己的中间件，所以需要有个 MiddlewareFolder 来存放中间件。

在后续的章节中，我们预计让框架支持命令行命令，并且使用命令行来控制进程的运行时状态，所以需要有个 CommandFolder 来存放各种命令行命令、一个 RuntimeFolder 来

存放运行时的进程 ID 等信息。

最后当然了还需要有一个文件来存放单元测试信息，比如单元测试的初始化和终止的一些默认操作等，所以我们有一个 TestFolder 目录。

按照目前能想到的需求，可以将目录先分为这些文件夹，后续在开发模块的过程中，遇到需要增加的目录，我们还可以继续迭代修改 App 接口。

按照以上的分析，我们在框架的 framework/contract/app.go 中定义了一个 App 的服务接口：

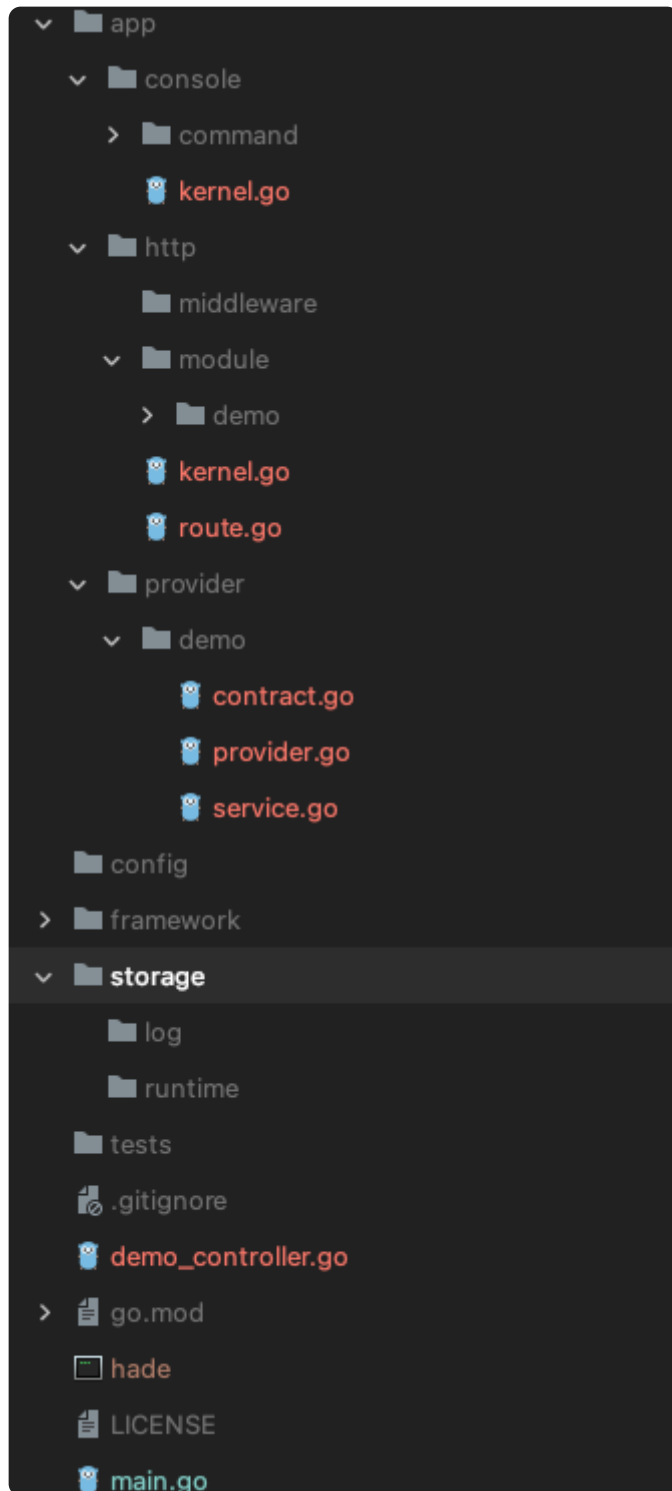
[复制代码](#)

```
1 package contract
2 // AppKey 定义字符串凭证
3 const AppKey = "hade:app"
4 // App 定义接口
5 type App interface {
6     // Version 定义当前版本
7     Version() string
8     //BaseFolder 定义项目基础地址
9     BaseFolder() string
10    // ConfigFolder 定义了配置文件的路径
11    ConfigFolder() string
12    // LogFolder 定义了日志所在路径
13    LogFolder() string
14    // ProviderFolder 定义业务自己的服务提供者地址
15    ProviderFolder() string
16    // MiddlewareFolder 定义业务自己定义的中间件
17    MiddlewareFolder() string
18    // CommandFolder 定义业务定义的命令
19    CommandFolder() string
20    // RuntimeFolder 定义业务的运行中间态信息
21    RuntimeFolder() string
22    // TestFolder 存放测试所需要的信息
23    TestFolder() string
24 }
```

## 定义默认目录结构

定义好了应用目录服务接口，我们再来思考它的实现，也就是设计使用这个 hade 框架的应用的默认目录。

目录结构的设计我们可以先去看看市面上的优秀框架，这里挑选的是 PHP 目前最火的应用框架 [Laravel](#)，它的目录设计非常值得我们学习和参考，**很合理地将各个功能模块放在各自的文件，并且文件定义清晰、无歧义**。所以有了下面这个默认目录结构，我们来解释下对应的每个目录。

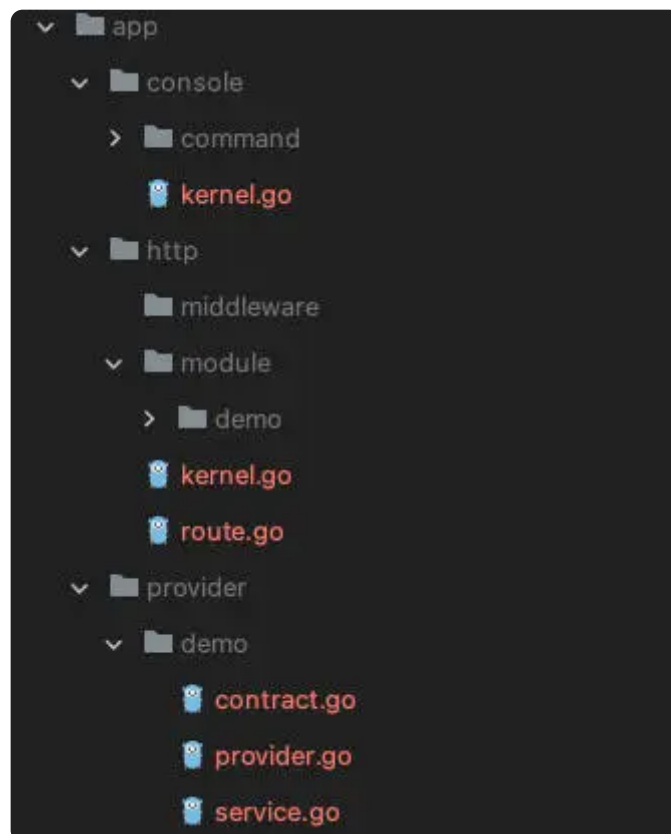


在根目录下，有五个文件：app、framework、config、storage、test，我们一个个描述功能看看下面都应该怎么划分。

## app 目录

一个业务就是一个 App，和业务代码相关的，比如所有的请求、控制台命令、控制器、服务提供者等都放在这个目录中，而业务逻辑代码之外的，比如配置文件、缓存、日志等，都放在外面。这样能把“业务逻辑”和“非业务逻辑”区分得更清晰。

我们继续看业务逻辑目录 app 下有哪些子目录。



第一层有三个目录：console、http、provider。因为业务除了提供 Web 服务，也提供控制台进程，所以在 app 目录下有两个子目录：一个是 console，业务中所有的控制台进程逻辑都放在这里；而另一个就是 http 目录，所有 Web 服务的逻辑都存放在里面。

同时这两者有很多逻辑是通用的，比如有可能用到同一个服务提供者，所以我们将两者通用的服务提供者的代码 provider 目录，也放在 app 目录下。

再看第二层的细分目录。

首先是 app 目录下的 console，存放的是所有命令行工具的逻辑代码。

控制台命令设计为多级命令，比如在命令 `./hade task info` 中，hade 是命令行工具，task 为一级命令，info 为二级命令。所以在 console 目录下，我们要创建一个



command 目录，并且 command 目录下按照一级、二级.....递归保存。（关于命令行工具的更多详细设计和实现我们在下节课实现。）

提供 Web 服务的 http 目录还有更细的层级。

HTTP 服务一般会按照模块划分，比如一个图书馆业务，我们会分为注册模块、图书模块、用户模块等，所以我们在 http 目录下定义了一个 module 目录，这个 module 目录下每个子目录代表一个模块服务。而 Web 服务特有的通用中间件，我们使用 http 目录下的 middleware 目录来保存。

app 下的 provider 目录，提供的是定义一个服务需要的文件。

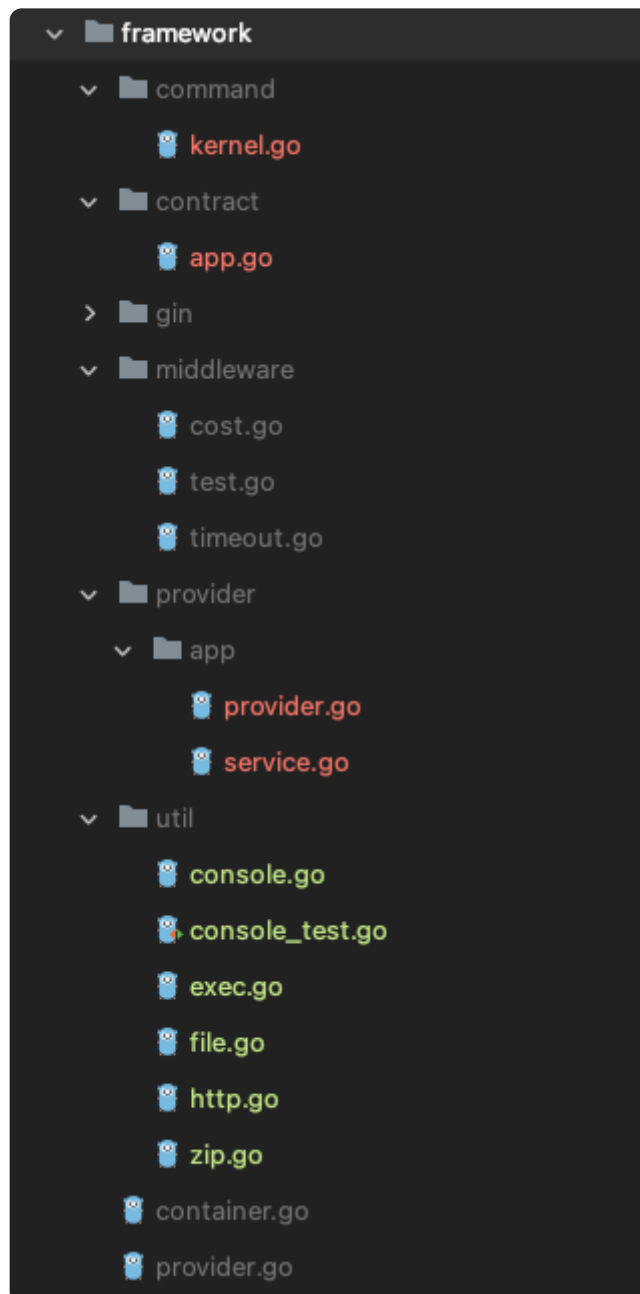
在 provider 目录中，每个子目录就代表一个业务服务。而按照上一节课描述的，每个服务都需要有三个文件：代表服务协议的 contract.go 文件、代表服务提供者的 provider.go 文件，以及代表具体服务实现的 service.go 文件。

这里要强调是，这个 provider 目录存放业务提供的服务，而框架提供的服务，我们会放在 framework 目录下。

app 目录我们解释完了，下面来研究根目录下的其他几个文件夹。

## framework 目录

framework 目录就是我们这个框架所有的代码。在这个目录里，除了之前引入了 Gin 框架有的 gin 目录，还有一些子目录：command、contract、middleware、provider、util。



这几个目录的功能也比较好理解。command 提供的是框架自带的命令行工具；middleware 存放框架为 Web 服务提供的中间件；contract 存放框架默认提供的服务协议；而对应服务协议的具体实现以及服务提供者，我们存放在 provider 目录下；util 目录则存放在框架研发过程中通用的一些函数。

不难发现，provider 和 command 的服务，在业务（app 目录）和框架（framework 目录）层面都有相同的目录，区别就是一个是业务提供的，一个是框架提供的。

我们这个项目为了演示框架开发的全过程，把框架和业务合并在一起，但是因为最终版框架在开源发布的时候，framework 目录是会剥离出去，作为一个单独的 git 项目的。所以，你在开发的时候要时刻明确，哪些是框架提供的命令行工具和服务，哪些是业务提供



**的命令和服务。**框架的内容需要放在 framework 目录下，业务的就要放在 framework 外。

## config、storage、test 目录

在应用根目录下还剩 config、storage、test 目录。config 这个目录存放的是配置文件，至于不同环境的配置文件如何存放和组织，我们后面会统一探讨整体的配置文件存放机制。test 目录存放的是测试相关的信息，比如测试用例或者测试数据等信息。

应用根目录下的 storage 存放应用运行过程中产生的内容。首先是日志，日志是应用运行过程中必然产生的信息，在 storage 下的 log 子目录里保存；其次是运行的进程 ID 等信息，这些都会存放在 storage 下的 runtime 目录中。

好，目录结构讲完了，我们整理下，从应用根目录开始，分为 5 个子目录 app、framework、config、storage、test，其中：

app 按应用的使用方式和通用性分为 console 和 http，以及通用的 provider 目录。framework 根据框架需要提供的功能分为 command、contract、gin、middleware、provider 和 util 6 个子目录。

config 和 test 分别保存配置和测试相关的信息；storage 目录保存应用运行产生的信息，分为 log 和 runtime 两个目录。

## 默认目录的实现

在今天最开始我们说了，目录结构也是一个服务，其他服务想要使用目录结构的时候，可以通过服务容器，来获取目录结构服务实例。而现在我们有了想要创建的默认目录结构，那么下面就要实现这个目录结构服务实例了。

我们定义一个 HadeApp 来实现这个目录结构服务接口，**这个 HadeApp 结构的元素除了服务实例之外，只需要一个项目的基础路径 basePath。**因为我们设计的默认目录结构只要有了这个 basePath，就可以按照设计找出其余的所有目录了。


我们在 framework/provider/app/service.go 中定义 HadeApp 结构：

```
1 // HadeApp 代表 hade 框架的 App 实现
2
3 type HadeApp struct {
4     container framework.Container // 服务容器
5     baseFolder string             // 基础路径
6 }
```

看最重要的 BaseFolder 的实现思路。


BaseFolder 是获取项目的基础路径，所以我们可以提供一种方式，在注册服务提供者的时候，实现 BaseFolder 的设置。来看代码，设置 hade 框架默认的 App 服务提供者 HadeAppProvider，它带有一个可以设置的属性 BaseFolder，这个属性作为参数传入给服务实例初始化函数。

我们在 framework/provider/app/provider.go 中定义 HadeAppProvider 结构和方法：

 复制代码

```
1 // HadeAppProvider 提供 App 的具体实现方法
2 type HadeAppProvider struct {
3     BaseFolder string
4 }
5
6 ...
7
8 // Params 获取初始化参数
9 func (h *HadeAppProvider) Params(container framework.Container) []interface{} {
10     return []interface{}{container, h.BaseFolder}
11 }
12 ...
13
14
15
16 // NewHadeApp 初始化 HadeApp
17 func NewHadeApp(params ...interface{}) (interface{}, error) {
18     if len(params) != 2 {
19         return nil, errors.New("param error")
20     }
21     // 有两个参数，一个是容器，一个是 baseFolder
22     container := params[0].(framework.Container)
23     baseFolder := params[1].(string)
24     return &HadeApp{baseFolder: baseFolder, container: container}, nil
25 }
```

这样，在业务目录的 main.go 中注册 App 服务的时候，就可以创建指定我们的 BaseFolder：

 复制代码

```
1 func main() {
2     // 创建 engine 结构
3     core := gin.New()
4     // 指定 BaseFolder
5     core.Bind(&app.HadeAppProvider{BaseFolder: "/tmp"})
6     ...
7 }
```

但是刚才的实现在每次注册服务的时候，都需要设置一遍，为了提高易用性，我们希望这个 BaseFolder 能自动设置。怎么办？

这里要先考虑使用这个框架开发的业务的场景有哪些，一般来说会有两个，开发场景和运行场景。

在开发场景中，我们在开发业务代码时，这个 BaseFolder 代表业务的开发路径，即业务代码所在的目录；而在另一个运行场景中，框架开发的业务需要去现网运行，只需要二进制文件、配置文件、日志文件即可，因为这时候的 BaseFolder 其实是会根据项目部署地址变化的。

所以对于第一种场景，我们提供获取当前开发路径的方法，可以使用 Golang 标准库的 os.Getwd() 方法获取到当前所在的开发路径。

 复制代码

```
1 // GetExecDirectory 获取当前执行程序目录
2 func GetExecDirectory() string {
3     file, err := os.Getwd()
4     if err == nil {
5         return file + "/"
6     }
7     return ""
8 }
```

而对于第二种运行场景，我们希望这个 BaseFolder 在运行的时候再指定。所以可以为程序设计一个 base\_folder 参数，这样在运行的时候，就解析参数将 BaseFolder 设置对应

**值。**借用 Golang 的 flag 标准库，我们能很容易做到解析参数、获取参数。

BaseFolder 的代码实现如下，其中有两次判断。我们先判断服务提供者是否有指定，如果有指定的话使用指定值，如果没有指定的话，再判断；如果是运行场景，看命令行参数中有没有传递 base\_folder 参数，传递了就使用参数设置 BaseFolder，否则默认为开发场景，使用当前路径。

于是 framework/provider/app/service.go 定义 BaseFolder 如下：

[复制代码](#)

```
1 // BaseFolder 表示基础目录，可以代表开发场景的目录，也可以代表运行时候的目录
2 func (h HadeApp) BaseFolder() string {
3     if h.baseFolder != "" {
4         return h.baseFolder
5     }
6     // 如果没有设置，则使用参数
7     var baseFolder string
8     flag.StringVar(&baseFolder, "base_folder", "", "base_folder 参数，默认为当前路
9     flag.Parse()
10    if baseFolder != "" {
11        return baseFolder
12    }
13    // 如果参数也没有，使用默认的当前路径
14    return util.GetExecDirectory()
15 }
```

在 BaseFolder 的实现中，关键点就是需要区分业务场景。服务协议的其他接口基本都是从 BaseFolder 这个目录上进行扩展的。

比如存放日志的目录 LogFolder，我们先通过 BaseFolder 定义出 StorageFolder 目录，然后再通过 StorageFolder，定义出 LogFolder 的位置。同样在 framework/provider/app/service.go 中定义对应的 StorageFolder 和 LogFolder 下，在后面讲配置的时候会更新这段代码：

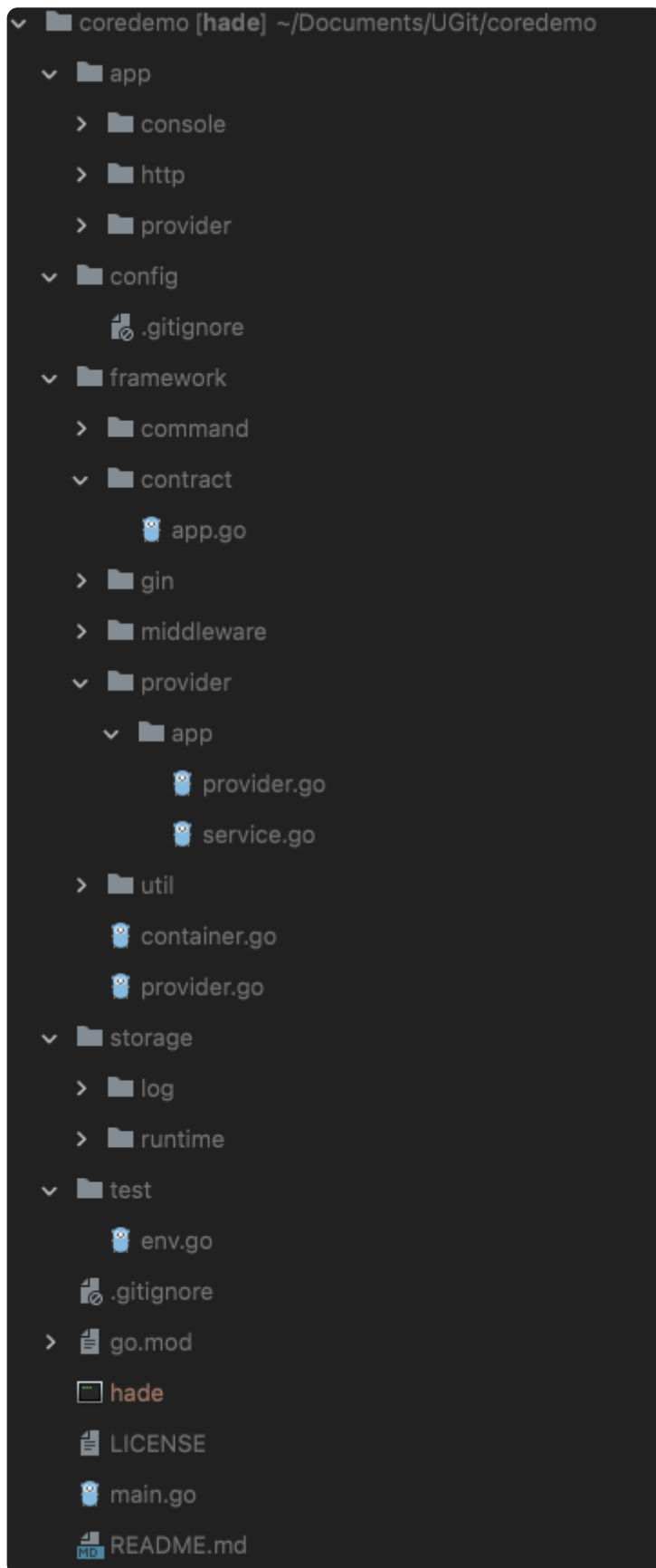
[复制代码](#)

```
1 func (h HadeApp) StorageFolder() string {
2     return filepath.Join(h.BaseFolder(), "storage")
3 }
4
5 // LogFolder 表示日志存放地址
```

```
6 func (h HadeApp) LogFolder() string {  
7     return filepath.Join(h.StorageFolder(), "log")  
8 }
```

其他目录地址的实现就不再赘述了，都是大同小异的。

本节课的所有代码我都存放在 GitHub 上的 [@geekbang/12](#) 分支上了。这里附上当前的代码结构截图。



## 总结

今天我们其实就讨论了一个核心问题：如何从框架层来规范业务的目录结构。是不是有点惊讶，只是一个目录结构的设计，居然也有如此多的门道。框架设计就是这样，好的框架



之所以能让所有人都喜欢，就是因为具备非常优秀的设计感，在每个模块和每个细节点上，都包含作者的思考。

比如我们的目录结构，**不仅仅是一种分目录的设计，还贯彻了面向接口的思想，将目录作为一个服务提供在服务容器中**，后续的所有服务在使用到业务目录的时候，可以直接通过这个目录服务，获取具体的目录路径，是非常方便的。

## 思考题

我们对框架的目录做了详细的拆分，其中 HTTP 服务中的 module 目录，表示具体的业务模块。

这个业务模块的文件设计，我们没有限制，你可以使用 MVC，也可以使用 DDD 来做具体的业务模块文件设计，这个文件结构不知道你日常有没有什么设计倾向？这是一个开放性问题，我们可以一起讨论。

欢迎在留言区分享你的思考。感谢你的收听，如果你觉得有收获，也欢迎你把今天的内容分享给你身边的朋友，邀他一起学习。我们下节课见。

分享给需要的人，Ta 订阅后你可得 **20 元现金奖励**



生成海报并分享

👍 赞 0

💡 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 11 | 面向接口编程：一切皆服务，服务基于协议(下)

下一篇 13 | 交互：可以执行命令行的框架才是好框架

# 技术管理案例课

踩坑复盘+案例分析+精进攻略=高效管理

许健

eBay 基础架构工程研发总监



新版升级：点击「🔗 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言 (2)

💬 写留言



qinsi

2021-10-13

老师能不能举个例子，App服务会有哪些不同的实现？常见的服务比如日志、数据库、消息队列之类的，很容易理解会有不同的实现。那App服务不同的实现是怎么样的？

光看目前App服务的接口的话，似乎都是在规定路径？那么不同的实现就是路径可能会不一样？可能换个名字叫ProjectLayout服务更能体现其用途？

展开 ∨



芒果少侠

2021-10-13

首先看这节课的时候，我是感觉有点抽象和难以理解的，烦请老师指正一下。这个app服务，是不是就相当于一个业务服务（基于hade框架完成）？

关于思考题：一般来说，我这边工作上是这样的：

DDD是在类似本文的app这一维度划分的，比如说我有多个服务app1,app2,app3。每个服...

展开 ∨

💬 1

👍 1

