



下载APP



18 | 一体化：前端和后端一定要项目分开吗？

2021-10-27 叶剑峰

《手把手带你写一个Web框架》

[课程介绍 >](#)



讲述：叶剑峰

时长 15:58 大小 14.64M



你好，我是轩脉刃。

从这节课开始，我们一起进入了实战第三关。如果说实战第一关怎么从零开始搭建框架是研究如何种栽一株盆栽，实战第二关框架核心的优化，是搭建了盆栽的枝叶，让盆栽可以健康生长，那么实战第三关就是为盆栽增加花叶，之后每一节课讨论和实现的功能都能为框架增色不少。

下面让我们开始第三关的第一节课框架一体化吧。



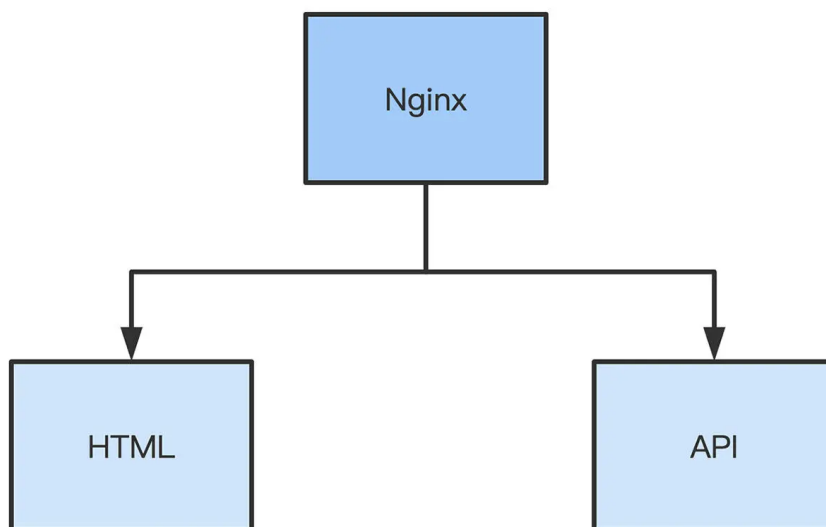
前后端一体化的架构方案

在前后端分离流行的现在，我们已经习惯了前端一个 Vue 项目 + 后端一个 Web 接口项目的组合。这样的组合对于大项目来说，非常符合社会分工的原则，前端由一个人或者一个部门来负责，后端由另外一个人或者一个部门来负责。

不过作为后端工程师，你一定遇到过要研发一个简单页面的需求。比如一个开发使用的运营后台或者一个简单的工具页面。这个时候如果一个人开发两个项目，你应该会有在两个项目中频繁切换疲于奔命的感觉。所以**如果一个框架能同时支持前端开发，又支持后端接口开发，且两者能完美融合一起**，那该多好啊。这也是我最开始萌生的想法。

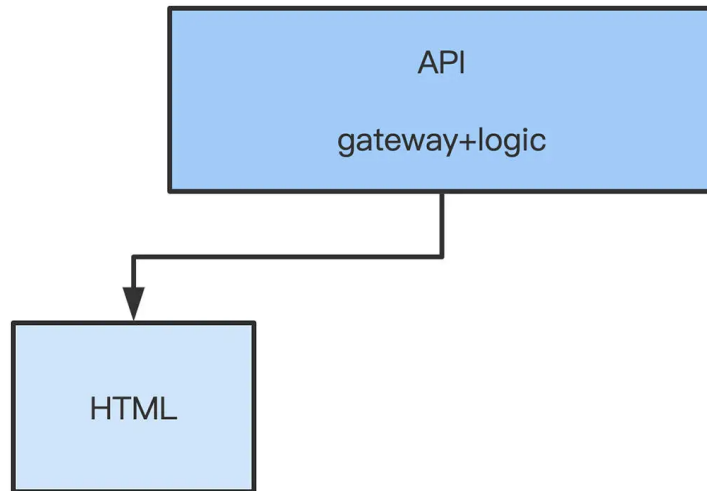
那我们先来思考下，是否可以把这个在一个项目中前后端一体化的设想，设计到真实架构层面的修改中？

在架构层面，前后端分离的架构经常是，Nginx 作为网关，前端页面作为 Nginx 的一个 location 路由，而后端接口作为 Nginx 的另外一个 location 路由。



这里其实还有一段逻辑代码是存放在 Nginx 配置中的。所以如果前后端分离，你需要调试开发的除了前端、后端，还有 Nginx 这一端。而且很多时候，我们会需要在网关 Nginx 层增加一些统一逻辑，比如有时候会增加路由重写、权限认证等，这里的逻辑也是比较复杂的，于是诞生出 nginx_lua 之类的网关层的逻辑编写方法。

但是在 Golang 的时代，我们是有另外一种选择的，那就是**将 API 层往上提，使用 Golang 替代网关的逻辑**，架构形如：



不知道你能否感受到这两种架构的区别和适用场景？先说明一下，**架构无好坏，只有适用不适用**。



第一种架构的缺点是复杂度高一些，而且一旦在网关层增加逻辑，因为语言异构，网关层逻辑和业务层逻辑不是一个，开发起来会十分痛苦。但是它的优点是模块化，前端、后端、网关是独立模块，互相不干扰。

而第二种架构的缺点是将三个模块一体化，但是也正是由于一体化了，开发复杂度降低了不少，开发效率提高，通用网关逻辑的研发成本就降低了。

在实际开发过程中，第二种架构也是有一定场景需求的，比如对一些管理后台、一些快速工具页面的开发，会非常有帮助。所以我们这节课要做的事情就是让 hade 框架，能快速支持这种前后端一体化的架构。

前后端一体化改造

既然架构上可以这么做，也有需求。我们来思考如何做前后端一体化。

前端框架我们就选用目前最火的  **Vue 框架**。Vue 框架是曾就职于 Google 的华人工程师尤雨溪的大作，目前  在 **GitHub** 上已经有 189k 之多的 star 了。这里简要说一下 Vue 的使用方式，具体的语法和原理也有很多课程介绍就不多说。

Vue 是一种带构建工具的前端代码生成方式，它的逻辑代码都是以 vue 后缀的文件实现的，每个 vue 文件代表一个页面模块，这个模块包含页面 HTML、逻辑脚本 JS 和页面样式 CSS。逻辑代码通过构建工具，最终会编译成在浏览器中可运行的 HTML 和 JS 文件。

所以我们只需要将 Vue 最终编译出来的文件目录，在 Golang 框架中进行代理，让某个路径能访问到这个静态目录文件即可。不知道你还记得第一节课讲 net/http 留的思考题么，思考题中提到 HTTP 库提供 FileServer 来封装对文件读取的 HTTP 服务。我们就可以使用这个 FileServer，来对前端编译出来的最终的浏览器可运行文件，提供 HTTP 服务。

所以，**将 Vue 的项目代码集成到业务代码中，然后确定其编译结果文件夹，在路由中，将某个请求路由到我们的编译结果文件夹**，就完成了上面架构提到的同一个项目同时支持前端请求和后端请求了。

逻辑很清晰，下面就来进行具体的操作。

把 Vue 项目代码集成到业务代码中

我们使用 Vue 自带的 vue-init 命令来初始化一个完整的 Vue 项目。在使用 vue-init 的时候，会需要选择使用何种构建方式，这里可以选择通用的 Webpack 构建工具。

使用命令 `vue-init webpack hade` 创建一个最完整的 hade 项目，在创建的过程中，具体是否初始化 vue-router、使用使用 eslint 等，都选用最默认开启的设置。具体的设置参考下图：

```
~/Documents/workspace/vuedemo > vue-init webpack hade

? Project name hade
? Project description A Vue.js project
? Author yejianfeng <jianfengye110@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests Yes
? Pick a test runner jest
? Setup e2e tests with Nightwatch? Yes
? Should we run `npm install` for you after the project has been created? (recommended) npm

vue-cli   Generated "hade".

# Installing project dependencies ...
# =====
```

创建完毕，初始化完成。

```
# Project initialization finished!
# =====

To get started:

  cd hade
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack
```

我们可以看到前端 hade 目录中的文件如下：

```
~/Documents/workspace/vuedemo/hade ll -a
total 72
drwxr-xr-x 16 yejianfeng staff 512B Sep 25 22:49 .
drwxr-xr-x  7 yejianfeng staff 224B Sep 25 17:00 ..
-rw-r--r--  1 yejianfeng staff 402B Sep 25 16:59 .babelrc
-rw-r--r--  1 yejianfeng staff 147B Sep 25 16:59 .editorconfig
-rw-r--r--  1 yejianfeng staff  51B Sep 25 16:59 .eslintignore
-rw-r--r--  1 yejianfeng staff 791B Sep 25 16:59 .eslintrc.js
-rw-r--r--  1 yejianfeng staff 213B Sep 25 16:59 .gitignore
-rw-r--r--  1 yejianfeng staff 246B Sep 25 16:59 .postcssrc.js
-rw-r--r--  1 yejianfeng staff 547B Sep 25 16:59 README.md
drwxr-xr-x 10 yejianfeng staff 320B Sep 25 16:59 build
drwxr-xr-x  6 yejianfeng staff 192B Sep 25 16:59 config
-rw-r--r--  1 yejianfeng staff 266B Sep 25 16:59 index.html
-rw-r--r--  1 yejianfeng staff 2.7K Sep 25 16:59 package.json
drwxr-xr-x  7 yejianfeng staff 224B Sep 25 16:59 src
drwxr-xr-x  3 yejianfeng staff  96B Sep 25 16:59 static
drwxr-xr-x  4 yejianfeng staff 128B Sep 25 16:59 test
```

这里很基础地认识一下这些目录和文件的具体作用。xxx 隐藏文件都是一些配置文件信息，比如语法配置、git 配置等信息。然后看加粗显示的五個目录：

build 目录，存放项目构建（Webpack）相关的代码。

config 是配置目录，包括端口号等配置。

src 目录，这个是我们开发的目录，业务逻辑代码基本上都在里面。其中有四项：assets 存放一些图片信息，比如 logo；components 存放组件信息文件；App.vue 存放项目的入口组件 App；main.js 是项目的入口 js，引用加载入口组件 App。

static 目录，存放静态文件信息，比如图片、字体等。

test 目录，存放测试相关的信息。

剩余的两项，index.html 是 Vue 首页的入口页面，package.json 是项目的配置文件，保存引用的第三方库等信息。

确定编译结果文件夹

前端 hade 项目生成之后，下一步我们就要把它的文件复制到 hade 框架的根目录下。

这里借用 IDE 就能直接 copy 前端 hade 目录下的文件到目标目录。因为和我们之前定义的文件夹和文件并没有什么冲突，所以就不修改任何的文件，保持文件和文件夹一致复制就行。这里需要再提醒一下的是，要将隐藏文件也复制到 hade 框架的根目录下。

然后在 hade 框架的根目录下，我们调用命令 `npm install` 加载目录所需要的第三方库，再执行 `npm run build` 就能执行 Vue 的编译操作，编译 src 中的 Vue 代码文件。生成的 index 文件，存放在根目录的 dist 目录下，这个目录就是我们需要的要代理的静态文件目录了。

这里 Vue 的目录结构和这些配置文件，使用了 Vue 最标准的设置。最终，我们只关注编译出来的目录，所以除了这个目录之外，其他的 Vue 生成目录你都是可以按照需求进行修改的，也就是说，**你完全可以自主替换上面列出的让你认识的 Vue 目录文件。**

如果你对 Vue 比较熟悉，或者你有一个现成的 Vue 项目，比如在课程最后一关我们会用到的 vue-element-admin，它有自己的目录结构和配置，会直接进行替换的。


把请求路由到结果文件夹

下一步要来修改我们的 Golang 路由了。

这里要先思考一下了，我们希望路由是什么样子的？一般进入一个网站，打开默认路由 a.com/ 后，会访问默认 index.html，就是编译生成的 dist 文件夹下的 index.html。而在这个网站中，我们要同时访问静态文件 /a.js 或者动态请求 /api/demo/demo。**所以，顺序应该是先看静态文件在 /dist 文件夹中是否存在，如果存在则返回静态文件，如果不存在，则访问动态请求。**

那这种“先查询静态文件，再进行动态请求”的逻辑，在 Gin 的路由中如何实现呢？

其实 Gin 的生态中已经有这个实现方式了。在 Gin 的开源贡献项目 github.com/gin-contrib 中的一个中间件 `static` 就是实现这个的。来看 GitHub 上的使用例子：

 复制代码

```
1 func main() {  
2     r := gin.Default()  
3  
4     // 根目录先查询本地的tmp文件夹中是否有文件，返回这个文件
```

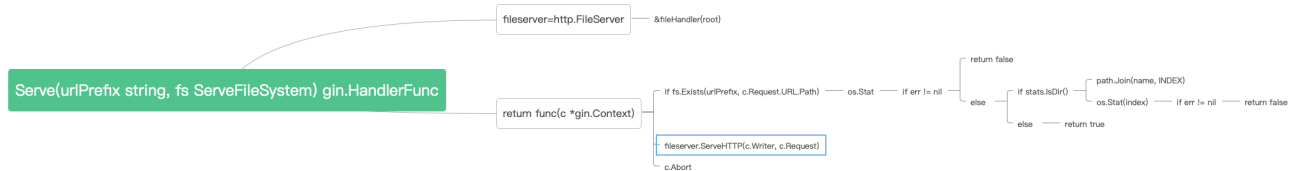
```

5  r.Use(static.Serve("/", static.LocalFile("/tmp", false)))
6  // 同时提供动态请求路由ping
7  r.GET("/ping", func(c *gin.Context) {
8      c.String(200, "test")
9  })
10 // 启动服务
11 r.Run(":8080")
12 }

```

它是怎么实现的呢？用的就是前面提过的 HTTP 库提供 FileServer。我们先将这个中间件复制到框架目录的 `framework/middleware/static/static.go` 文件中，再来具体分析下这段实现代码。

用之前的思维导图方法，就很快速能理清这个 `static` 中间件的原理。



它的逻辑大概是，先使用 `http.FileServer` 创建一个文件服务器，但是这个文件服务器只有处理逻辑 `fileHandler`，没有启动端口；所有请求会进入 `func(c*gin.Context)` 中间件处理函数，在这个处理函数中：


先判断最终路径的文件是否存在，如果存在，则找到目标文件，如果不存在，判断这个路径是否为目录，如果是目录，再判断目录下的 `index.html` 是否存在；如果存在，就找目标文件。

如果找到了目标文件，则调用文件服务器的 `ServeHTTP` 方法来处理这个请求，并且调用 `Abort` 来终止后续的请求。

如果没有找到目标文件，则什么都不做，继续后续的路由请求。

在复制的过程中注意两个点，一是将 `static.go` 中的 `github.com/gin-gonic/gin` 替换为我们 `hade` 框架的 `gin` 地址 `github.com/gohade/hade/framework/gin`；第二点是这个项目是 MIT 协议，所以我们要保留其协议申明，将 `LICENCE.md` 文件也同时复制。

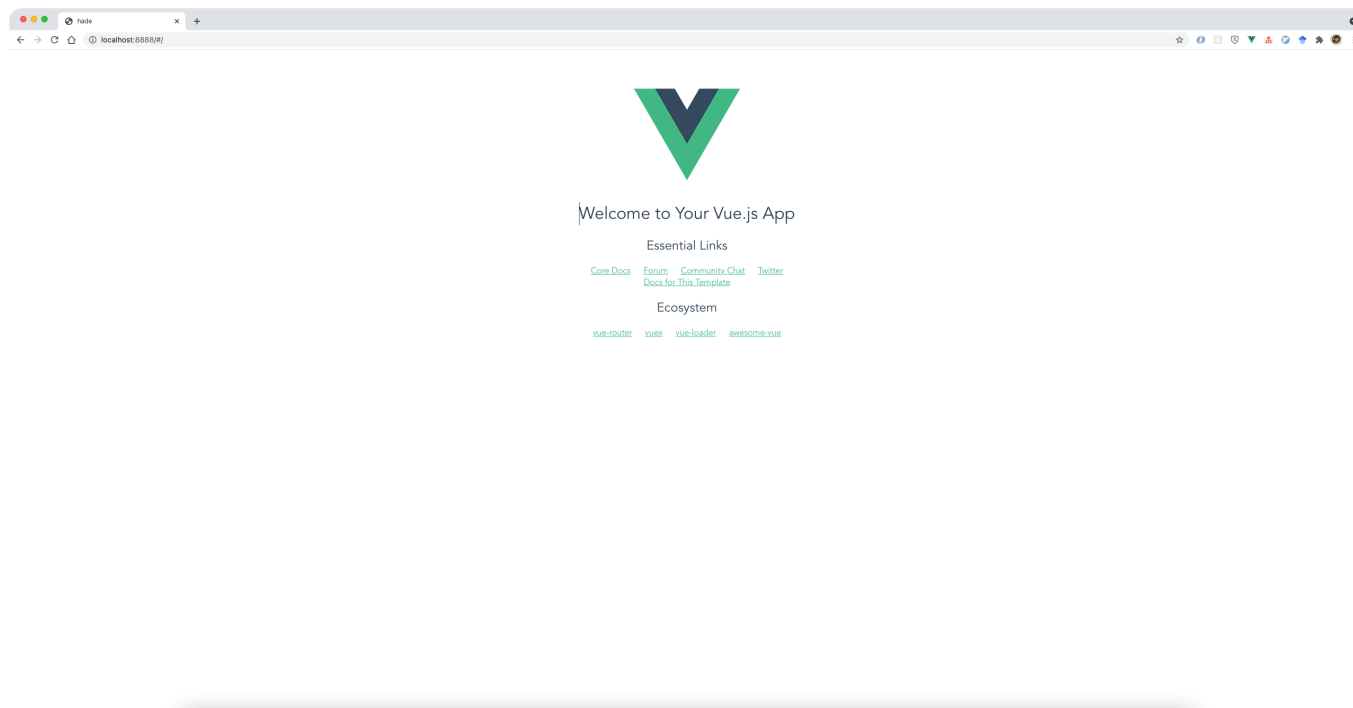
最后修改一下路由设置，业务目录下的 `app/http/route.go`：

 复制代码

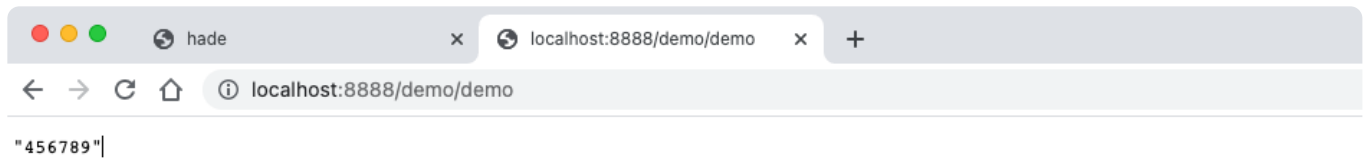
```
1 // Routes 绑定业务层路由
2 func Routes(r *gin.Engine) {
3
4     // /路径先去./dist目录下查找文件是否存在，找到使用文件服务提供服务
5     r.Use(static.Serve("/", static.LocalFile("./dist", false)))
6
7     // 动态路由定义
8     demo.Register(r)
9 }
```

好了，现在我们的前端 Vue 项目和静态路由就完成了。做一下验证，先使用命令 `npm install` 和 `npm build` 编译好前端，再使用 `go build` 编译好后端。然后使用命令 `./hade app start` 启动服务。

访问地址 <http://localhost:8888/> 能访问到 dist 目录下生成的 Vue 静态文件地址：



同时访问 <http://localhost:8888/demo/demo> 也能访问 Golang 定义的动态路由地址：




验证完毕，前后端一体化改造完毕！

前后端一体化编译命令改造

在改造过程中，不知道你有没有发现，我们频繁使用到了 `go build`、`npm build` 等命令进行前后端编译，在实际生产过程中，这种命令执行肯定会更加频繁。那顺着这个问题思考，既然我们这个框架有很方便的命令行工具，**能不能将这些编译命令统一封装一下变成方便使用的命令呢？**

可以把需求整理成以下四个命令：

- 1 编译前端 `./hade build frontend`
- 2 编译后端 `./hade build backend`
- 3 同时编译前后端 `./hade build all`
- 4 自编译 `./hade build self`


 复制代码

编译前端，封装 `npm build` 命令，而编译后端封装 `go build` 命令，同时编译前后端我们同时调用 `npm build` 和 `go build` 就行，自编译其实和编译后端一样，编译 `./hade` 命令行工具自身。

这里我们就用编译前端命令 `./hade build frontend` 来做一个具体实现说明，其他实现基本上都是大同小异了。

获取 `npm` 命令，我们使用 `Golang` 标准库自带的 `exec.LookPath` 来查找。如果查找到了，就接着使用 `exec.Command` 来运行 `npm run build`，并且将输出 `cmd.CombinedOutput`，输出到控制台中；如果查找不到，就打印出错误信息。

实现也不难，在框架目录 `framework/command/build.go` 中，我们编辑下列代码：

 复制代码


```
1 // 打印前端的命令
2 var buildFrontendCommand = &cobra.Command{
3     Use:     "frontend",
4     Short:   "使用npm编译前端",
5     RunE:    func(c *cobra.Command, args []string) error {
6         // 获取path路径下的npm命令
7         path, err := exec.LookPath("npm")
8         if err != nil {
9             log.Fatalln("请安装npm在你的PATH路径下")
10        }
11
12        // 执行npm run build
13        cmd := exec.Command(path, "run", "build")
14        // 将输出保存在out中
15        out, err := cmd.CombinedOutput()
16        if err != nil {
17            fmt.Println("===== 前端编译失败 =====")
18            fmt.Println(string(out))
19            fmt.Println("===== 前端编译失败 =====")
20            return err
21        }
22        // 打印输出
23        fmt.Print(string(out))
24        fmt.Println("===== 前端编译成功 =====")
25        return nil
26    },
27 }
```

同时要记得将这个 `buildFrontendCommand` 挂载到 `build` 系列命令中。在 `framework/command/build.go` 中：

 复制代码

```
1 // build相关的命令
2 func initBuildCommand() *cobra.Command {
3     ...
4     buildCommand.AddCommand(buildFrontendCommand)
5     ...
6     return buildCommand
7 }
```

并且挂载到框架目录的 `framework/command/kernel.go` 中：

 复制代码

```

1 // AddKernelCommands will add all command/* to root command
2 func AddKernelCommands(root *cobra.Command) {
3     ...
4     // build 命令
5     root.AddCommand(initBuildCommand())

```

现在我们可以使用命令 `framework/command/kernel.go` 来编译前端了：

```

~/Documents/UGit/coredemo  geekbang/18  ./hade build frontend
> hade@1.0.0 build /Users/yejianfeng/Documents/UGit/coredemo
> node build/build.js

Hash: 1a4be8a739fdf5fc594a
Version: webpack 3.12.0
Time: 4990ms

      Asset      Size  Chunks             Chunk Names
static/js/vendor.b38abd1e8f82054d7d19.js  124 kB      0  [emitted]  vendor
static/js/app.b22ce679862c47a75225.js    11.6 kB      1  [emitted]  app
static/js/manifest.2ae2e69a05c33dfc65f8.js  857 bytes      2  [emitted]  manifest
static/css/app.30790115300ab27614ce176899523b62.css  432 bytes      1  [emitted]  app
static/css/app.30790115300ab27614ce176899523b62.css.map  797 bytes             [emitted]
static/js/vendor.b38abd1e8f82054d7d19.js.map    627 kB      0  [emitted]  vendor
static/js/app.b22ce679862c47a75225.js.map    22.2 kB      1  [emitted]  app
static/js/manifest.2ae2e69a05c33dfc65f8.js.map   4.97 kB      2  [emitted]  manifest
index.html  506 bytes             [emitted]

Build complete.

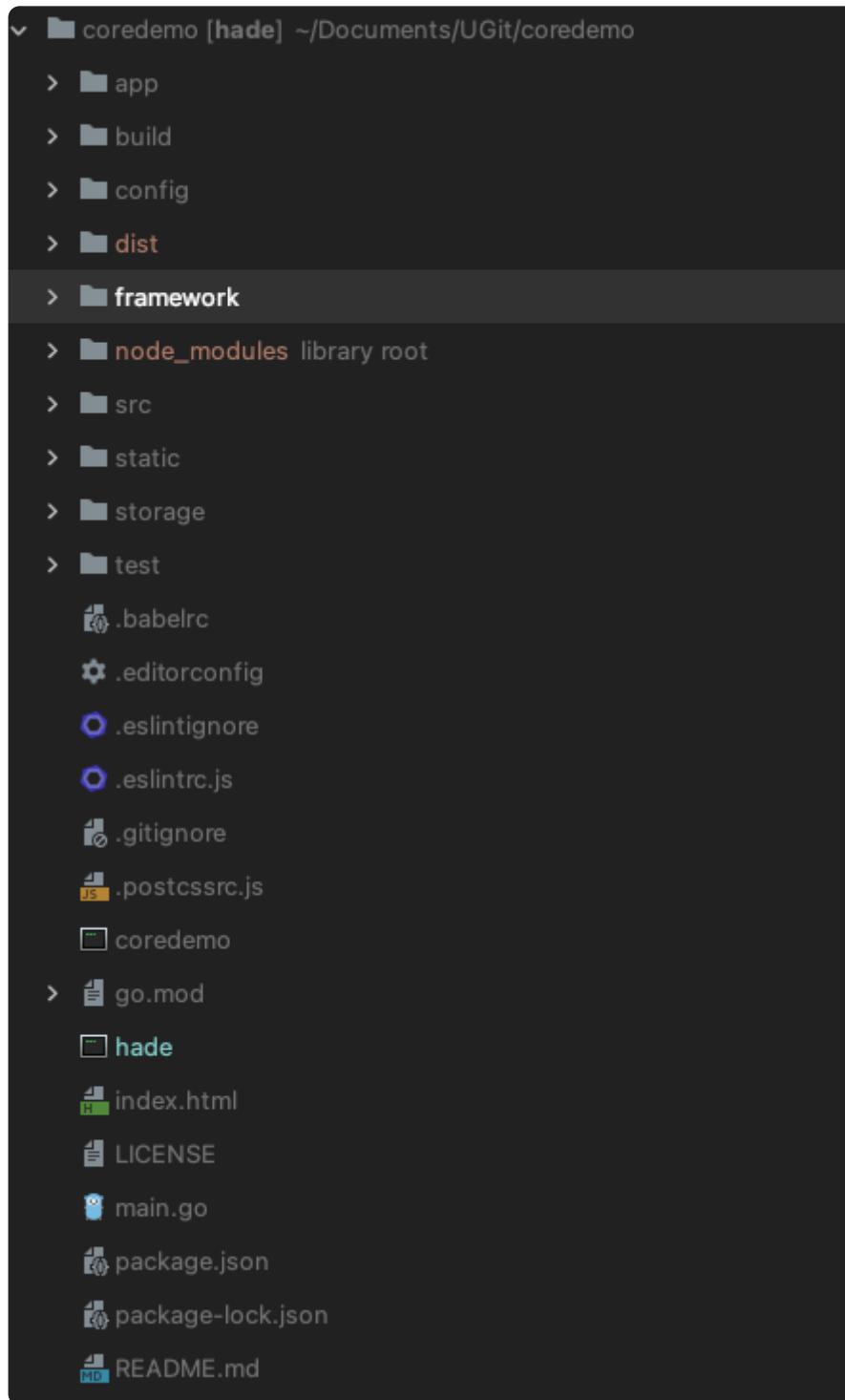
Tip: built files are meant to be served over an HTTP server.
Opening index.html over file:// won't work.

===== 前端编译成功 =====

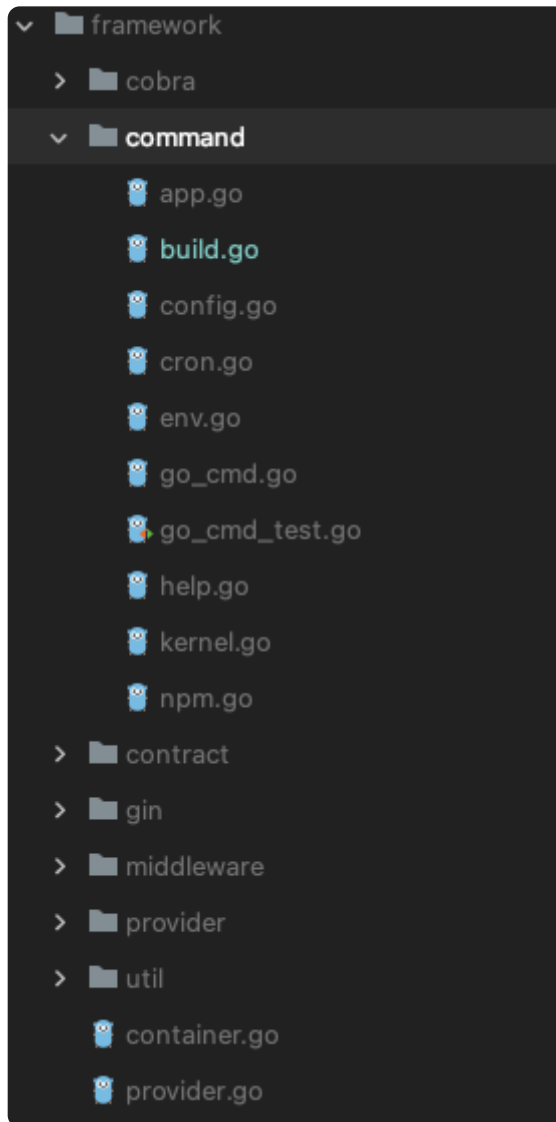
```

今天的主要内容前后端一体化改造，以及进一步优化成编译命令我们就完成了。完整的代码示例在 GitHub 上的 [@geekbang/18](#) 分支，欢迎比对查看。

本节课我们为 hade 框架增加了 Vue 标准项目文件：



并且在框架目录中增加了 build 系列命令：



小结

我们将前端的 Vue 集成进到了 hade 框架中，并且增加了 static 中间件，让框架具有同时提供前后端服务的功能。最后在改造过程中，发现频繁用到 go build、npm build 等命令做前后端编译，不太方便，所以我们改造了 build 命令行工具为统一编译前后端命令，提升了编译效率。

目前的框架，确实很少有支持前后端一体化的，但是我个人的工作经验来说，如果你开发的是一个运营后台系统，很多时候前端和后端都是一个人开发的，那么，前后端一体化的功能就是非常实用的，能大大加快我们的开发效率。

思考题

我们为 hade 框架增加了 build 命令，但是在运行过程中还会用到 go 和 npm 命令，我们希望框架所有可能用到的命令都封装在 ./hade 命令下，所以最好能封装一个 ./hade go 和 ./hade npm 命令。这个怎么实现呢？

欢迎在留言区分享你的思考。感谢你的收听，如果觉得有收获，也欢迎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 17 | 日志：如何设计多输出的日志服务？

1024 活动特惠

VIP 年卡直降 ¥2000

新课上线即解锁，享 365 天畅看全场

超值拿下 ¥999 



精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。