



下载APP



## 23 | 管理接口：如何集成swagger自动生成文件？

2021-11-08 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 18:52 大小 17.29M



你好，我是轩脉刃。

不管你是前端页面开发，还是后端服务开发，你一定经历过前后端联调的场景，前后端联调最痛苦的事情，莫过于没有完善的接口文档、没有可以调用调试的接口返回值了，所以一般都会采用形如 Postman 这样的第三方工具，来进行接口的调用和联调。

但是这一节课，我们要做的事情，就是为自己的 Web 应用集成 swagger，使用 swagger 自动生成一个可以查看接口、可以调用执行的页面。

**swagger**

说到 swagger，可能有的同学还比较陌生，我来简要介绍一下。swagger 框架在 2009 年启动，之前是 Reverb 公司内部开发的一个项目，他们的工程师在与第三方调试 REST 接口的过程中，为了解决大量的接口与文档问题，就设计了 swagger 这个项目。

项目最终成型的方案是，先设计一个 JSON 规则，开发工程师把所有服务接口按照这种规则来写成一个 JSON 文件，**这个 JSON 文件可以直接生成一个交互式 UI，可以提供调用者查看、调用调试。**

swagger 的应用是非常广泛的。非常多的开源项目在提供对外接口的时候都使用 swagger 来进行描述。比如目前最火的 Kubernetes 项目，每次在发布版本的时候，都会在项目根目录上，带上符合 swagger 规则的 [JSON 文件](#)，用来向使用者提供内部接口。

swagger 的产品有两类。

一个是前面说的 JSON 规则，就是 OpenAPI 的文档，它说明了我们要写一个接口说明文档的每个字段含义和要求。

OpenAPI 的规则也是有版本的，目前最新版本是 3.0，但是 3.0 版本目前市场上相应的配套支持还不成熟，比如 Golang 版本的 SDK 库 [spec](#) 还不支持。目前市面上对 OpenAPI2.0 的支持还是最全的。所以我们的 hade 框架就使用 swagger2.0 版本。

swagger 的另外一类产品是工具，包括 swagger-ui、swagger-editor 和 swagger-codegen。

[swagger-editor](#) 提供一个开源网站，在线编辑 swagger 文件。[swagger-codegen](#) 提供一个 Java 命令行工具，通过 swagger 文件生成 client 端代码。而 [swagger-ui](#)，通过提供一个开源网站，将 swagger 接口在线展示出来，并且可以让调用者查看、调试。我们的目标是生成一个可以查看接口，进行调用调试的页面，所以要将 swagger-ui 集成进 hade 框架。

## 命令设计

了解了 swagger，结合框架，我们照例先思考下希望如何使用它。

按照 swagger 的定义，我们应该在业务项目中维护一个 JSON 文件，这个文件描述了这个业务的所有接口。但是你想过没有，**随着项目的接口数越来越大，维护 swagger 的 JSON 描述文档本身，就是一个很大很繁杂的工作量。**

由于每个接口在代码开发的时候，我们都会有注释，而更新代码的时候，我们是会去更新注释的。所以能不能有一个方法，通过代码的注释，自动生成这个 JSON 文件呢？

好，这个就是我们希望定义的一个 swagger 命令，`./hade swagger gen`，能通过注释生成 swagger.json 文件。

但是考虑具体的实现设计，怎么用 Golang 的代码，注释生成 swagger.json 呢？既然 swagger.json 是有一定的规则的，那么注释的写法也是有一定规则的吧？是的。目前有一个最流行的将 Golang 注释转化为 swagger.json 的开源项目 [🔗 swag](#)。

## swag 项目


这个 swag 项目是 MIT 协议，目前已经有 4.9k 个 star 了。它的用法和我们想要的一样，生成 swagger.json 分三步：

在 API 接口中编写注释。注释的详细写法需要参考 [🔗 说明文档](#)。

下载 swag 工具或者安装 swag 库

使用工具或者库将指定代码生成 swagger.json

步骤很简单，不过第一步怎么写 swag 的注释说明文档，是使用这个技术必须要学习的一个知识，这个的学习确实是有些门槛的，需要熟读对应的说明文档才能写出比较好的注释。这里我们用一个例子来讲解我在编写代码的时候常用的一些字段，供你参考。

 复制代码

```
1 // Demo2 for godoc
2 // @Summary 获取所有学生
3 // @Description 获取所有学生，不进行分页
4 // @Produce json
5 // @Tags demo
6 // @Success 200 {array} []UserDTO
7 // @Router /demo/demo2 [get]
8 func (api *DemoApi) Demo2(c *gin.Context) {
9     demoProvider := c.MustMake(demoService.DemoKey).(demoService.IService)
```

```
10     students := demoProvider.GetAllStudent()
11     usersDTO := StudentsToUserDTOs(students)
12     c.JSON(200, usersDTO)
13 }
14
15 type UserDTO struct {
16     ID    int    `json:"id"`
17     Name string `json:"name"`
18 }
```

观察注释。第一行 Demo2 for godoc 这个在 swagger 中并没有实际作用，它是用来给 godoc 工具生成说明文档的。从第二行开始，就是我们 swaggo 的注释语法了，使用 @ 符号加上关键字的方式来进行说明。

例子的关键字有这些：

Summary，为接口增加简要说明

Description，为接口增加详细说明

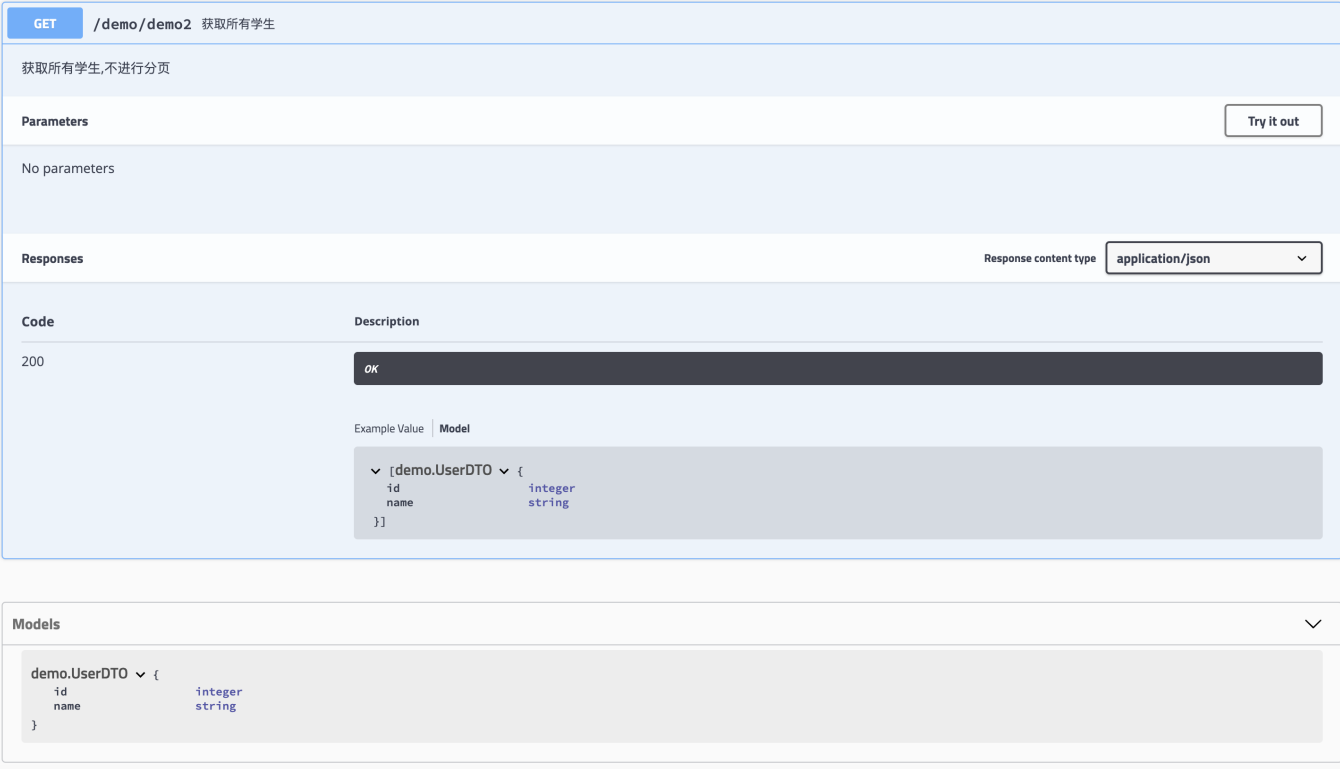
Produce，说明接口返回格式

Tags，为接口打标签，可以为多个，便于查看者查找

Success，接口返回成功时候的说明

Router，接口的路由调用

具体对应的 swagger-ui 界面是这样的：



我们对照注释和界面，很容易就看出每个注释的最终显示效果。不过这里再啰嗦解释下比较复杂的 `Success` 注释。

在这个例子中，是这样使用 `Success` 注释的：

```
1 // @Success 200 {array} UserDTO
```

复制代码

在成功的时候，返回 `UserDTO` 结构的数组，这里，`swaggo` 会自动去项目中寻找 `UserDTO` 结构，来生成 `swagger-ui` 中的返回结构说明。


不过这里能这么写，是因为恰好 `UserDTO` 是和 `API` 放在同一个 `namespace` 下，如果你的返回结构放在不同的 `namespace` 下，需要在注释中注明返回结构的命名空间。比如：

```
1 // @Success 200 {array} model.Account
```

复制代码

同时，这个返回结构还支持返回对象嵌套，比如下面这个例子：



 复制代码

```
1 // 返回了一个JsonResult对象，其中这个对象的数据字段是Order结构
2 @success 200 {object} jsonresult.JSONResult{data=proto.Order} "desc"
3
4 type JsonResult struct {
5     Code    int           `json:"code" `
6     Message string         `json:"message"`
7     Data    interface{}   `json:"data"`
8 }
9
10 type Order struct { //in `proto` package
11     Id    uint           `json:"id"`
12     Data interface{}     `json:"data"`
13 }
```


它返回了一个 JsonResult 对象，这个 JsonResult 对象中的一个字段 data 是 Order 结构。

## 命令实现

现在注释已经标记好了，我们再回到生成 JSON 文件的命令 `./hade swagger gen`。

这个命令通过 swaggo 准备好的命令行工具 swag 或者类库，来生成 JSON 文件。由于我们的框架已经集成了命令行工具，所以不会选择额外使用 swag 工具，而是在我们的命令中集成 swaggo 类库：[🔗 swag/gen](#)。

这个类库最核心的结构就是 [🔗 Config](#) 结构。来看这个 swagger gen 命令的具体实现代码，写在 `framework/command/swagger.go` 中：

 复制代码

```
1 // swaggerGenCommand 生成具体的swagger文档
2 var swaggerGenCommand = &cobra.Command{
3     Use:    "gen",
4     Short:  "生成对应的swagger文件，contain swagger.yaml, doc.go",
5     Run: func(c *cobra.Command, args []string) {
6         container := c.GetContainer()
7         appService := container.MustMake(contract.AppKey).(contract.App)
8         outputDir := filepath.Join(appService.AppFolder(), "http", "swagger")
9         httpFolder := filepath.Join(appService.AppFolder(), "http")
10        conf := &gen.Config{
11            // 遍历需要查询注释的目录
12            SearchDir: httpFolder,
13            // 不包含哪些文件
14            Excludes: "",
```

```
15         // 输出目录
16         OutputDir: outputDir,
17         // 整个swagger接口的说明文档注释
18         MainAPIFile: "swagger.go",
19         // 名字的显示策略，比如首字母大写等
20         PropNamingStrategy: "",
21         // 是否要解析vendor目录
22         ParseVendor: false,
23         // 是否要解析外部依赖库的包
24         ParseDependency: false,
25         // 是否要解析标准库的包
26         ParseInternal: false,
27         // 是否要查找markdown文件，这个markdown文件能用来为tag增加说明格式
28         MarkdownFilesDir: "",
29         // 是否应该在docs.go中生成时间戳
30         GeneratedTime: false,
31     }
32     err := gen.New().Build(conf)
33     if err != nil {
34         fmt.Println(err)
35     }
36 },
37 }
```

结合这个具体实现，我们来看这个 Config 结构的关键字段 SearchDir、OutputDir 和 MainAPIFile，这几个字段的含义必须完全理解才能设置正确，其他的几个字段如果不理解，直接使用默认值就行。

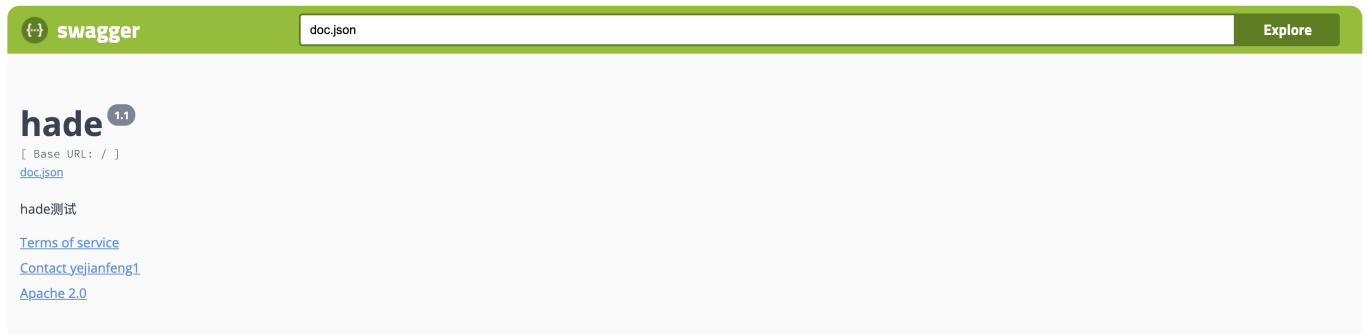
**第一个 SearchDir 表示要 swaggo 去哪个目录遍历代码的注释，来生成 swagger 的 JSON 文件。**对于我们的 hade 框架，所有接口文件都存放在 app/http 文件夹中，所以要遍历的就是这个文件夹了。

第二个关键字段 OutputDir，表示要输出的 swagger 文件的存放地址。我们在 app/http 目录下，创建一个 swagger 目录，来存放要输出的 swagger 文件。这里再补充一点，前面说 swagger 最终会生成 JSON 文件，但是你运行一次 swagger gen 会发现，这个生成目录下除了有 swagger.json 这个文件，还有两个文件 swagger.yaml 和 docs.go。

对于额外生成的这两个文件，swagger.yaml 是 YAML 格式的接口说明文档，里面的内容和 swagger.json 其实是一样的。而 docs.go 是“接口说明文档”的代码，它是为 go 项目直接引入接口说明文档生成 swagger-ui 用的。

也就是说生成的 docs.go，我们的框架只需要 import 它，就能从这个文件的变量 doc 中直接获取到“接口说明文档”，不需要用读取文件的方式读取 swagger.json 或者 swagger.yaml。下一节课我们会使用它来让框架启动服务的时候自动启动 swagger-ui。

**第三个关键字段 MainAPIFile，表示整个 swagger 接口的说明文档。**这是什么意思呢？在最终生成的 swagger-ui 界面中的头部，你会看到对当前 swagger 接口的整体说明，包括作者、接口版本、接口 licence 等信息。



这些信息也都是使用注释来自动生成的，而这一部分注释，就是存放在这个 MainApiFile 所指向的 Go 文件中。

我们的项目就固定将这个文件命名为 swagger.go，存放在 app/http/swagger.go 文件中。这个文件只是增加注释，不增加任何的逻辑。其中的每个注释的关键字说明，也是参考 swaggo 的 [说明文档](#)。

[复制代码](#)

```
1 // Package http API.
2 // @title hade
3 // @version 1.1
4 // @description hade测试
5 // @termsOfService https://github.com/swaggo/swag
6
7 // @contact.name yejianfeng1
8 // @contact.email yejianfeng
9
10 // @license.name Apache 2.0
11 // @license.url http://www.apache.org/licenses/LICENSE-2.0.html
12
13 // @BasePath /
14 // @query.collection.format multi
15
16 // @securityDefinitions.basic BasicAuth
17
18 // @securityDefinitions.apikey ApiKeyAuth
```



```
19 // @in header
20 // @name Authorization
21
22 // @x-extension-openapi {"example": "value on a json format"}
23
24 package http
```

现在按照前面的说明，我们设置好了 Config 结构，在 swaggerGenCommand 命令逻辑的最后，只需调用一次 Build 方法，就能按照 Config 配置来生成 docs.go、swagger.json、swagger.yaml 这三个文件了。

```
1 err := gen.New().Build(conf)
```

[复制代码](#)

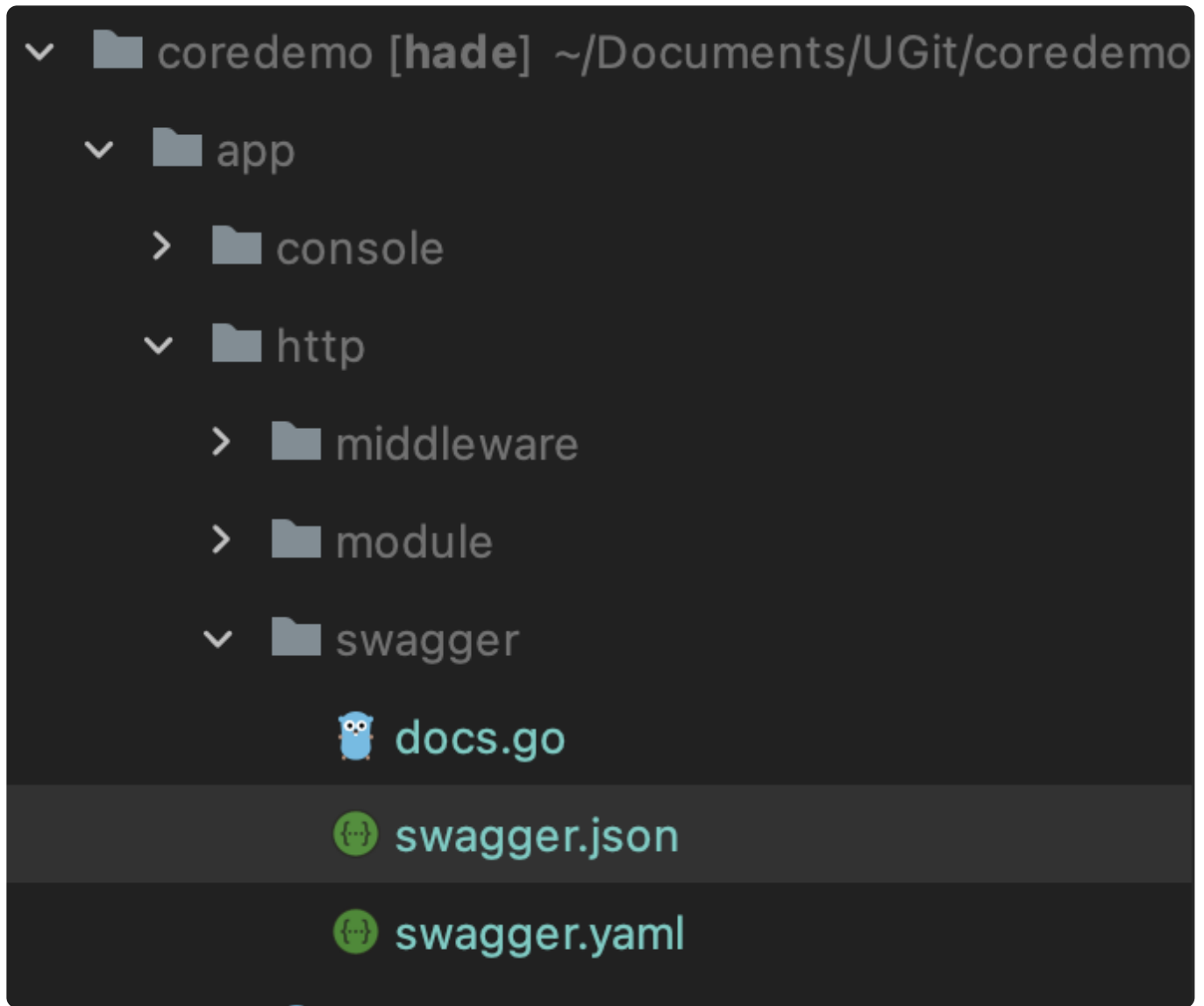
记得将这个二级命令 ./hade swagger gen 挂载到一级命令 ./hade swagger 下，并且挂载到 framework/command/kernel.go 中。这个步骤，前面几章中都已经重复做过了，这里就不赘述了。

挂载好之后，我们尝试运行一下：

```
~/Documents/UGit/coredemo geekbang/23 ● ./hade swagger gen
2021/10/18 10:08:24 Generate swagger docs....
2021/10/18 10:08:24 Generate general API Info, search dir:/Users/yejianfeng/Documents/UGit/coredemo/app/http
2021/10/18 10:08:25 Generating demo.UserDTO
2021/10/18 10:08:25 create docs.go at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/docs.go
2021/10/18 10:08:25 create swagger.json at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/swagger.json
2021/10/18 10:08:25 create swagger.yaml at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/swagger.yaml
```

可以看到，日志信息打印非常详细，包括去哪个目录查找、最终生成哪些文件。

查看 app/http/swagger 目录，确实最终生成了 docs.go、swagger.json、swagger.yaml 这三个文件：




## 启动 swagger-ui

有了 swagger 生成文件了，应该怎么使用它呢？还是先设想，我们希望的是，能在启动服务的时候，同时启动一个 swagger-ui 页面，给接口使用人员来查看服务接口，并且他们可以直接在这个页面进行接口调用。

到这里相信你已经想到了，在启动服务的时候，**增加一个打开 swagger-ui 页面路由**，就可以达到我们的目的了。还是查看 swaggo 这个项目，[🔗 官方文档](#)中有一段如何将 swaggo 结合 Gin 来生成路由的方法说明，正好我们框架的路由用的 Gin，所以就考虑使用这个方法来开辟一个 swagger-ui 路由。

swaggo 开发了一个 [🔗 gin-swagger](#) 中间件，来为 Gin 框架增加路由设置。怎么使用它呢？看官方文档的这个 [🔗 例子](#)：

 复制代码

```
1 package main
2
3 import (
4     "github.com/gin-gonic/gin"
5     docs "github.com/go-project-name/docs"
6     swaggerfiles "github.com/swaggo/files"
7     ginSwagger "github.com/swaggo/gin-swagger"
8     "net/http"
9 )
10 // @BasePath /api/v1
11
12 // PingExample godoc
13 // @Summary ping example
14 // @Schemes
15 // @Description do ping
16 // @Tags example
17 // @Accept json
18 // @Produce json
19 // @Success 200 {string} Helloworld
20 // @Router /example/helloworld [get]
21 func Helloworld(g *gin.Context) {
22     g.JSON(http.StatusOK, "helloworld")
23 }
24
25 func main() {
26     r := gin.Default()
27     docs.SwaggerInfo.BasePath = "/api/v1"
28     v1 := r.Group("/api/v1")
29     {
30         eg := v1.Group("/example")
31         {
32             eg.GET("/helloworld", Helloworld)
33         }
34     }
35     r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))
36     r.Run(":8080")
37 }
```

## gin-swagger 原理分析

我们着重注意下这几行：

 复制代码

```
1 package main
2
3 import (
4     ...
5     docs "github.com/go-project-name/docs"
```

```
6     swaggerfiles "github.com/swaggo/gin-swagger/swaggerFiles"
7     ginSwagger "github.com/swaggo/gin-swagger"
8     ...
9 )
10 ...
11
12 func main() {
13     ...
14     r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))
15     ...
16 }
```

首先，import 的三个文件分别是做什么用的，必须要理解清楚。

swagger-ui 是一个 HTML + JSON 接口的页面，那么里面分别有动态内容和静态内容，静态内容包括 HTML、JS、CSS、png 等，动态内容包括 swagger JSON 化的结构。也就是说我们现在的**目标是要创建一个包含 HTML+JSON 的服务**，如何做呢？

一种方法当然是将这些静态文件直接放在项目中，在服务启动的时候，使用读取文件的方式并返回这些静态文件提供服务。但是这就要求在发布的时候，这些静态文件，必须同时被带着上线，它们成为了服务必须的一部分，特别是作为一个类库提供的时候，如果要求使用者必须带着库的静态文件，是非常不方便的。

而这里 gin-swagger 采用了另外一种更为极致的做法，是**将这些静态文件代码化，嵌入到 go 代码中**，比如让一个变量返回 HTML 的内容，我们在提供获取 HTML 页面的服务时，直接将变量返回就可以了。

这里代码第 6 行的 `github.com/swaggo/gin-swagger/swaggerFiles`，这个库就做了这个事情，它将 swagger-ui 的所有 HTML、JS、CSS、png 文件都变化成为了 go 文件，并且作为 HTTP 服务提供出来。其中的 `swaggerfiles.Handler` 就是实现了 `net/http` 的 [HandlerFunc 接口](#)。

而另外一部分，动态 JSON 接口，返回的是具体的 swagger JSON 化的内容。这个怎么获取呢？

是通过前面我们说的 swaggo 生成的几个文件中的 `doc.go` 文件来获取的，在例子中就是第 5 行引入的 `github.com/go-project-name/docs` 库，它的原理就是生成 doc 全局变量，并且通过 `ReadDoc()` 方法来提供 JSON 的内容读取。

好，现在有了动态 JSON 接口和静态文件服务接口，如何集成到 Gin 的 Engine 里呢？

要一个中间件就行了，就是第 7 行 import 中引入的 `github.com/swaggo/gin-swagger` 库。它通过创建一个 Gin 的中间件，将动态和静态的请求都承接起来，静态请求就请求到 `swaggo/files` 库，动态请求就请求到 `docs` 库中。

所以在路由中，我们 \*创建一个路由 `/swagger/any`，就可以获取 `swagger-ui` 并且读取 `swaggo` 创建的 `doc.go` 文件内容了。

```
1 r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))
```

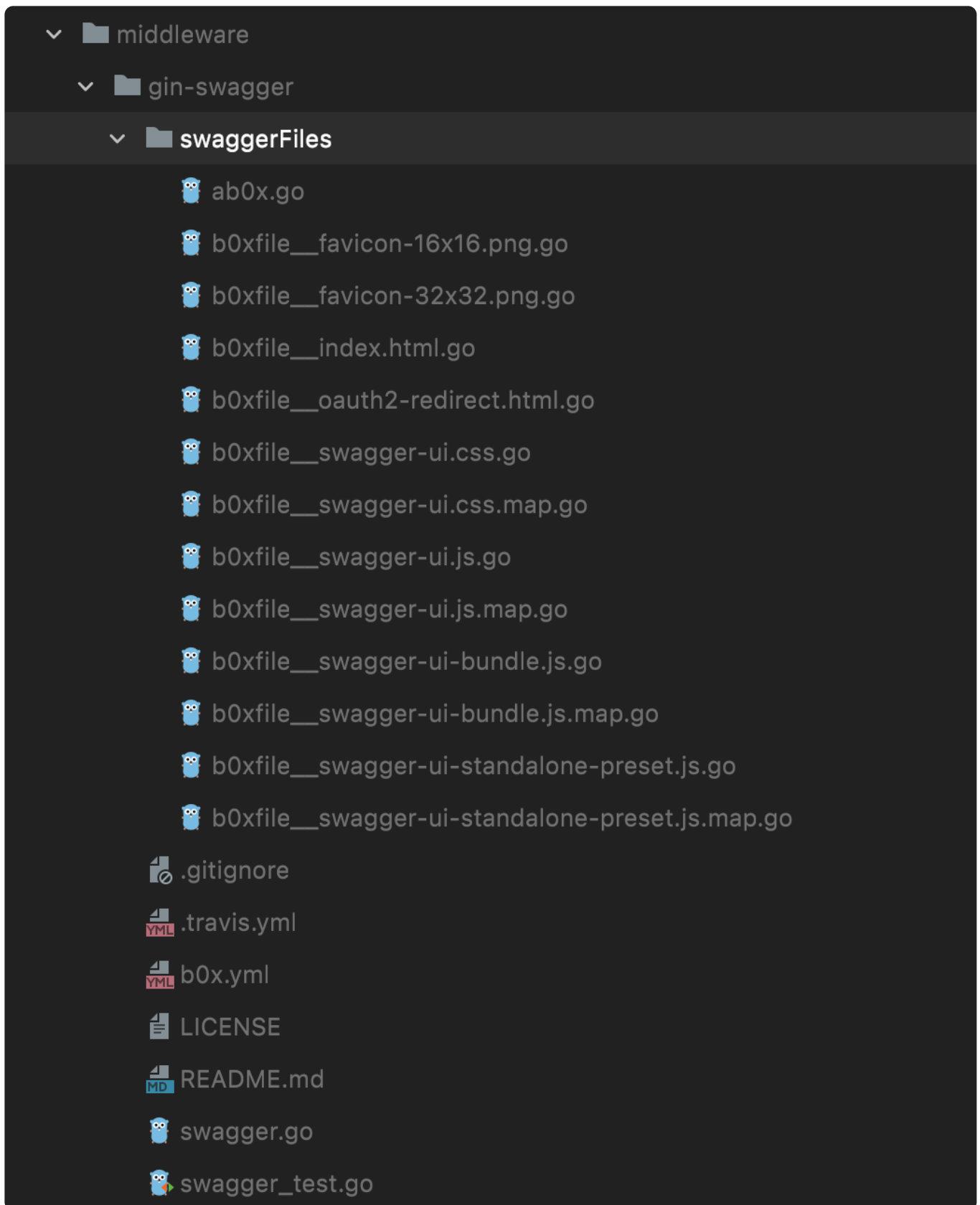
[复制代码](#)

## 如何集成

好，gin-swagger 的原理理解清楚了，如何集成进入我们框架呢？

gin-swagger 本质就是一个 Gin 中间件。集成 Gin 的中间件，我们只需要拷贝这个中间件源码，并且将中间件的 `github.com/gin-gonic/gin` 替换为 hade 框架的 gin 地址：`github.com/gohade/hade/framework/gin` 就可以了。


所以把这个中间件放在 `framework/middleware/gin-swagger` 下。存放完之后的目录截图，你可以对比检查一下：



把 gin-swagger 中间件处理完，就剩下将路由存放到我们的 app 业务路由中去了。

这里再设计一个小细节。毕竟我们其实并不希望线上服务也提供这么一个 swagger 路由，也就是说只希望 swagger-ui 在测试和开发环境使用，所以可以在配置文件 app.yml 中有这么一个配置项：




 复制代码

```
1 swagger: true
```

来表示是否开启这个 swagger 路由。

那在应用路由中如何获取到这个配置呢？之前应用路由中的参数只有一个 gin.Engine。我们需要为 gin.Engine 增加一个获取服务容器的接口，GetContainer()。来修改 framework/gin/hade\_engine.go：

 复制代码

```
1 // GetContainer 从Engine中获取container
2 func (engine *Engine) GetContainer() framework.Container {
3     return engine.container
4 }
```


现在就是真正的万事俱备了，我们来改造应用路由 app/http/route.go。

首先要引入 gin-swagger 提示的三个 import。

这里我将最后一个 docs 对应的 import，放在了同级目录的 app/http/swagger.go 文件中。

我是这么考虑的，**docs.go 是我们用命令行生成的，而生成的时候 swagger 的全局说明配置是放在 swagger.go 中的**，所以这两个文件关系更为紧密，比较适合放在一起。

app/http/swagger.go 文件信息：


 复制代码

```
1 // Package http API.
2 // @title hade
3 // @version 1.1
4 // @description hade测试
5 // @termsOfService https://github.com/swaggo/swag
6 ...
7 package http
8
9 import (
10     _ "github.com/gohade/hade/app/http/swagger"
```

```
11 )
```

回到 `app/http/route.go` 中。我们引入了另外两个库，并且先判断 `app.swagger` 配置项是否为 `true`，如果为 `true`，则开启 `swagger` 路由。

`app/http/route.go` 代码实现，你应该能很容易写出来。要点刚才都详细讲过了：

 复制代码

```
1 package http
2
3 import (
4     ...
5     ginSwagger "github.com/gohade/hade/framework/middleware/gin-swagger"
6     "github.com/gohade/hade/framework/middleware/gin-swagger/swaggerFiles"
7     ...
8 )
9
10 // Routes 绑定业务层路由
11 func Routes(r *gin.Engine) {
12     container := r.GetContainer()
13     configService := container.MustMake(contract.ConfigKey).(contract.Config)
14
15     ...
16
17     // 如果配置了swagger，则显示swagger的中间件
18     if configService.GetBool("app.swagger") == true {
19         r.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerFiles.Handler))
20     }
21
22     ...
23 }
```

到这里我们就完美将 `swagger-ui` 集成进入我们服务了。

## 验证

最后做一下验证。先用 `./hade swagger gen` 命令生成我们要的 `docs.go` 文件：

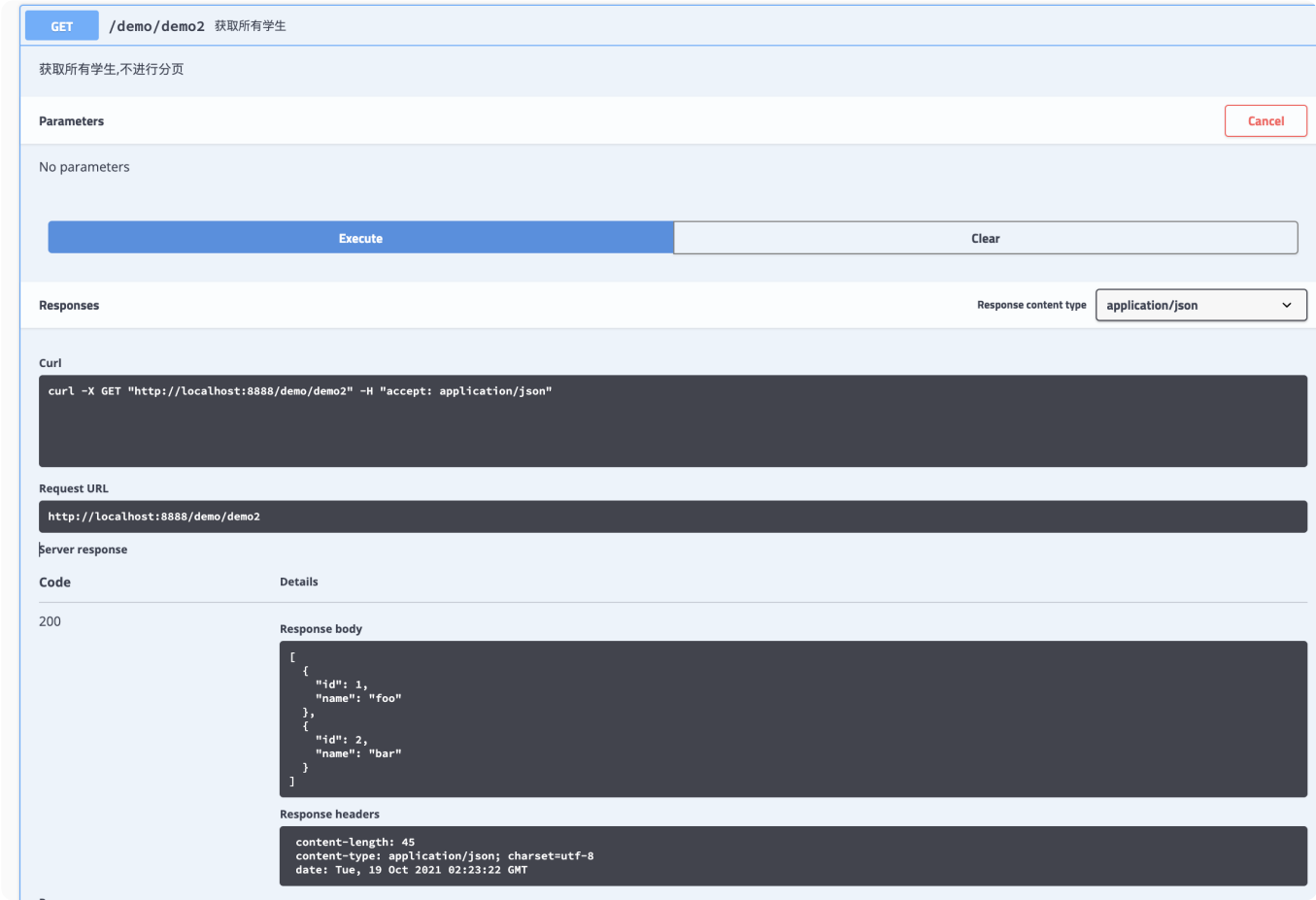
```
~/Documents/UGit/coredemo geekbang/23 ● ./hade swagger gen
2021/10/19 10:18:07 Generate swagger docs....
2021/10/19 10:18:07 Generate general API Info, search dir:/Users/yejianfeng/Documents/UGit/coredemo/app/http
2021/10/19 10:18:07 Generating demo.UserDTO
2021/10/19 10:18:07 create docs.go at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/docs.go
2021/10/19 10:18:07 create swagger.json at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/swagger.json
2021/10/19 10:18:07 create swagger.yaml at /Users/yejianfeng/Documents/UGit/coredemo/app/http/swagger/swagger.yaml
```

再使用 `./hade build self`，将生成的 docs.go 打包编译进 ./hade 命令，启动服务  
`./hade app start`：

```
~/Documents/UGit/coredemo geekbang/23 ● ./hade app start
[PID] 5251
app serve url: :8888
```

浏览器打开地址 <http://localhost:8888/swagger/index.html>，可以看到整个 swagger-ui 界面：

并且点击某个接口的 `execute` 按钮，可以真实地调用这个接口，返回返回数据，进行测试。非常方便：



验证完成！

今天所有代码都保存在 GitHub 上的 [@geekbang/23](#)分支了。附上目录结构供你对比查看。

```
▼ coredemo [hade] ~/Documents/UGit/coredemo
  ▼ app
    > console
  ▼ http
    > middleware
    > module
  ▼ swagger
    🦉 docs.go
    📄 swagger.json
    📄 swagger.yaml
    🦉 kernel.go
    🦉 route.go
    🦉 swagger.go
  > provider
  > build
  > config
```

- ▼ framework
  - > cobra
  - > command
  - > contract
  - > gin
- ▼ middleware
  - ▼ gin-swagger
    - ▼ swaggerFiles
      - 🦉 ab0x.go
      - 🦉 b0xfile\_\_favicon-16x16.png.go
      - 🦉 b0xfile\_\_favicon-32x32.png.go
      - 🦉 b0xfile\_\_index.html.go
      - 🦉 b0xfile\_\_oauth2-redirect.html.go
      - 🦉 b0xfile\_\_swagger-ui.css.go
      - 🦉 b0xfile\_\_swagger-ui.css.map.go
      - 🦉 b0xfile\_\_swagger-ui.js.go
      - 🦉 b0xfile\_\_swagger-ui.js.map.go
      - 🦉 b0xfile\_\_swagger-ui-bundle.js.go
      - 🦉 b0xfile\_\_swagger-ui-bundle.js.map.go
      - 🦉 b0xfile\_\_swagger-ui-standalone-preset.js.go
      - 🦉 b0xfile\_\_swagger-ui-standalone-preset.js.map.go
    - 📄 .gitignore
    - 📄 .travis.yml
    - 📄 b0x.yml
    - 📄 LICENSE
    - 📄 README.md
    - 🦉 swagger.go
    - 🦉 swagger\_test.go

## 小结



这一节课，我们其实就做了一件事情：将 swagger 融合进入 hade 框架。

我们依赖 swag 项目和 gin-swagger 中间件，成功地将 swagger 放到 hade 框架中，之后使用一个配置，能同时启动 hade 后端服务和 swagger 前端调试工具，自动生成一个可以查看接口、可以调用执行的页面。相信在实际工作中开发过后端接口的同学就知道这个工具是有多实用。

当然熟练使用 swagger，以及熟练编写 swagger 的代码注释，需要对 swagger 的规则和 swag 的注释定义有一定了解，这个需要你花时间去掌握。但是相信我，虽然写 swagger 注释有一些繁琐，但是它能节省大量你和前端同学联调的时间。

## 思考题

我之前在一个项目中使用 swagger 的 JSON 文件自动生成了项目的接口 word 说明文档。不知道你在实际工作中，是如何使用 swagger 的呢？能分享一下你 / 你们公司使用 swagger 的一些经历么？

欢迎在留言区分享你的思考。感谢你的收听，如果觉得今天的内容对你有所帮助，也欢迎分享给你身边的朋友，邀请他一起学习。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 22 | 自动化：DRY，如何自动化一切重复性劳动？（下）

下一篇 24 | 管理进程：如何设计完善的运行命令？

训练营推荐

# Java 学习包免费领<sup>NEW</sup>

面试题答案均由大厂工程师整理

阿里、美团等  
大厂真题

18 大知识点  
专项练习

大厂面试  
流程解析

可复用的  
面试方法

面试前  
要做的准备

## 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。