



下载APP



## 22 | 自动化：DRY，如何自动化一切重复性劳动？（下）

2021-11-05 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 15:12 大小 13.93M



你好，我是轩脉刃。

上一节课我们增加了自动化创建服务工具、命令行工具，以及中间件迁移工具。你会发现，这些工具实现起来并不复杂，但是在实际工作中却非常有用。今天我们继续思考还能做点什么。

我们的框架是定义了业务的目录结构的，每次创建一个新的应用，都需要将 AppServ 中定义的目录结构创建好，如果这个行为能自动化，实现一个命令就能创建一个定义好所有目录结构，甚至有 demo 示例的新应用呢？是不是有点心动，这就是我们今天要实现的工具了，听起来功能有点庞大，所以我们还是慢慢来，先设计再实现。



### 初始化脚手架设计

这个功能倒不是什么新想法, 有用过 Vue 的同学就知道, Vue 官网有介绍一个 `vue create` 命令, 可以从零开始创建一个包含基本 Vue 结构的目录, 这个目录可以直接编译运行。

在初始化一个 Vue 项目的时候, 大多数刚接触 Vue 的同学对框架的若干文件还不熟悉, 很容易建立错误 vue 的目录结构, 而这个工具能帮 Vue 新手们有效规避这种错误。

同理, 我们的框架也有基本的 hade 结构的目录, 初学者在创建 hade 应用的时候, 也大概率容易建立错误目录。所以参考这一点, 让自己的框架也有这么一个命令, 能直接创建一个新的包含 hade 框架业务脚手架目录的命令。这样, 能很大程度方便使用者就在这个脚手架目录上不断开发, 完成所需的业务功能。

我们要设计的命令是一个一级命令 `./hade new`。一般来说, 新建命令创建一个脚手架, 要做的事情就是:

- 确定目标目录, 如果没有就创建目录

- 创建业务模块目录

- 初始化 go module 模块, 补充模块名称、框架版本号

- 在业务模块目录中创建对应的文件代码

我们跟着这个思路走。先梳理一下在这个命令中, 要传入的参数有哪些?

首先是目录, 在控制台目录之下要创建一个子目录, 这个**子目录的名称, 是需要用户传递进入的**。不过, 这个参数记得做一下验证, 如果子目录已经存在了, 给用户一个提示, 是直接删除原先的子目录? 还是停止操作? 如果用户需要删除原先的子目录, 我们就直接删除。

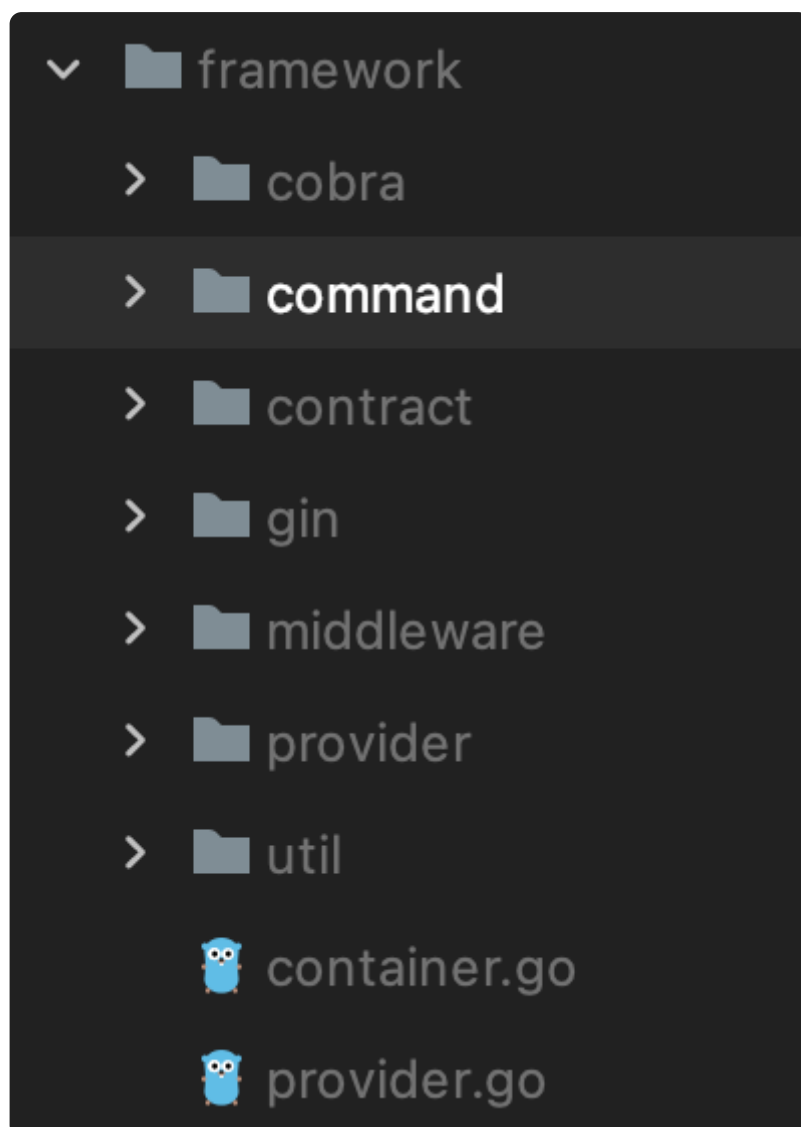
其次是**需要用户传入新应用的模块名称**, 也就是 go.mod 中的 module 后面的名称, 一般会设置为应用的项目地址, 比如 `github.com/jianfengye/testdemo`。关于模块名称, 我们要详细做一下解说。

## 业务、框架模块地址

一直到这一节课的 GitHub 地址, 不知道你有没有疑惑, 别的框架, 比如 Gin、Echo, 都是把框架代码放在 GitHub 上, 比如 `github/gin-gonic/gin`, 而业务代码是单独存放的。但我们这个项目 `github.com/gohade/coredemo`, 却是把业务代码和框架代码都放在一个项目中?

其实是这样, 这个项目 `github.com/gohade/coredemo`, 是我为 `geekbang` 这个课程单独设置的项目, 将 `hade` 框架的每个实现步骤, 重新在这个项目做了一次还原。而 `github.com/gohade/hade` 才是我们最终的项目地址。所以不管在 `coredemo` 这个项目还是 `hade` 这个项目, `go.mod` 中的 `module` 都是叫做 `github.com/gohade/hade`。

**但是即使是最终的 `github.com/gohade/hade` 项目, 我们的业务代码 `app` 目录和框架目录 `framework` 目录也是在一个项目里的**, 按道理说在这个 `hade` 项目中, 应该只有 `framework` 目录的内容即可啊?




这里我是这么设计的, 将 framework 目录和其他的业务目录都同时放在 `github.com/gohade/hade` 项目中, 这样这个项目也同时就是我们 hade 框架的一个示例项目。只是这个项目带着 framework 目录而已。

后续如果要创建一个新的业务项目, 比如 `github.com/jianfengye/testdemo`。我们**不是**做加法把业务文件夹一点点复制过来, 而是做依赖这个 `github.com/gohade/hade` 项目做减法, 把不必要的文件夹(比如框架文件夹)删掉。

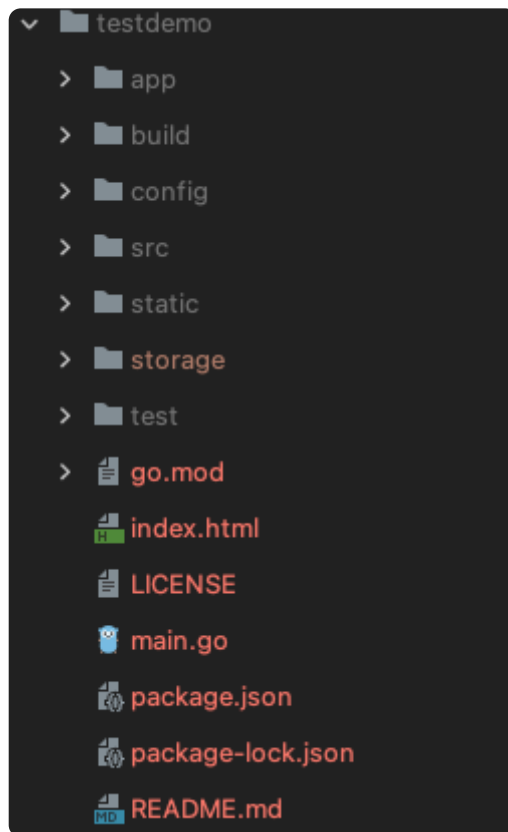
即我们只需要直接拷贝这个 `github.com/gohade/hade` 项目, 并且将其中的 framework 目录删除, 保留业务目录, 同时把 `go.mod` 中的原先的 “`github.com/gohade/hade`” 模块名修改为 `github.com/jianfengye/testdemo` 这个模块名, 用到 hade 框架的部分直接引用 “`github.com/gohade/hade/framework`” 即可。

这就是说, 如果你要创建的项目的模块名为 `github.com/jianfengye/testdemo`, `go.mod` 应该如下:

 复制代码

```
1 // 这里是你的模块地址
2 module github.com/jianfengye/testdemo
3
4 go 1.15
5
6 require (
7     // 这里引用github.com/gohade/hade
8     github.com/gohade/hade v0.0.2
9     ...
10
11 )
```

目录应该和 `github.com/gohade/hade` 只有一处不同: 没有 framework 目录。



而在你自己的 `github.com/jianfengye/testdemo` 项目中的所有文件, 如果是框架中的, 也就是要使用 `hade` 已有的服务提供者、中间件、命令行的时候, 是使用 `import github.com/gohade/hade/framework`; 而在使用自己的服务提供者、中间件、命令行, 所有在业务目录内的结构的时候, 是使用 `import github.com/jianfengye/testdemo/xxx`。

比如 `main.go` 就形如:

 复制代码

```
1 package main
2
3 import (
4     // 业务的目录app内的文件
5     "github.com/jianfengye/testdemo/app/console"
6     "github.com/jianfengye/testdemo/app/http"
7     // 框架目录的文件
8     "github.com/gohade/hade/framework"
9     "github.com/gohade/hade/framework/provider/app"
10    "github.com/gohade/hade/framework/provider/config"
11    "github.com/gohade/hade/framework/provider/distributed"
12    "github.com/gohade/hade/framework/provider/env"
13    "github.com/gohade/hade/framework/provider/id"
14    "github.com/gohade/hade/framework/provider/kernel"
15    "github.com/gohade/hade/framework/provider/log"
16    "github.com/gohade/hade/framework/provider/trace"
```

```
17 )
18
19 func main() {
20     // 初始化服务容器
21     container := framework.NewHadeContainer()
22     // 绑定App服务提供者
23     container.Bind(&app.HadeAppProvider{})
24     // 后续初始化需要绑定的服务提供者...
25     container.Bind(&env.HadeEnvProvider{})
26     container.Bind(&distributed.LocalDistributedProvider{})
27     container.Bind(&config.HadeConfigProvider{})
28     container.Bind(&id.HadeIDProvider{})
29     container.Bind(&trace.HadeTraceProvider{})
30     container.Bind(&log.HadeLogServiceProvider{})
31
32     // 将HTTP引擎初始化,并且作为服务提供者绑定到服务容器中
33     if engine, err := http.NewHttpEngine(); err == nil {
34         container.Bind(&kernel.HadeKernelProvider{HttpEngine: engine})
35     }
36
37     // 运行root命令
38     console.RunCommand(container)
39 }
```

说到这里相信你应该理解了，最终我们这个框架只维护 [github.com/gohade/hade](https://github.com/gohade/hade) 这么一个项目，**这个项目中的 framework 目录，存放的是框架所有的代码，而 framework 之外的目录和文件都是示例代码。**

所以，回到今天的主题，让 `./hade new` 命令创建一个脚手架，要做的事情现在就变成了：

下载 [github.com/gohade/hade](https://github.com/gohade/hade) 项目到目标文件夹

删除 framework 目录

修改 go.mod 中的模块名称

修改 go.mod 中的 require 信息，增加 require [github.com/gohade/hade](https://github.com/gohade/hade)

修改所有文件使用业务目录的地方，将原本使用 “[github.com/gohade/hade](https://github.com/gohade/hade)/app” 的所有引用改成 “[模块名称]/app”

也就是说第二个输入，我们需要用户确切输入一个模块名称。

## 框架的版本号信息

除了新建时必须的子目录的名称和新建模块的名称，第三个需要用户输入的是 hade 的版本号。

我们的 hade 框架是会不断变化的，和 Golang 语言一样，使用形如 v1.2.3 这样的版本号进行迭代，v 代表版本的英文缩写，1 代表的是大版本，只有非常大变更的时候我们才会更新这个版本；2 代表的是小版本，有接口变更或者类库变更之类的时候我们会迭代这个版本；3 代表的是补丁版本，如果发现有需要补丁修复的地方，就会使用这个版本。

而每个 hade 框架版本对应的脚手架，也有可能有一定变化的。因为在脚手架中，我们会把框架的使用示例等放在应用代码中。

hade 框架的每个版本发布时，都会打对应的 tag，每个 tag 我们都会在 GitHub 上发布一个 release 版本与之对应，比如截止到 10/7 日，已经发布了 v0.0.1 和 v0.0.2 两个 tag 和 release 版本，你可以直接通过 [🔗 GitHub 地址](#)来进行查看。

---

Releases	Tags
Tags	
v0.0.2 ...	
🕒 yesterday 🔗 e1a7d74 📦 zip 📦 tar.gz 📄 Notes	
v0.0.1 ...	
🕒 on 14 Dec 2020 🔗 d7120a2 📦 zip 📦 tar.gz 📄 Notes	





所以回到 `./hade new` 命令，第三个需要用户输入的就是这个版本号，如果用户需要创建一个 v0.0.1 版本的 hade 脚手架，则需要输入 v0.0.1，如果用户没有输入，我们默认使用最新的版本。

好了，简单总结一下，用户目前输入的三个信息：

目录名，最终是“当前执行目录 + 目录名”

模块名，最终创建应用的 module

版本号，对应的 hade 的 release 版本号

用户输入相关的代码如下，在我们的 `framework/command/new.go` 中：

```
1  var name string
2  var folder string
3  var mod string
4  var version string
5  var release *github.RepositoryRelease
6  {
```

复制代码



```
7      prompt := &survey.Input{
8          Message: "请输入目录名称:",
9      }
10     err := survey.AskOne(prompt, &name)
11     if err != nil {
12         return err
13     }
14
15     folder = filepath.Join(currentPath, name)
16     if util.Exists(folder) {
17         isForce := false
18         prompt2 := &survey.Confirm{
19             Message: "目录" + folder + "已经存在,是否删除重新创建?(确认后立刻执行)",
20             Default: false,
21         }
22         err := survey.AskOne(prompt2, &isForce)
23         if err != nil {
24             return err
25         }
26
27         if isForce {
28             if err := os.RemoveAll(folder); err != nil {
29                 return err
30             }
31         } else {
32             fmt.Println("目录已存在, 创建应用失败")
33             return nil
34         }
35     }
36 }
37 {
38     prompt := &survey.Input{
39         Message: "请输入模块名称(go.mod中的module, 默认为文件夹名称):",
40     }
41     err := survey.AskOne(prompt, &mod)
42     if err != nil {
43         return err
44     }
45     if mod == "" {
46         mod = name
47     }
48 }
49 {
50     // 获取hade的版本
51     client := github.NewClient(nil)
52     prompt := &survey.Input{
53         Message: "请输入版本名称(参考 https://github.com/gohade/hade/releases
54     )
55     err := survey.AskOne(prompt, &version)
56     if err != nil {
57         return err
58     }
```

```
59         if version != "" {
60             // 确认版本是否正确
61             release, _, err = client.Repositories.GetReleaseByTag(context.Back
62             if err != nil || release == nil {
63                 fmt.Println("版本不存在，创建应用失败，请参考 https://github.com/goh
64                 return nil
65             }
66         }
67         if version == "" {
68             release, _, err = client.Repositories.GetLatestRelease(context.Bac
69             version = release.GetTagName()
70         }
71     }
```

## 初始化脚手架具体实现

有了这三个信息，我们将之前讨论的 hade new 命令的步骤再详细展开讨论：

下载 [github.com/gohade/hade](https://github.com/gohade/hade) 项目到目标文件夹

删除 framework 目录

修改 go.mod 中的模块名称

修改 go.mod 中的 require 信息，增加 require [github.com/gohade/hade](https://github.com/gohade/hade)

修改所有文件使用业务目录的地方，将原本使用 “[github.com/gohade/hade/app](https://github.com/gohade/hade/app)” 的所有引用改成 “[模块名称]/app”

第一步下载稍微复杂一点，我们重点说，剩下四步就是简单的按部就班了。

## 项目下载

因为有版本号更新的可能，其中的第一步 “复制 [github.com/gohade/hade](https://github.com/gohade/hade) 项目到目标文件夹” ，我们就要变化为 “下载 [github.com/gohade/hade](https://github.com/gohade/hade) 的某个 release 版本到目标文件夹” 。

这个能怎么做呢？可以想到 GitHub 有提供对外的开放平台接口 [api.github.com](https://api.github.com)，你可以看它的 [🔗 官方文档地址](#)。

我们可以通过开放平台接口，对公共的 GitHub 仓库进行信息查询。比如要查看某个 GitHub 仓库的 release 分支，可以通过调用 “[🔗 /repos/{owner}/{repo}/releases](#)” ，而

获取某个 GitHub 仓库的最新 release 分支, 可以通过调用 “[@/repos/{owner}/{repo}/releases/latest](#)”。

使用 GitHub 的开放平台接口, 是可以直接调用, 但是这个方法有个明显的问题, 我们还要手动封装这个接口调用。

其实更简单的方式是, 使用 Google 给我们提供好的 Golang 语言的 SDK, [@go-github](#)。这个库本质就是封装了 GitHub 的调用接口。比如获取仓库 [github.com/gohade/hade](#) 的 release 分支:

[复制代码](#)

```
1 client := github.NewClient(nil)
2 releases, _, err = client.Repositories.GetReleases(context.Background(), "goha
```

而获取它最新 release 分支也很简单:

[复制代码](#)

```
1 client := github.NewClient(nil)
2 release, _, err = client.Repositories.GetLatestRelease(context.Background(), "
```


在返回的 RepositoryRelease 结构中, 我们可以找到下载这个 release 版本的各种信息。其中包括 release 版本对应的版本号信息和 zip 下载地址:

[复制代码](#)

```
1 // RepositoryRelease represents a GitHub release in a repository.
2 type RepositoryRelease struct {
3     // 对应的版本号信息
4     TagName          *string `json:"tag_name,omitempty"`
5     ...
6     // release版本的zip下载地址
7     ZipballURL       *string `json:"zipball_url,omitempty"`
8
9     ...
10 }
```

库信息了解到这里，我们回到刚才要执行的第一步“下载 `github.com/gohade/hade` 的某个 release 版本到目标文件夹”，就可以使用这个 zip 下载地址，下载对应的 zip 包，并且使用 `unzip` 解压这个 zip 目录。


对于下载 zip 包，直接使用 `http.Get` 就能下载了。这个函数我们封装在 `framework/util/file.go` 中：

 复制代码

```
1 // DownloadFile 下载url中的内容保存到本地的filepath中
2 func DownloadFile(filepath string, url string) error {
3
4     // 获取
5     resp, err := http.Get(url)
6     if err != nil {
7         return err
8     }
9     defer resp.Body.Close()
10
11     // 创建目标文件
12     out, err := os.Create(filepath)
13     if err != nil {
14         return err
15     }
16     defer out.Close()
17
18     // 拷贝内容
19     _, err = io.Copy(out, resp.Body)
20     return err
21 }
```

而 `unzip` 解压，我们可以使用 Golang 标准库的 `archive/zip`，来读取 zip 包中的内容，然后将每个文件都复制到目标目录中。`unzip` 的基本逻辑就是使用 zip 包读取压缩文件，然后遍历压缩文件中的文件夹，将对应的文件和文件夹都复制到目标目录中。

具体代码存放在 `framework/util/zip.go` 中，代码中也做了对应注释：

 复制代码

```
1 // Unzip 解压缩zip文件，复制文件和目录都到目标目录中
2 func Unzip(src string, dest string) ([]string, error) {
3
4     var filenames []string
5
```

```
6 // 使用archive/zip读取
7 r, err := zip.OpenReader(src)
8 if err != nil {
9     return filenames, err
10 }
11 defer r.Close()
12
13 // 所有内部文件都读取
14 for _, f := range r.File {
15
16     // 目标路径
17     fpath := filepath.Join(dest, f.Name)
18
19     if !strings.HasPrefix(fpath, filepath.Clean(dest)+string(os.PathSeparato
20         return filenames, fmt.Errorf("%s: illegal file path", fpath)
21     }
22
23     filenames = append(filenames, fpath)
24
25     if f.FileInfo().IsDir() {
26         // 如果是目录, 则创建目录
27         os.MkdirAll(fpath, os.ModePerm)
28         continue
29     }
30
31     //否则创建文件
32     if err = os.MkdirAll(filepath.Dir(fpath), os.ModePerm); err != nil {
33         return filenames, err
34     }
35
36     outFile, err := os.OpenFile(fpath, os.O_WRONLY|os.O_CREATE|os.O_TRUNC, f
37     if err != nil {
38         return filenames, err
39     }
40
41     rc, err := f.Open()
42     if err != nil {
43         return filenames, err
44     }
45
46     // 复制内容
47     _, err = io.Copy(outFile, rc)
48
49
50     outFile.Close()
51     rc.Close()
52
53     if err != nil {
54         return filenames, err
55     }
56 }
57 return filenames, nil
```


但是你在调试的过程中就会发现, 下载的 zip 包中带有一层目录, gohade-hade-xxxx, 目录下面才是我们需要的 hade 库的真实代码。如果直接复制 zip 包, 就会在目标文件夹下创建 gohade-hade-xxx 目录, 但是这个目录层级并不是我们想要的。

所以这里要修改 “下载 github.com/gohade/hade 的某个 release 版本到目标文件夹” 的实现步骤, 大致思路就是**通过创建和删除一个临时目录, 来达到把 zip 包解压的目的**。

具体操作就是, 先创建临时目录 template-hade-version-[timestamp], 然后下载 release 的 zip 包地址临时目录, 并命名为 template.zip, 在临时目录中解压 zip 包 template.zip, 生成 gohade-hade-xxxx 目录。这个时候就完成了一半, 拿到了需要的 hade 库真实代码。

之后, 查找临时目录中名为 gohade-hade- 开头的目录, 定位到 gohade-hade-xxx 目录, 将这个目录使用 os.rename 移动成为目标文件夹。最后收尾删除临时目录。

对应代码在 framework/command/new.go 中:

 复制代码

```
1  templateFolder := filepath.Join(currentPath, "template-hade-"+version+"-"+
2  os.Mkdir(templateFolder, os.ModePerm)
3  fmt.Println("创建临时目录", templateFolder)
4
5  // 拷贝template项目
6  url := release.GetZipballURL()
7  err := util.DownloadFile(filepath.Join(templateFolder, "template.zip"),
8  if err != nil {
9      return err
10 }
11 fmt.Println("下载zip包到template.zip")
12
13 _, err = util.Unzip(filepath.Join(templateFolder, "template.zip"), templ
14 if err != nil {
15     return err
16 }
17
18 // 获取folder下的gohade-hade-xxx相关解压目录
19 fInfos, err := ioutil.ReadDir(templateFolder)
20 if err != nil {
21     return err
```

```
22     }
23     for _, fInfo := range fInfos {
24         // 找到解压后的文件夹
25         if fInfo.IsDir() && strings.Contains(fInfo.Name(), "gohade-hade-") {
26             if err := os.Rename(filepath.Join(templateFolder, fInfo.Name()), f
27                 return err
28             }
29         }
30     }
31     fmt.Println("解压zip包")
32
33     if err := os.RemoveAll(templateFolder); err != nil {
34         return err
35     }
36     fmt.Println("删除临时文件夹", templateFolder)
```

第一步的源码复制完成之后, 就是后面很简单的四步了, 我直接把顺序写在注释中了, 你可以对照代码看, 同样在 framework/command/new.go 中:

[复制代码](#)

```
1  os.RemoveAll(path.Join(folder, ".git"))
2      fmt.Println("删除.git目录")
3
4      // 删除framework 目录
5      os.RemoveAll(path.Join(folder, "framework"))
6      fmt.Println("删除framework目录")
7
8      filepath.Walk(folder, func(path string, info os.FileInfo, err error) error {
9          if info.IsDir() {
10              return nil
11          }
12
13          c, err := ioutil.ReadFile(path)
14          if err != nil {
15              return err
16          }
17          // 修改go.mod中的模块名称、修改go.mod中的require信息
18          // 增加require github.com/gohade/hade
19          if path == filepath.Join(folder, "go.mod") {
20              fmt.Println("更新文件:" + path)
21              c = bytes.ReplaceAll(c, []byte("module github.com/gohade/hade"), [
22              c = bytes.ReplaceAll(c, []byte("require ("), []byte("require (\n\t
23              err = ioutil.WriteFile(path, c, 0644)
24              if err != nil {
25                  return err
26              }
27              return nil
28          }
```



```
29 // 最后修改所有文件使用业务目录的地方，
30 // 将原本使用“github.com/gohade/hade/app”的所有引用
31 // 改成 “[模块名称]/app”
32 isContain := bytes.Contains(c, []byte("github.com/gohade/hade/app"))
33 if isContain {
34     fmt.Println("更新文件:" + path)
35     c = bytes.ReplaceAll(c, []byte("github.com/gohade/hade/app"), []byte("[模块名称]/app"))
36     err = ioutil.WriteFile(path, c, 0644)
37     if err != nil {
38         return err
39     }
40 }
41
42 return nil
43 })
44 fmt.Println("创建应用结束")
45 fmt.Println("目录:", folder)
46 fmt.Println("=====")
47 return nil
```

## 验证

最后我们验证下。使用 `./hade new` 创建一个目录名称为 `testdemo`、模块名为 `github.com/jianfengye/testdemo`、版本为最新版本 `v0.0.2` 的脚手架。

```
~/Documents/UGit/coredemo geekbang/21 ➤ ./hade new
? 请输入目录名称: testdemo
? 请输入模块名称(go.mod中的module, 默认为文件夹名称): github.com/jianfengye/testdemo
? 请输入版本名称(参考 https://github.com/gohade/hade/releases, 默认为最新版本):

=====
开始进行创建应用操作
创建目录: /Users/yejianfeng/Documents/UGit/coredemo/testdemo
应用名称: github.com/jianfengye/testdemo
hade框架版本: v0.0.2
创建临时目录 /Users/yejianfeng/Documents/UGit/coredemo/template-hade-v0.0.2-1633620088
下载zip包到template.zip
解压zip包
删除临时文件夹 /Users/yejianfeng/Documents/UGit/coredemo/template-hade-v0.0.2-1633620088
删除.git目录
删除framework目录
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/app/console/kernel.go
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/app/http/module/demo/api.go
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/app/http/module/demo/mapper.go
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/app/http/route.go
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/go.mod
更新文件:/Users/yejianfeng/Documents/UGit/coredemo/testdemo/main.go
创建应用结束
目录: /Users/yejianfeng/Documents/UGit/coredemo/testdemo
=====
```

进入 testdemo 目录, 执行 `go build` 命令可直接编译, 并且生成了可运行的二进制文件。

```
~/Documents/UGit/coredemo/testdemo geekbang/21 ➤ ./testdemo
hade 框架提供的命令行工具, 使用这个命令行工具能很方便执行框架自带命令, 也能很方便编写业务命令

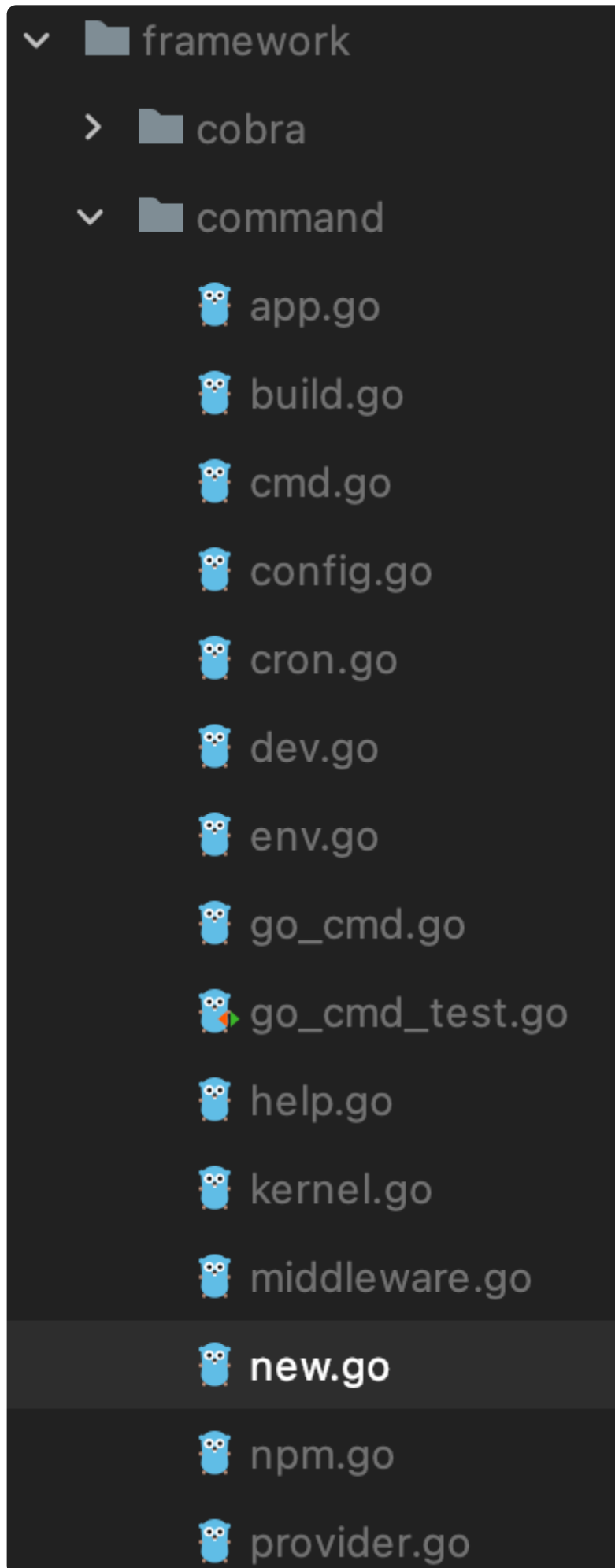
Usage:
  hade [flags]
  hade [command]

Available Commands:
  app      业务应用控制命令
  build    编译相关命令
  command  控制台命令相关
  config   获取配置相关信息
  cron     定时任务相关命令
  dev      调试模式
  env      获取当前的App环境
  foo      foo的简要说明
  go       运行path/go程序, 要求go 必须安装
  help     Help about any command
  middleware 中间件相关命令
  new      创建一个新的应用
  npm      运行 PATH/npm 的命令
  provider 服务提供相关命令

Flags:
  -h, --help  help for hade
```

自动化初始化脚手架命令完成!

今天所有代码都保存在 GitHub 上的 [@geekbang/22](#) 分支了。附上目录结构供你对比查看, 只修改了 framework/command/ 目录下的 new.go 代码。



## 小结

今天我们增加了一个新的命令，自动化初始化脚手架的命令设计，让 hade 框架也可以像 Vue 框架一样，直接使用一个二进制命令 `./hade new` 创建一个脚手架。我们把框架和脚手架示例代码同时放在 `github.com/gohade/hade` 仓库中，实现了框架和脚手架示例代码版本的关联。

在创建脚手架的时候，我们是**基于这个仓库的某个 tag 版本做减法**，而不是费劲地做加法来进行创建。

同时在每次更新框架的时候，我们也会自然而然更新这个示例代码，**框架和示例代码永远是一一对应的，而下载的时候会保留这种一一对应的关系**。这种设计让 hade 版本的框架设计更为方便了。

这两节课的四个工具的自动化，是我们目前能想到的比较常用的“重复性”劳动了。当然随着框架使用的深入，还可能有更多的自动化需求，但是基本上都和这几个自动化命令是同样的套路，所以掌握这两节课的内容和方法，你已经可以自行简化这些“重复性”劳动了。

## 思考题

这节课的代码比较多，希望你能仔细对比 GitHub 上的代码。经过这两节课的练习，你可以思考一下，作为一个“懒惰”的程序员，在 hade 框架中，我们还有哪些工作还可以自动化么？

欢迎在留言区分享你的思考。如果你觉得有收获，也欢迎把今天的内容分享给身边的朋友，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 21 | 自动化：DRY，如何自动化一切重复性劳动？（上）

11.11 全年底价

VIP 年卡限定 3 折

畅学 200 门课程 & 新课上线即解锁

超值拿下 ¥999

极客时间 VIP 年卡

365 天畅学

11.11 超值福

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。