

# go mod管理工具

发布日期	版本号	作者	描述
2020年10月04日	0.9.2	zwb	

## 目录

### go mod管理工具

- 目录
- 更新日志
- 关于勘误
- Go包管理
- 总体介绍
- 功能列表
- 安装步骤
- 使用文档
  - 1. 前提条件
  - 2. 子命令
  - 3. 项目打包
    - 帮助文档
    - 示例
  - 4. 项目安装
    - 帮助文档
    - 示例
  - 5. 项目发布
    - 帮助文档
    - 示例
- 参考引用

## 更新日志

更新日期	版本号	作者	描述
2020年10月04日	0.9.2	zwb	增加部署功能
2020年10月03日	0.9.1	zwb	优化打zip包逻辑
2020年10月02日	0.9.0	zwb	初始版本

## 关于勘误

由于时间水平都比较有限，所有文档中难免会出现些纰漏和错误，如果大家发现了一些问题，欢迎大家提交issue反馈

# Go包管理

---

在 Go1.5 之前用 `GOPATH` 以及 `GOROOT` 这两个环境变量来管理包的位置，`GOROOT` 为 Go 的安装目录，以及编译过程中使用到的系统库存放位置，如 `fmt`。Go1.5 到 Go1.7 开始稳定到 `vendor` 方式，即依赖包需要放到 `$GOPATH/src/vendor` 目录下，这样每个项目都有自己的 `vendor` 目录，但是如果依赖同样的三方包，很容易造成资源重复，Go `vendor` 出现了几种主流的管理工具，包括 `godep`、`govendor`、`golide` 等。

在 Go1.11 之前，`GOPATH` 是开发时的工作目录，其中包含三个子目录：

- `src`目录：存放go项目源码和依赖源码，包括使用 `go get` 下载的包
- `bin`目录：通过使用 `go install` 命令将 `go build` 编译出的二进制可执行文件存放于此
- `pkg`目录：go源码包编译生成的lib文件存储的地方

在 Go1.11 之前，`import` 包时的搜索路径

- `GOROOT/src`：该目录保存了Go标准库代码(首先搜寻导入包的地方)
- `GOPATH/src`：该目录保存了应用自身的各个包代码和第三方依赖的代码
- `./vendor`：`vendor` 方式第三方依赖包（如果支持Vendor）

在 Unix 和类 Unix 系统上，`GOPATH` 默认值是 `$HOME/go`，Go1.11 版本后，开启 Go Modules 后，`GOPATH` 的作用仅仅为 存放 依赖的目录了。

在 Go 的 1.11 版本之前，`GOPATH` 是必需的，且所有的 Go 项目代码都要保存在 `GOPATH/src` 目录下，也就是如果想引用本地的包，你需要将包放在 `$GOPATH/src` 目录下才能找得到。Go 的 1.11 版本之后，Go 官方引入了 Go Modules，不仅仅方便的使用我们的依赖，而且还对依赖的版本进行了管理。

在Go1.11后通过 `go mod vendor` 和 `-mod=vendor` 来实现 Vendor 管理依赖方式。本来在 `vgo` 项目 (Go Modules前身)是要完全放弃 `vendor`，但是在社区反馈下还是保留了。总之就是在 Go.1.11 之后需要开启 Go Modules 条件下才能使用 Vendor，具体地感兴趣或还沿用了 Vendor 的朋友可以去了解，不过建议以后仅使用 Go Modules 包管理方式了

## 总体介绍

---

虽然 Go1.11 引入了Go Modules的包管理机制，但是依然存在一些问题：

1. 项目需要托管在公共仓库上，比如GitHub，GitLab等，对于内部私有项目存在一些问题
2. 虽然一些办法解决私有项目访问权限的问题，但是配置过于繁琐，详见 [Go Modules私有项目配置](#)

Java、`nodejs` 一样通过 `maven`、`npm` 去进行管理，java、`nodejs`私有包都可以通过nexus发布进行管理，解决项目内部私有访问问题，我希望Go也有这样方式；找了很多资料，发现 [athens](#) 是一个比较理想的私有化包管理方案，详细介绍：[不一样的go语言-athens私仓安装](#)，虽然解决私有包下载问题，但是如何比较友好的发布至 `athens`，没有提供一个好的办法。

为了解决以上问题，开发了 `go-mod` 命令行工具，用于管理本地Go私有项目，用于go mod的打包、安装、发布，配合 [athens](#)使用就更加完美了。

# 功能列表

---

## ☑️ 打包

将本地Go Module项目打包，可手工将此包部署至 `athens`

## ☑️ 安装

将本地Go Module项目安装到本地Go mod缓存目录，方便

## ☑️ 发布

☑️ 将本地Go Module项目并发布至 `go-mod-proxy`

`go-mod-proxy` 的[安装部署](#)

☐ 将本地Go Module项目并发布至 `athens`，待实现

## ☑️ 清理

清理本地打包目录

# 安装步骤

---

## 方式一：安装最新版本

git clone源码，然后执行 `scripts` 中的 `build.sh` 脚本进行打包，生成文件在项目的 `releases` 目录下

## 方式二：直接Releases中直接下载对应版本

windows版本: [go-mod 0.9.2 windows amd64.exe](#)

linux版本: [go-mod 0.9.2 linux amd64](#)

mac版本: [go-mod 0.9.2 darwin amd64](#)

下载完成后，重名为简单名字比如 `go-mod.exe`，然后配置好PATH环境变量就可以直接使用了

# 使用文档

---

## 1. 前提条件

### 1) 配置GOBIN环境变量

默认目录在`$user_home/go`目录下

### 2) 配置GONOSUMDB

`GONOSUMDB` Go环境变量用于设置哪些前缀的模块都被视为私有模块，不进行HASH检验，通过逗号分隔配置多个匹配路径，因为我们的打包项目未生成hash校验值，所以需要配置，**后续实现校验文件，就无需进行配置**

```
go env -w GONOSUMDB=github.com/wenit
```

也可通过关闭整个HASH校验，不建议这么做

GOSUMDB="off"  
可以被使用

# 关闭校验，任何模块

## 2. 子命令

```
go-mod.exe help
```

使用说明：

Usage:

```
go-mod.exe  
go-mod.exe [command]
```

Available Commands:

clean	清理目录
help	更多帮助文档
install	安装go module到本地module库
package	本地项目打包

Flags:

-h, --help	帮助信息
-v, --version	版本信息

Use "go-mod.exe [command] --help" for more information about a command.

## 3. 项目打包

### 帮助文档

```
go-mod.exe package --help
```

Package your Go modules

Usage:

```
go-mod.exe package [flags]
```

Flags:

-e, --excludes string	排除目录，多个目录使用逗号分割 (default ".svn,.git,.vscode,target,releases")
-f, --from string	输入目录 (default ".")
-h, --help	help for package
-t, --to string	输出目录 (default "./target")
-v, --version string	输出版本号 (default "v1.0.0")

### 示例

输入：

```
go-mod.exe package -f F:/github/go-mod
```

输出：

2020/10/02 11:00:21 项目打包完成，输出目录: ./target

## 4. 项目安装

### 帮助文档

```
go-mod.exe install --help
```

安装go module到本地module库

Usage:

```
go-mod.exe install [flags]
```

Flags:

```
-e, --excludes string  排除目录，多个目录使用逗号分割 (default ".svn,.git,.vscode,target,releases")
-f, --from string       输入目录 (default ".")
-h, --help              help for install
-t, --to string         输出目录 (default "./target")
-v, --version string    输出版本号 (default "v1.0.0")
```

### 示例

输入:

```
go-mod.exe install -f F:/github/go-mod -v v1.1.0
```

输出:

```
go-mod.exe install -f F:/github/go-mod -v v1.1.0
2020/10/02 11:00:44 项目打包完成，输出目录: ./target
2020/10/02 11:00:44 项目解压至本地mod仓库，输出目录: C:\Users\wenit\go\pkg\mod
2020/10/02 11:00:44 复制info文件至缓存目录
[C:\Users\wenit\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.0.zip]
完成
2020/10/02 11:00:44 复制mod文件至缓存目录
[C:\Users\wenit\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.0.zip]
完成
2020/10/02 11:00:44 复制zip文件至缓存目录
[C:\Users\wenit\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.0.zip]
完成
```

## 5. 项目发布

### 帮助文档

```
go-mod.exe deploy --help
```

安装go module到本地module库，并部署至go-mod-proxy

Usage:

```
go-mod.exe deploy [flags]
```

Flags:

```
-e, --excludes string    排除目录，多个目录使用逗号分割 (default  
".svn,.git,.vscode,target,releases")  
-f, --from string        输入目录 (default ".")  
-h, --help               help for deploy  
-p, --proxyaddr string    go-mod-server的api服务地址，端口一般为go-mod-server的代理端  
口+1  
-t, --to string           输出目录 (default "./target")  
-v, --version string      输出版本号 (default "v1.0.0")
```

## 示例

输入:

```
go-mod.exe deploy -f f:/github/go-mod -v v1.1.2 -p http://localhost:8082/upload
```

输出:

```
2020/10/04 18:37:01 部署私服地址: http://localhost:8082/upload  
2020/10/04 18:37:01 项目打包完成, 输出目录: ./target  
2020/10/04 18:37:01 项目解压至本地mod仓库, 输出目录: C:\Users\zwb\go\pkg\mod  
2020/10/04 18:37:01 复制info文件至缓存目录  
[C:\Users\zwb\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.2.zip] 完  
成  
2020/10/04 18:37:01 复制mod文件至缓存目录  
[C:\Users\zwb\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.2.zip] 完  
成  
2020/10/04 18:37:01 复制zip文件至缓存目录  
[C:\Users\zwb\go\pkg\mod\cache\download\github.com\wenit\go-mod\@v\v1.1.2.zip]  
完成  
2020/10/04 18:37:01 复制ziphash文件至缓存目录  
[C:\Users\zwb\go\pkg\mod\cache\download\github.com\wenit\go-  
mod\@v\v1.1.2.ziphash] 完成  
2020/10/04 18:37:01 上传成功[github.com/wenit/go-mod-v1.1.2]
```

## 参考引用

本项目参考: [pacmod](#) 实现

