Go Modules、Go Module Proxy 和 goproxy.cn

盛傲飞 · 应届生 · goproxy.cn 的作者

第 61 期
2019-09-26

内容大纲

- Go Modules 简介
- 快速迁移项目至 Go Modules
- 使用 Go Modules 时常遇见的坑
  - 坑 1：判断项目是否启用了 Go Modules
  - 坑 2：管理 Go 的环境变量
  - 坑 3：从 dep、glide 等迁移至 Go Modules
  - 坑 4：拉取私有模块
  - 坑 5：更新现有的模块
  - 坑 6：主版本号
- Go Module Proxy 简介
- 快速搭建私有 Go Module Proxy
- Goproxy 中国（goproxy.cn）

注：本次分享的所有内容都将以 Unix-like 系统终端下的 Go 1.13 为标准来讲解！

Go Modules 简介

Go Modules 简介

- Go modules（前身 vgo）是 Go team（Russ Cox）**强推**的一个**理想化**的**类语言级**依赖管理解决方案
- 发布于 Go 1.11，成长于 Go 1.12，丰富于 Go 1.13，目测完善于 Go++
- 为淘汰 GOPATH 而生
- Opt-in 式设计
- 官方 Wiki 介绍：github.com/golang/go/wiki/Modules
- go help modules：

  A module is a collection of related Go packages.
  Modules are the unit of source code interchange and versioning.
  The go command has direct support for working with modules,
  including recording and resolving dependencies on other modules.
  Modules replace the old GOPATH-based approach to specifying
  which source files are used in a given build.
  ...

youtu.be/F8nrpe0XWRg

Go Modules 简介

- 1 个开关环境变量：GO111MODULE
- 5 个辅助环境变量：GOPROXY、GONOPROXY、GOSUMDB、GONOSUMDB 和 GOPRIVATE
- 2 个辅助概念：Go module proxy 和 Go checksum database
- 2 个主要文件：go.mod 和 go.sum
- 1 个主要管理子命令：go mod
- 内置在几乎所有其他子命令中：go get、go install、go list、go test、go run、go build……
- Global Caching：不同项目的相同模块版本只会在你的电脑上缓存一份儿（让 npm 嫉妒死 😉）

go.mod

go.mod 是启用了 Go moduels 的项目所必须的最重要的文件，它描述了当前项目（aka 当前模块）的元信息，每一行都以一个动词开头，目前有以下 5 个动词：

- **module**：用于定义当前项目的模块路径
- **go**：用于设置预期的 Go 版本
- **require**：用于需求一个特定的模块版本
- **exclude**：用于从使用中排除一个特定的模块版本
- **replace**：用于将一个模块版本替换为另外一个模块版本

```
module example.com/foobar

go 1.13

require (
        example.com/apple v0.1.2
        example.com/banana v1.2.3
        example.com/banana/v2 v2.3.4
        example.com/pineapple v0.0.0-20190924185754-1b0db40df49a
)

exclude example.com/banana v1.2.4

replace example.com/apple v0.1.2 => example.com/rda v0.1.0
replace example.com/banana => example.com/hugebanana
```

go.sum

go.sum 是类似于比如 dep 的 Gopkg.lock 的一类文件，它详细罗列了当前项目直接或间接依赖的所有模块版本，并写明了那些模块版本的 SHA-256 哈希值以备 Go 在今后的操作中保证项目所依赖的那些模块版本不会被篡改。

```
example.com/apple v0.1.2 h1:WXkYYl6Yr3qBf1K79EBnL4mak0OimBfB0XUf9Vl28OQ=
example.com/apple v0.1.2/go.mod h1:xHWCNGjB5oqiDr8zfno3MHue2Ht5sIBksp03qcyfWMU=
example.com/banana v1.2.3 h1:qHgHjyoNFV7jgucU8QZUuU4gcdhfs8QW1kw68OD2Lag=
example.com/banana v1.2.3/go.mod h1:HSdplMjZKSmBqAxg5vPj2TmRDmfkzw+cTzAElWljhcU=
example.com/banana/v2 v2.3.4 h1:zl/OfRA6nftbBK9qTohYBJ5xvw6C/oNKizR7cZGl3cI=
example.com/banana/v2 v2.3.4/go.mod h1:eZbhyaAYD41SGSSsnmcpxVoRiQ/MPUTjUdIIOT9Um7Q=
...
```

Go Modules 简介

GO111MODULE

- 它共有三个值
  - auto：只在项目包含了 go.mod 文件时启用 Go modules，在 Go 1.13 中仍然是默认值，详见：golang.org/issue/31857
  - on：无脑启用 Go modules，推荐设置，未来版本中的默认值，让 GOPATH 从此成为历史
  - off：禁用 Go modules 🙄

GOPROXY

- 它的值是一个以英文逗号“,”分割的 Go module proxy 列表（稍后讲解），用于使 Go 在后续拉取模块版本时能够脱离传统的 VCS 方式从镜像站点快速拉取，它的值也可以是“off”即禁止 Go 从任何地方拉取模块版本
- 拥有默认值：https://proxy.golang.org,direct
- proxy.golang.org 在中国无法访问，故而建议使用 goproxy.cn 作为替代
  - go env -w GOPROXY=https://goproxy.cn,direct
- 值列表中的“direct”为特殊指示符，用于指示 Go 回源到模块版本的源地址去抓取（比如 GitHub 等）
- 当值列表中上一个 Go module proxy 返回 404 或 410 错误时，Go 会自动尝试列表中的下一个，遇见“direct”时终止并回源，遇见 EOF 时终止并抛出类似“invalid version: unknown revision...”的错误

GOSUMDB

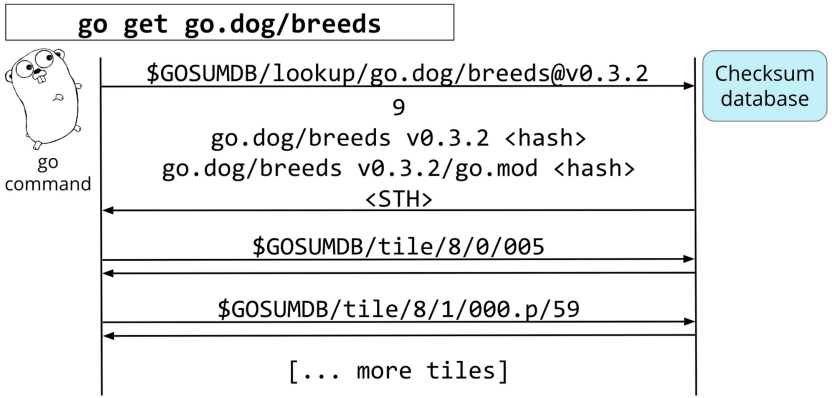
- 它的值是一个 Go checksum database，用于使 Go 在拉取模块版本时（无论是从源站拉取还是通过 Go module proxy 拉取）保证拉取到的模块版本数据未经篡改，它的值也可以是“off”即禁止 Go 校验任何模块版本
  - 格式 1：<SUMDB_NAME>+<PUBLIC_KEY>
  - 格式 2：<SUMDB_NAME>+<PUBLIC_KEY> <SUMDB_URL>
- 拥有默认值：sum.golang.org（之所以没有按照上面的格式是因为 Go 对默认值做了特殊处理）
- 可被 Go module proxy 代理（详见：Proxying a Checksum Database）
- sum.golang.org 在中国无法访问，故而更加建议将 GOPROXY 设置为 goproxy.cn，因为 goproxy.cn 支持代理 sum.golang.org

Go Checksum Database

- 前身 notary，透明公证人系统
- 保护 Go 模块的生态系统
- 防止 Go 从任何源头（包括 Go module proxy）意外拉取了经过篡改模块版本，引入了意外的代码更改
- Proposal: Secure the Public Go Module Ecosystem
- go help module-auth：

  The go command tries to authenticate every downloaded module, checking that the bits downloaded for a specific module version today match bits downloaded yesterday. This ensures repeatable builds and detects introduction of unexpected changes, malicious or not.

  ...



GONOPROXY、GONOSUMDB 和 GOPRIVATE

- 这三个环境变量都是用在当前项目依赖了私有模块，也就是依赖了由 GOPROXY 指定的 Go module proxy 或由 GOSUMDB 指定 Go checksum database 无法访问到的模块时的场景
- 它们三个的值都是一个以英文逗号“,”分割的模块路径前缀，匹配规则同 path.Match
- 其中 GOPRIVATE 较为特殊，它的值将作为 GONOPROXY 和 GONOSUMDB 的默认值，所以建议的最佳姿势是只使用 GOPRIVATE
  - 比如 GOPRIVATE=*.corp.example.com 表示所有模块路径以 corp.example.com 的下一级域名（如 team1.corp.example.com）为前缀的模块版本都将不经过 Go module proxy 和 Go checksum database，需要注意的是不包括 corp.example.com 本身
- go hlep module-private：

The go command defaults to downloading modules from the public Go module mirror at proxy.golang.org. It also defaults to validating downloaded modules, regardless of source, against the public Go checksum database at sum.golang.org. These defaults work well for publicly available source code.
...

Global Caching

- 同一个模块版本的数据只缓存一份，所有其他模块共享使用
- 目前所有模块版本数据均缓存在 $GOPATH/pkg/mod 和 $GOPATH/pkg/sum 下，未来或将移至 $GOCACHE/mod 和 $GOCACHE/sum 下（当 GOPATH 被淘汰后）
- 可以使用 go clean -modcache 清理所有已缓存的模块版本数据

快速迁移项目至 Go Modules

快速迁移项目至 Go Modules

- 第一步：升级到 Go 1.13（我们并不建议在 Go 1.11.x 或 Go 1.12.x 时迁移至 Go modules，因为问题太多）
- 第二步：让 GOPATH 从你的脑海中完全消失，早一步踏入未来
  - go env -w GOBIN=$HOME/bin
  - go env -w GO111MODULE=on
  - go env -w GOPROXY=https://goproxy.cn,direct # 在中国是必须的，因为它的默认值被墙了
- 第三步（可选）：按照你喜欢的目录结构重新组织你的所有项目（再也不用在 $GOPATH/src 里挣扎啦！🎉）
- 第四步：在你项目的根目录下执行 go mod init <OPTIONAL_MODULE_PATH> 以生成 go.mod 文件
- **第五步：想办法说服你身边所有的人都去走一下前四步** 😬

快速迁移项目至 Go Modules

- 用 go help module-get 和 go help gopath-get 分别去了解 Go modules 启用和未启用两种状态下的 go get 的行为
- 用 go get 拉取新的依赖
  - go get golang.org/x/text@latest # 拉取最新的版本（优先择取 tag）
  - go get golang.org/x/text@master # 拉取 master 分支的最新 commit
  - go get golang.org/x/text@v0.3.2 # 拉取 tag 为 v0.3.2 的 commit
  - go get golang.org/x/text@342b2e1 # 拉取 hash 为 342b231 的 commit，最终会被转换为 v0.3.2
- 用 go get -u 更新现有的依赖
- 用 go mod download 下载 go.mod 文件中指明的所有依赖
- 用 go mod tidy 整理现有的依赖
- 用 go mod graph 查看现有的依赖结构
- 用 go mod init 生成 go.mod 文件（Go 1.13 中唯一一个可以生成 go.mod 文件的子命令）
- 用 go mod edit 编辑 go.mod 文件
- 用 go mod vendor 导出现有的所有依赖（事实上 Go modules 正在淡化 Vendor 的概念）
- 用 go mod verify 校验一个模块是否被篡改过

使用 Go Modules 时常遇见的坑

使用 Go Modules 时常遇见的坑（坑 1：判断项目是否启用了 Go Modules）

- 切记，在 Go 1.13 中，一个项目只要包含了 go.mod 文件，且它所处环境中的 GO111MODULE 不为 off，那么 Go 就会为这个项目启用 Go modules
- 为了一劳永逸地解决这个判断烦恼，建议大家将 GO111MODULE 设置为 on
  - go env -w GO111MODULE=on
- 另外，若当前项目没有包含 go.mod 文件，且 GO111MODULE 为 on，那么每一次构建代码时 Go 都会从头推算并拉取所需要的模块版本，**但是并不会自动生成 go.mod 文件**（以前会自动生成，Go 1.13 做了调整）

使用 Go Modules 时常遇见的坑（坑 2：管理 Go 的环境变量）

- 在 Go 1.13 中管理环境变量会变得稍显混乱，因为 Go 1.13 建议将所有跟 Go 相关的环境变量都交由新出的 go env -w 来管理，比如执行 go env -w GO111MODULE=on 就会在 $HOME/.config/go/env 文件中追加一行 “GO111MODULE=on”
- **但 go env -w 不会覆盖你的系统环境变量**
- 所以建议大家在升级到 Go 1.13 后做一次环境变量的管理方式的转变，比如删除你系统中所有跟 Go 相关的环境变量，并使用 go env -w 重写会
- os.UserConfigDir：

> UserConfigDir returns the default root directory to use for user-specific configuration data. Users should create their own application-specific subdirectory within this one and use that.
>
> On Unix systems, it returns $XDG_CONFIG_HOME as specified by https://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html if non-empty, else $HOME/.config. On Darwin, it returns $HOME/Library/Application Support. On Windows, it returns %AppData%. On Plan 9, it returns $home/lib.
>
> If the location cannot be determined (for example, $HOME is not defined), then it will return an error.

使用 Go Modules 时常遇见的坑（坑 3：从 dep、glide 等迁移至 Go Modules）

- 在执行 go mod init <OPTIONAL_MODULE_PATH> 时，如果当前目录包含了 Gopkg.lock、glide.lock 等老的依赖管理文件时，Go 会用它们所指定的依赖版本信息来推算并生成 go.mod 文件，此时所有的推算行为都是直接回源的，并不会经过 Go module proxy，这是个 Bug，详见：golang.org/issue/33767
- 不出意外的话这个 Bug 将在 Go 1.14 中被修复，我们正在努力
- 临时的解决办法时先按照 go help go.mod 里指定的规则手动创建一个 go.mod 文件（可以只包含第一行的模块路径），然后再执行 go mod tidy 来推算并补充 go.mod 文件，这样儿就可以走 Go module proxy 了

使用 Go Modules 时常遇见的坑（坑 4：拉取私有模块）

- 常见的公共 Go module proxy，比如 proxy.golang.org 和 goproxy.cn 都是无权访问任何人的私有模块的
- 即使不使用任何 Go module proxy，也就是将 GOPROXY 设置为了 direct，同样默认情况下 Go 也是无法抓取你的私有模块的，详见：golang.org/doc/faq#git_https
- 解决办法是想办法修改你的 VCS 拉取行为，比如使用 Git 时就在 $HOME/.gitconfig 文件中追加：

```
[url "ssh://git@github.com/"]
        insteadOf = https://github.com/
```

- 并为 GOPROXY 设置一个 fallback 选项（当列表中的前一个 404 或 410 时就自动尝试下一个，遇见 direct 时回源也就是直接去模块所在地拉取）
  - go env -w GOPROXY=https://goproxy.cn,direct
- 还可以通过设置 GONOPROXY 或 GOPRIVATE 来指使 Go 在拉取哪些模块时忽略 Go module proxy

使用 Go Modules 时常遇见的坑（坑 5：更新现有的模块）

- go get -u
  - 只会更新主要模块，忽略了单元测试
- go get -u ./...
  - 递归更新所有子目录的所有模块，忽略了单元测试
- go get -u -t
  - 同样是只会更新主要模块，但考虑了单元测试
- go get -u -t ./...
  - 同样是递归更新所有子目录的所有模块，但考虑了单元测试
- go get -u all
  - 更新所有模块，推荐使用

使用 Go Modules 时常遇见的坑（坑 6：主版本号）

- 除了 v0 和 v1 外主版本号必须显式地出现在模块路径的尾部
- Go 会将 example.com/foo/bar 和 example.com/foo/bar/v2 视为完全不同的两个模块来对待
- 一个项目中可能同时存在多个拥有不同的主版本号的相同模块，但互不影响
- go get -u 不会更新主版本号，如需更新，需要手动修改代码中对应的导入路径

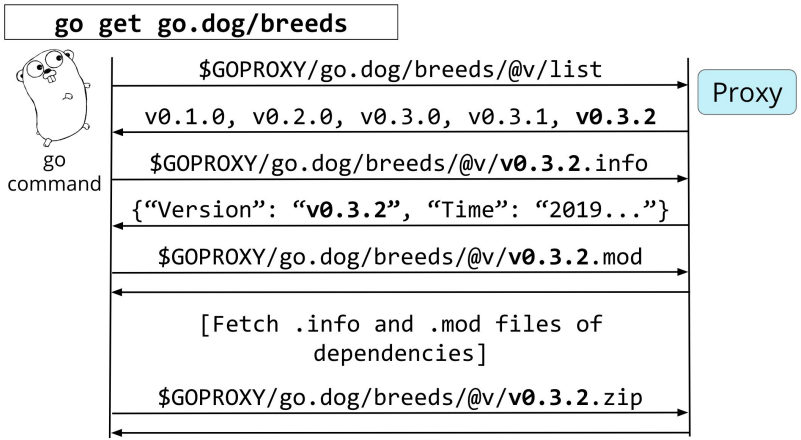
Go Module Proxy 简介

Go Module Proxy 简介

- 集中式模块版本管理
- 加快模块版本的拉取速度
- 加快项目的构建速度
- 使 Go 脱离对 VCS 的依赖
- 防止项目由于依赖某个模块被其作者删除而被 break 掉
- 淡化 Vendor 概念
- go help goproxy:

  A Go module proxy is any web server that can respond to GET requests for URLs of a specified form. The requests have no query parameters, so even a site serving from a fixed file system (including a file:/// URL) can be a module proxy.

  ...



Go Module Proxy 简介

- 1 个主要环境变量：GOPROXY（默认值为国内无法访问的“https://proxy.golang.org,direct”）
- 1 个辅助环境变量：GONOPROXY
- **从 Go 1.13 起，以后在国内必须修改 GOPROXY 才能正常开发 Go 程序**
  - **go env -w GOPROXY=https://goproxy.cn,direct**
- 可以指定 GONOPROXY 来指使 Go 在拉取哪些模块时忽略 Go module proxy 并无脑回源
  - 如 go env -w GONOPROXY=*.corp.example.com,git.example.com/foo/bar 就意味着 Go 在拉取所有模块路径的前缀和列表相匹配的模块版本时都将无脑回源，匹配规则同 path.Match

快速搭建私有 Go Module Proxy

快速搭建私有 Go Module Proxy

两个主流方法

- 使用现成软件 Athens：github.com/gomods/athens（不讲解）
- 使用 Goproxy 以编码的形式实现：github.com/goproxy/goproxy

Goproxy - github.com/goproxy/goproxy

- 一个极简主义的 Go module proxy 处理器
  - 完全实现了 Go module proxy protocol
  - 实现了 http.Handler 接口，可结合 Gin 之类的框架使用
- 极易使用
  - 一个 struct：goproxy.Goproxy
  - 两个 interface：goproxy.Cacher 和 goproxy.Cache
- 可通过设置 GOPROXY 环境变量来指定上游代理
- 内置实现了对 GONOPROXY、GOSUMDB、GONOSUMDB、GOPRIVATE 的支持
- 支持代理 Go checksum database
- 丰富的 goproxy.Cacher 实现
  - 磁盘：cacher.Disk
  - MinIO：cacher.MinIO
  - 谷歌云存储：cacher.GCS
  - 亚马逊简易存储服务：cacher.S3
  - 微软 Azure Blob 存储：cacher.MABS
  - DigitalOcean Spaces：cacher.DOS
  - 阿里云对象存储服务：cacher.OSS
  - 七牛云对象存储服务：cacher.Kodo

快速搭建私有 Go Module Proxy

Goproxy - github.com/goproxy/goproxy

```go
package main

import (
	"net/http"

	"github.com/goproxy/goproxy"
)

func main() {
	http.ListenAndServe("localhost:8080", goproxy.New())
}
```

Goproxy - github.com/goproxy/goproxy

```
package main

import (
        "net/http"

        "github.com/goproxy/goproxy"
)

func main() {
        http.ListenAndServe("localhost:8080", goproxy.New())
}
```

LIVE DEMO

Goproxy 中国（goproxy.cn）

中国 Go 语言社区迫切地需要有一个咱们自己家的 Go Module Proxy

在 Go 1.13 中 GOPROXY 和 GOSUMDB 这两个环境变量都有了在中国无法 访问的默认值，尽管我在 golang.org/issue/31755 里努力尝试过，但最终仍然无法为咱们中国的 Go 语言开发者谋得一个完美的解决方案。所以从今以后咱 们中国的所有 Go 语言开发者，只要是使用了 Go modules 的，那么都必须先修改 GOPROXY 和 GOSUMDB 才能正常使用 Go 做开发，否则可能连一个最简单的程序都跑不起来（只要它有依 赖第三方模块）。

我创建 Goproxy 中国（goproxy.cn）的主要原因

其实更早的时候，也就是今年年初我也曾 试图在 golang.org/issue/31020 中请求 Go team 能想办法避免那时的 GOPROXY 即将拥有的默认值可以在中国正常 访问，但 Go team 似乎也无能 为力，为此我才坚定了创建 goproxy.cn 的信念。既然 别人没法儿帮忙，那咱 们就得自己动手，不为别的，就为了让大家以后能 够更愉快地使用 Go 语言配合 Go modules 做开发。

最初我先是和七牛云的 许叔（七牛云的 创始人兼 CEO 许式伟）提出了我打算 创建 goproxy.cn 的想法，本是抱着 试试看的目的，但没想 到许叔几乎是没有超 过一分钟的考虑便认可了我的想法并表示愿意一起推 动。那一阵子刚好赶上我在写 毕业论文，所以项目开发完后就一直没和七牛云做交接，一直跑在我的个人服 务器上。直到有一次 goproxy.cn 被攻击了，一下午的功夫 烧了我一百多美元，然后我才意识到这种项目真不能个人来做。个人来做不靠 谱，万一依赖这个项目的人多了，项目再出什么事儿，那就会 给大家造成不必要的 损失。所以我赶 紧和七牛云做了交接，把 goproxy.cn 完全交给了七牛云，甚至 连域名都过户了去。

Goproxy 中国（goproxy.cn）

- Goproxy 中国（goproxy.cn）是目前中国最可靠的 Go module proxy（真不是在自卖自夸 🥺）
- 为中国 Go 语言开发者量身打造，支持代理 GOSUMDB 的默认值，经过全球 CDN 加速，高可用，可应用进公司复杂的开发环境中，亦可用作上游代理
- 由中国倍受信赖的云服务提供商七牛云无偿提供基础设施支持的开源的非营利性项目
- 目标是为中国乃至全世界的 Go 语言开发者提供一个免费的、可靠的、持续在线的且经过 CDN 加速的 Go module proxy
- 域名已由七牛云进行了备案（沪ICP备11037377号-56）

区域	最快/最慢	平均
全网	北京(鹏博士) 0.2ms/柬埔寨金边(cambo.host) 360.8ms	25.6ms
移动	安徽合肥(hccn2) 3.3ms/广东广州(移动) 35.8ms	15.1ms
联通	天津(联通) 6.7ms/重庆(联通) 27.7ms	17.2ms
电信	四川成都(天翼云) 1.5ms/新疆乌鲁木齐(天翼云) 66.3ms	27.4ms
华南	广东广州(腾讯云) 9.0ms / 海南海口(天翼云) 49.9ms	25.3ms
华北	北京(鹏博士) 0.2ms / 内蒙古呼和浩特(蓝视云) 45.4ms	24.0ms
华东	上海(阿里云) 1.0ms / 山东青岛(天翼云) 48.2ms	20.1ms
华中	湖南长沙(天翼云) 5.4ms / 河南郑州(景安网络) 58.9ms	20.3ms
东北	辽宁鞍山(电信) 19.6ms / 吉林长春(天翼云) 57.7ms	33.3ms
西北	陕西西安(天翼云一区) 13.8ms / 新疆乌鲁木齐(天翼云) 66.3ms	41.7ms
西南	四川成都(天翼云) 1.5ms / 贵州贵阳(华为云) 58.5ms	26.6ms
港澳台	台湾台北(远传电信) 0.6ms / 香港(hkc) 155.6ms	27.9ms
亚洲	新加坡(腾讯云) 0.4ms/柬埔寨金边(cambo.host) 360.8ms	15.3ms
欧洲	意大利米兰 0.4ms/法国巴黎(VULTR) 146.5ms	22.1ms
非洲	南非约翰内斯堡 0.2ms/尼日利亚拉各斯 265.2ms	48.9ms
北美洲	美国西雅图(VULTR) 0.3ms/美国波特兰(AMAZON-LIGHTSAIL) 97.2ms	34.9ms
南美洲	巴西圣保罗(GCP) 3.7ms/阿根廷布宜诺斯艾利斯(latincloud.com) 145.0ms	50.9ms
大洋洲	澳大利亚悉尼(xtom.com) 0.7ms/澳大利亚阿德莱德 52.6ms	15.1ms

Goproxy 中国（goproxy.cn）截止到 2019-09-25 的状态

- 存储空间总大小已超过 **568.88 GB**
- 已缓存 **278,737** 个模块版本
- **0-10 MB** 的模块版本的 ZIP 文件共有 **262,615** 个
- **10-50 MB** 的模块版本的 ZIP 文件共有 **15,580** 个
- **50-100 MB** 的模块版本的 ZIP 文件共有 **451** 个
- **大于 100 MB** 的模块版本的 ZIP 文件共有 **91** 个（哥几个……你们是往代码里面加铁块儿了吗？🤨）

Goproxy 中国（goproxy.cn）在 21 天内的数据

2019-09-04 至 2019-09-25（15 个工作日）

- 处理了 **12,369,839** 条请求
- 出站了 **444.18 GB** 的流量
- 峰值带宽 **157.37 Mbit/s**
- 平均峰值带宽 **95.20 Mbit/s**

Goproxy 中国（goproxy.cn）

你只需知道一下 goproxy.cn，然后选择性用一下它，用顺手了考虑留下它，从今以后哪怕是你忘记它，再也不去访问它的首页，都没关系，它只是在你看不 见的地方默默地为你提供服务。

Q&A

谢谢观看！

分享者： 盛傲飞

个人网站： aofeisheng.com

电子邮件： aofei@aofeisheng.com

GitHub： github.com/aofei