
深入Go Module
鸟窝
细节、使用、坑以及未来

鸟窝

- Go微服务框架rpcx作者
- Go并发编程实战课专栏作者
- 《Scala集合技术手册》作者

GopherChina 2021

目 录

第一部分

Go版本管理历史

Go版本历史


GOPATH /2012/go 1.0


goven /2012/ .../kr/goven


godep /2013/ .../tools/godep


gopkg.in /2014


glide /2014/ .../Masterminds/glide


gb /2015/ .../constabulary/gb


govendor /2015/ .../kardianos/govendor


vendor dir /2015/ go1.5,go1.6


dep /2017/ .../golang/dep


go mod /2018/ go1.11,go1.12~go1.17

Go module历代变更

Go1.12	Go1.13	Go 1.14	Go 1.15	Go 1.16	Go 1.17
• GO111MODULE • 正式推出 • 影响go命令	• GO111MODULE未如预期去掉 • GOPROXY • 影响go命令	• go命令全面支持module，可以产品使用 • 支持-mod=vendor • GOINSECURE • 许多go命令的不兼容	• GOMODCACHE • GOPROXY支持跳过失败的proxy	• GO111MODULE默认为on • 又是一堆go命令不兼容 • retract • -e • 禁止特殊import path	• Lazy module loading • // Deprecated: comment • 不包含go directive，默认为go 1.11 • 又是一堆go命令不兼容

第二部分

认识 Go Module

概念

什么是 module?

module的定义

minor

module vs package

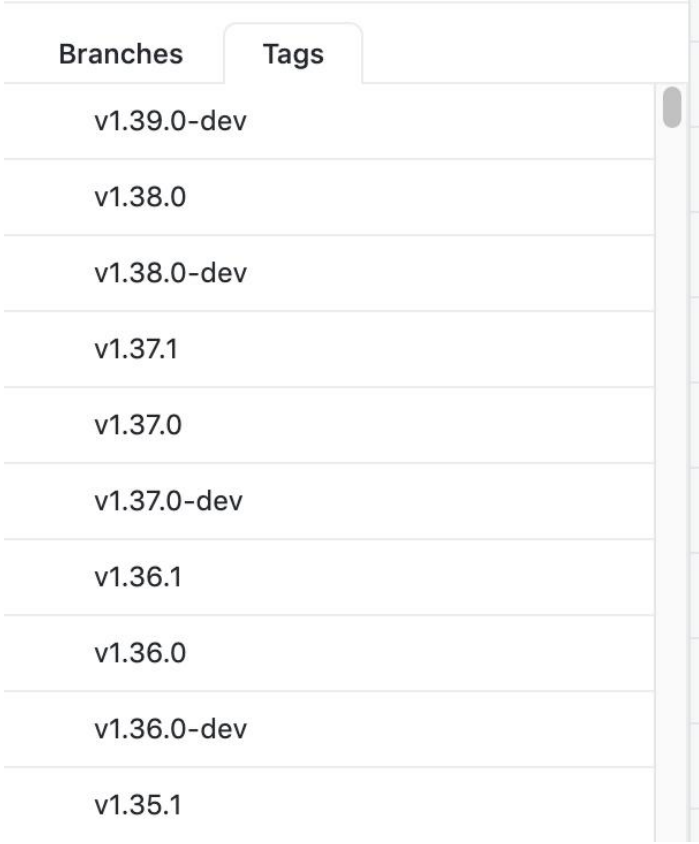
语义化版本 2.0.0

major.minor.patch-beta+metadata

版本格式：主版本号.次版本号.修订号，版本号递增规则如下：

1. 主版本号：当你做了不兼容的 API 修改，
2. 次版本号：当你做了向下兼容的功能性新增，
3. 修订号：当你做了向下兼容的问题修正。

先行版本号及版本编译信息可以加到“主版本号.次版本号.修订号”的后面，作为延伸。



go.mod 文件

```
module github.com/panicthis/modfile

go 1.16

require (
    github.com/coreos/bbolt v1.3.3
    github.com/stretchr/testify v1.7.0
    github.com/edwingeng/doublejump v0.0.0-20200330080233-e4ea8bd1cbed
    github.com/cenk/backoff v2.2.1+incompatible
    github.com/stretchr/objx v0.3.0 // indirect
    go.etcd.io/bbolt v1.3.6 // indirect
    golang.org/x/net v0.0.0-20210610132358-84b48f89b13b // indirect
    golang.org/x/sys v0.0.0-20210611083646-a4fc73990273 // indirect
)

exclude (
    go.etcd.io/etcd/client/v2 v2.305.0-rc.1
    go.etcd.io/etcd/client/v3 v3.5.0-rc.1
)

replace github.com/coreos/bbolt => go.etcd.io/bbolt v1.3.3
```

module path

go directive

version, Semantic Versioning 2.0.0
vmajor.minor.patch-prerelease+meta

go.mod 文件

```
module github.com/panicthis/modfile

go 1.16

require (
    github.com/coreos/bbolt v1.3.3
    github.com/stretchr/testify v1.7.0
    github.com/edwingeng/doublejump v0.0.0-20200330080233-e4ea8bd1cbed
    github.com/cenk/backoff v2.2.1+incompatible
    github.com/stretchr/objx v0.3.0 // indirect
    go.etcd.io/bbolt v1.3.6 // indirect
    golang.org/x/net v0.0.0-20210610132358-84b48f89b13b // indirect
    golang.org/x/sys v0.0.0-20210611083646-a4fc73990273 // indirect
)

exclude (
    go.etcd.io/etcd/client/v2 v2.305.0-rc.1
    go.etcd.io/etcd/client/v3 v3.5.0-rc.1
)

replace github.com/coreos/bbolt => go.etcd.io/bbolt v1.3.3
```

pseudo-version

vx.y.z-yyyymmddhhmmss-abcdefgh

incompatible

版本>=2,但是没有go.mod

indirect

依赖引入的依赖，go.mod不包含

require中的xxx

go.mod 文件

```
module github.com/panicthis/modfile

go 1.16

require (
    github.com/cenk/backoff v2.2.1+incompatible
    github.com/coreos/bbolt v1.3.3
    github.com/edwingeng/doublejump v0.0.0-20200330080233-e4ea8bd1cbed
    github.com/stretchr/objx v0.3.0 // indirect
    github.com/stretchr/testify v1.7.0
    go.etcd.io/bbolt v1.3.6 // indirect
    go.etcd.io/etcd/client/v2 v2.305.0-rc.1
    go.etcd.io/etcd/client/v3 v3.5.0-rc.1
    golang.org/x/net v0.0.0-20210610132358-84b48f89b13b // indirect
    golang.org/x/sys v0.0.0-20210611083646-a4fc73990273 // indirect
)

exclude (
    go.etcd.io/etcd/client/v2 v2.305.0-rc.0
    go.etcd.io/etcd/client/v3 v3.5.0-rc.0
)

replace github.com/coreos/bbolt => go.etcd.io/bbolt v1.3.3
```

exclude
排除特定的版本

replace
替换特定的版本、或者所有版本

go.mod 文件

```
exclude (
    go.etcd.io/etcd/client/v2 v2.305.0-rc.0
    go.etcd.io/etcd/client/v3 v3.5.0-rc.0
)

replace (
    github.com/coreos/bbolt => go.etcd.io/bbolt v1.3.3

    golang.org/x/net v1.2.3 => example.com/fork/net v1.4.5
    golang.org/x/net => example.com/fork/net v1.4.5
    golang.org/x/net v1.2.3 => ./fork/net
    golang.org/x/net => ./fork/net
)
```

替换bbolt所有版本

替换x/net v1.2.3版本

使用本地文件替换

replace中更多配置

go.mod 文件

```
exclude (
    go.etcd.io/etcd/client/v2 v2.305.0-rc.0
    go.etcd.io/etcd/client/v3 v3.5.0-rc.0
)

replace github.com/coreos/bbolt => go.etcd.io/bbolt v1.3.3

retract (
    v1.0.0 // 哎呀，错误的实现版本
    v1.0.1 // 只是为了撤回版本，请使用v0.9.x版本
)
```

撤回一个意外发布的版本

撤回本版本

go.sum 文件

用来验证依赖库的可重复性、避免恶意更改或者意外更改。

```
① github.com/panicthis/A v1.2.0 h1:bv4JpK1Pl1UCYsKMzo+yG0RSNw7ZK+VM6qh03m/c690=
② github.com/panicthis/A v1.2.0/go.mod h1:haRL5lvZ5rM0mZ/WqdAv8EUViiVwyH7xWJrGLS2jx2w=

github.com/panicthis/B v1.2.0 h1:V+p3pmS3iA8Wtb9o4X7HhgqliV5xuKbiHmTeITWYxI4=
github.com/panicthis/B v1.2.0/go.mod h1:2LtkupHYVRVPzSwjYr9ZCq7s+oIp6t8aJ276qwPXurc=

github.com/panicthis/C v1.3.0/go.mod h1:bGWAUBhvWNlRKlXRkZ/GvJ4t1L9TViuVqt3CCiBGJRE=
github.com/panicthis/C v1.4.0 h1:mVo5rnq0AeVPpxhWFqsb6YErQbzyNJgv7fumlXiUDtc=
github.com/panicthis/C v1.4.0/go.mod h1:bGWAUBhvWNlRKlXRkZ/GvJ4t1L9TViuVqt3CCiBGJRE=

github.com/panicthis/D v1.2.0 h1:tYnu/e89BSzrIMKbpfafNY/aJBcbkrTHdn+fxdfm3OE=
github.com/panicthis/D v1.2.0/go.mod h1:j8I1cIC2pOEDl5kXp0TV+a0dsLUL4GmocSohNxGaz94=
```

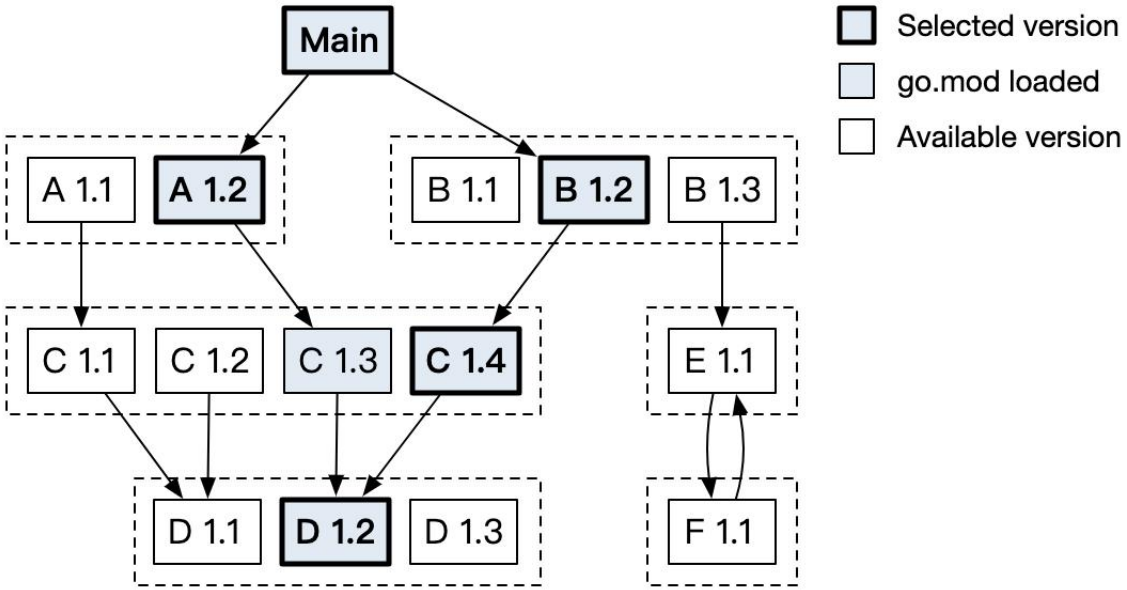
① module version 和 dependency file tree的 checksum

② module version/go.mod 和此文件的checksum。

MVS

Minimal Version Selection by Russ Cox

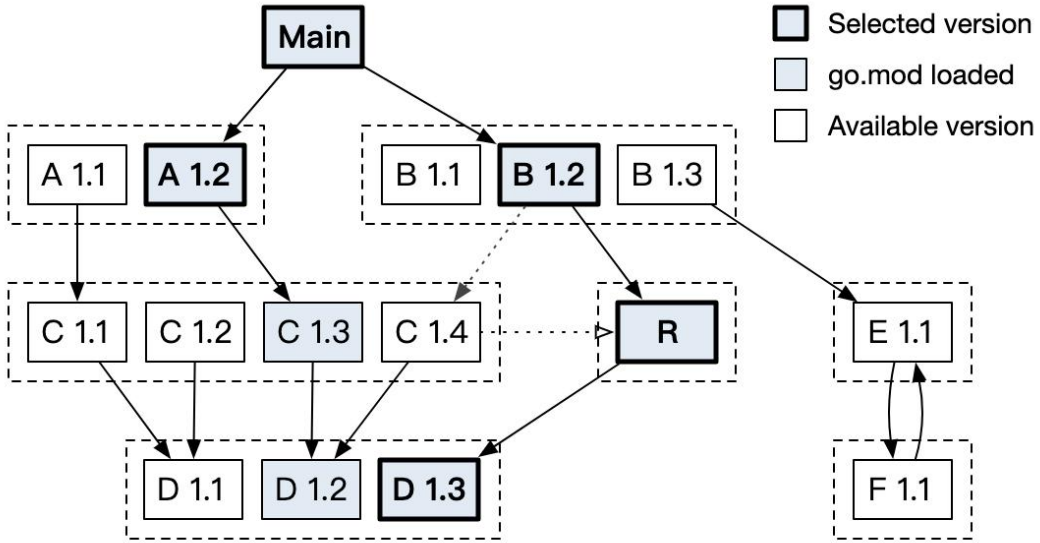
构建main module时每个依赖的版本选择算法。
go生成模块依赖图，遍历找到每个module最大的版本，这些版本的module就是满足所有需求的最小module版本



- A 1.2
- B 1.2
- C 1.4
- D 1.2

MVS

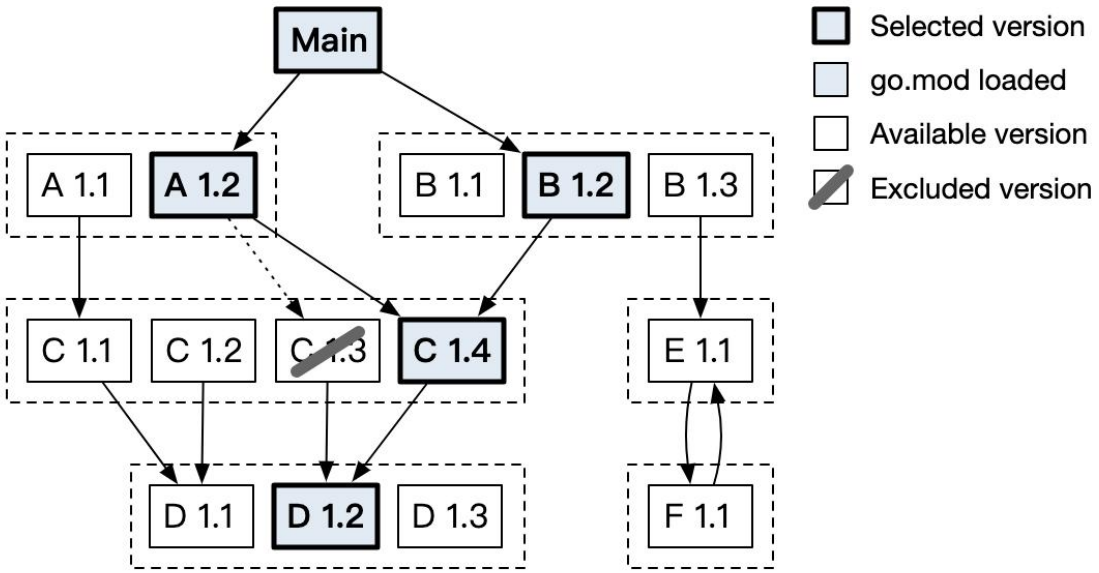
```
replace github.com/panicthis/C v1.4.0 => github.com/panicthis/R v1.0.0
```



- A 1.2
- B 1.2
- R 1.0
- D 1.3

MVS

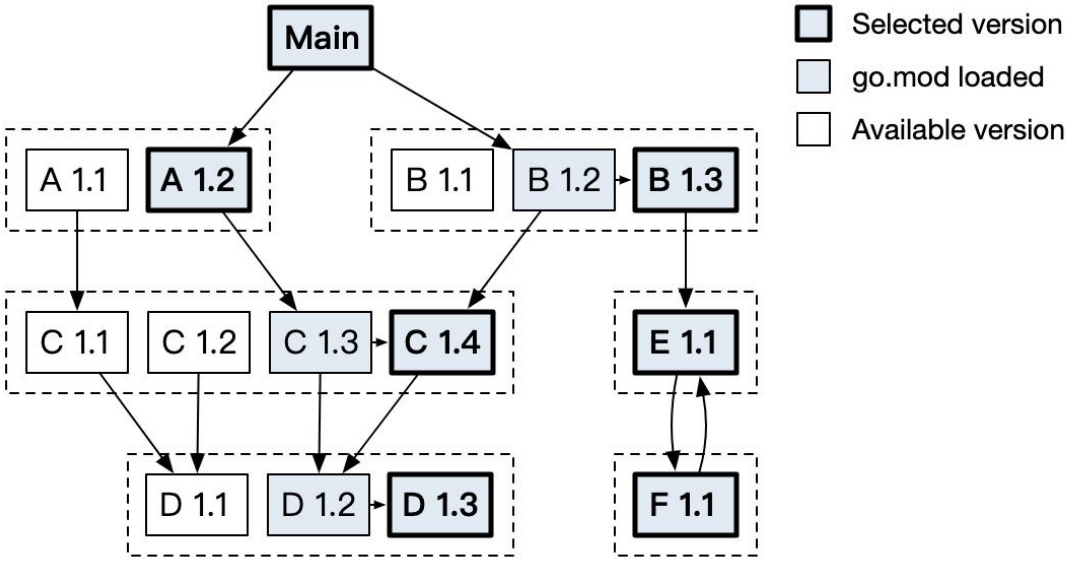
```
exclude github.com/panicthis/C v1.3.0
```



- A 1.2
- B 1.2
- C 1.4
- D 1.2

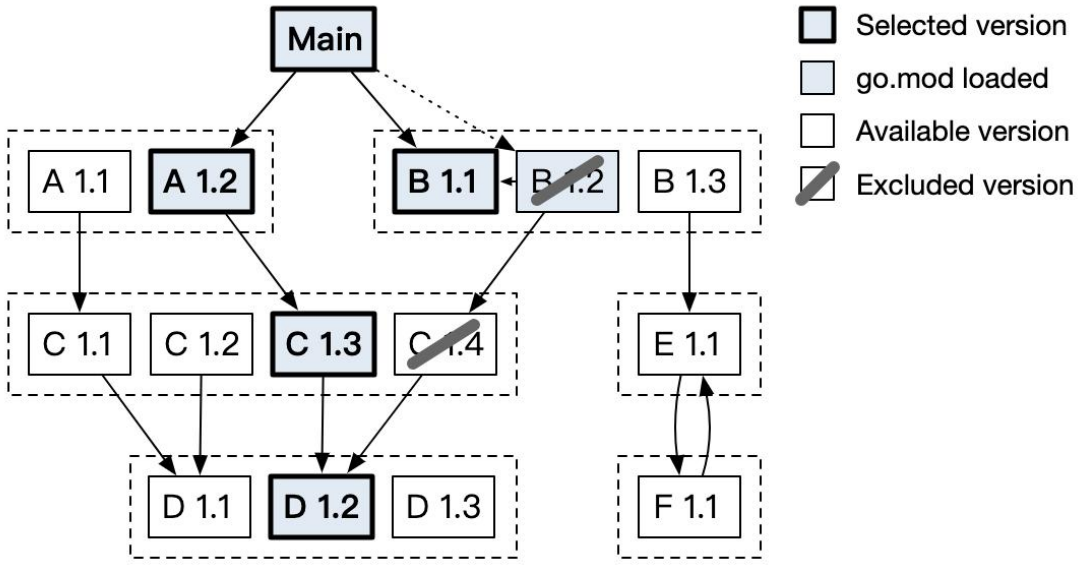
MVS

upgrade



- A 1.2
- B 1.3
- C 1.4
- D 1.3
- E 1.1
- F 1.1

MVS



C 1.4有问题，需要降级

- A 1.2
- B 1.1
- C 1.3
- D 1.2

non-canonical version

before

```
module github.com/panicthis/modefile2

go 1.16


require (
    github.com/panicthis/A v1.1.0
    github.com/panicthis/B v1.2
    github.com/panicthis/C v1
    github.com/panicthis/D latest
    github.com/panicthis/E upgrade
    github.com/panicthis/F patch
    github.com/panicthis/G/v2 v2.0.0
    github.com/panicthis/H 4f7657a
    github.com/panicthis/I HEAD
    github.com/panicthis/J master
    github.com/panicthis/K release-1.1.1
)
```

after

```
module github.com/panicthis/modefile2

go 1.16

Check for upgrades | Upgrade transitive dependencies | Upgrade direct dependencies
require (
    github.com/panicthis/A v1.1.0
    github.com/panicthis/B v1.2.1
    github.com/panicthis/C v1.4.0
    github.com/panicthis/D v1.3.0
    github.com/panicthis/E v1.1.0
    github.com/panicthis/F v1.1.0
    github.com/panicthis/G/v2 v2.0.0
    github.com/panicthis/H v0.0.0-20210620074350-4f7657a88770
    github.com/panicthis/I v0.1.0
    github.com/panicthis/J v1.0.0
    github.com/panicthis/K v1.1.1-0.20210620080121-20a593f94f90
)
```

non-canonical version

```
go get github.com/panicthis/A@v1.1.0
go get github.com/panicthis/B@v1.2
go get github.com/panicthis/C@v1
go get github.com/panicthis/D@latest
go get github.com/panicthis/E@upgrade
go get github.com/panicthis/F@patch
go get github.com/panicthis/G/v2@v2.0.0
go get github.com/panicthis/H@4f7657a
go get github.com/panicthis/I@HEAD
go get github.com/panicthis/J@master
go get github.com/panicthis/K@release-1.1.1
go get github.com/panicthis/L@none
```

go get可以使用non-canonical version

go mod xxx

命令	功能
go mod **download** [-x] [-json] [modules]	下载module到本地缓存
go mod **edit** [editing flags] [go.mod]	编辑go.mod文件
go mod **graph**	打印出module requirement graph
go mod **init** [module]	初始化一个module，创建go.mod
go mod **tidy** [-e] [-v]	修改go.mod和go.sum以匹配实际依赖
go mod **vendor** [-e] [-v]	增加build&test依赖到vendor文件夹下
go mod **verify**	校验本地缓存的文件自下载后未曾被修改
go mod **why** [-m] [-vendor] packages...	package或者module为什么被依赖，在哪里依赖

module-aware commands

- go build
- go fix
- go generate
- go get
- go install
- go list
- go run
- go test
- go vet
- go clean -modcache
- go version -m

- -mod=[mod,readonly,vendor,]
- -modcacherw
- -modfile=file.mod

module-aware commands

- go build
- go fix
- go generate
- go get
- go install
- go list
- go run
- go test
- go vet
- go clean -modcache

- -mod=[mod,readonly,vendor,]
- -modcacherw
- -modfile=file.mod

module-aware commands

- go get

```
go get [-d] [-t] [-u] [build flags] [packages]

$ go get -d golang.org/x/net
$ go get -d -u ./...
$ go get -d golang.org/x/text@v0.3.2
$ go get -d golang.org/x/text@master
$ go get -d golang.org/x/text@none
```

module-aware commands

- go list

```
go list -m [-u] [-retracted] [-versions] [list flags] [modules]

$ go list -m all
$ go list -m -versions example.com/m
$ go list -m -json example.com/m@latest
$ go list -m -u all

example.com/main/module
golang.org/x/old v1.9.9 (deprecated)
golang.org/x/net v0.1.0 (retracted) [v0.2.0]
golang.org/x/text v0.3.0 [v0.4.0] => /tmp/text
rsc.io/pdf v0.1.1 [v0.1.2]
```

第三部分

使用 Go Module

go-proxy & private modules

- **GOPROXY**: proxy列表
- **GOPRIVATE**: 私有module列表，前缀模糊匹配
- **GONOPROXY**: 不需要proxy代理的module列表
- **GONOSUMDB**: 不需要使用公共checksum库(sum.golang.org)检查的module列表
- **GOINSECURE**: 可以通过HTTP获取的module列表

GO111MODULE

- **on或者未设置**：启用module模式，即使go.mod不存在
- **off**：禁用module模式，使用GOPATH模式
- **auto**: go.mod存在或者任意的父目录存在，启用module模式

go-proxy & private modules

- 代理所有的**module**

```
GOPROXY=https://proxy.corp.example.com
GONOSUMDB=corp.example.com
```

- 混合代理

```
GOPROXY=https://proxy.corp.example.com,https://proxy.golang.org,direct
GONOSUMDB=corp.example.com
```

- 直接访问私有库

```
GOPRIVATE=corp.example.com
```

- 实践中

```
GOPROXY=https://proxy.corp.example.com,direct
GOPRIVATE=*.corp.example.com,*.research.example.com
```

创建一个项目

- 创建go.mod
  - go mod init: 如果在GOPATH下创建
  - go mod init: 如果在GOPATH,但是已经有一个仓库比如github.com/xxx/xxx关联了它
  - go mod init path: GOPATH外的裸项目
- go mod tidy 自动创建依赖
- go mod download 下载依赖
- go get github.com/a/b 引入新的依赖，或者go mod tidy重建go.mod
- 非稳定版本: git tag v0.a.b && git push origin v0.a.b
- 稳定版本: git tag v1.a.b && git push origin v1.a.b
- v2及以上版本： 慎重!!!
  - 分支法
  - 新目录法

既有项目 version < 2

- go mod init
- go mod tidy
- 非稳定版本: git tag v0.a.b && git push origin v0.a.b
- 稳定版本: git tag v1.a.b && git push origin v1.a.b

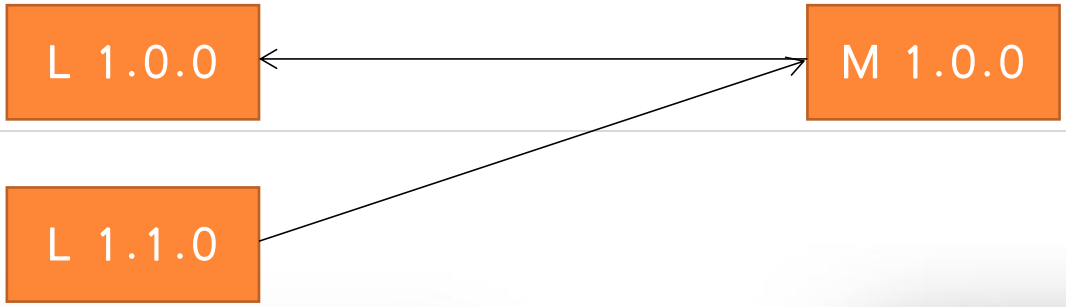
既有项目 version >= 2

- go mod init
- 提高major， 比如v3, v4,...
- 修改module path
- 修改代码引用，加上v3,v4,...后缀
- go mod tidy
- 发布新版本: git tag v3.0.0 && git push origin v3.0.0

第四部分

血泪史

循环依赖

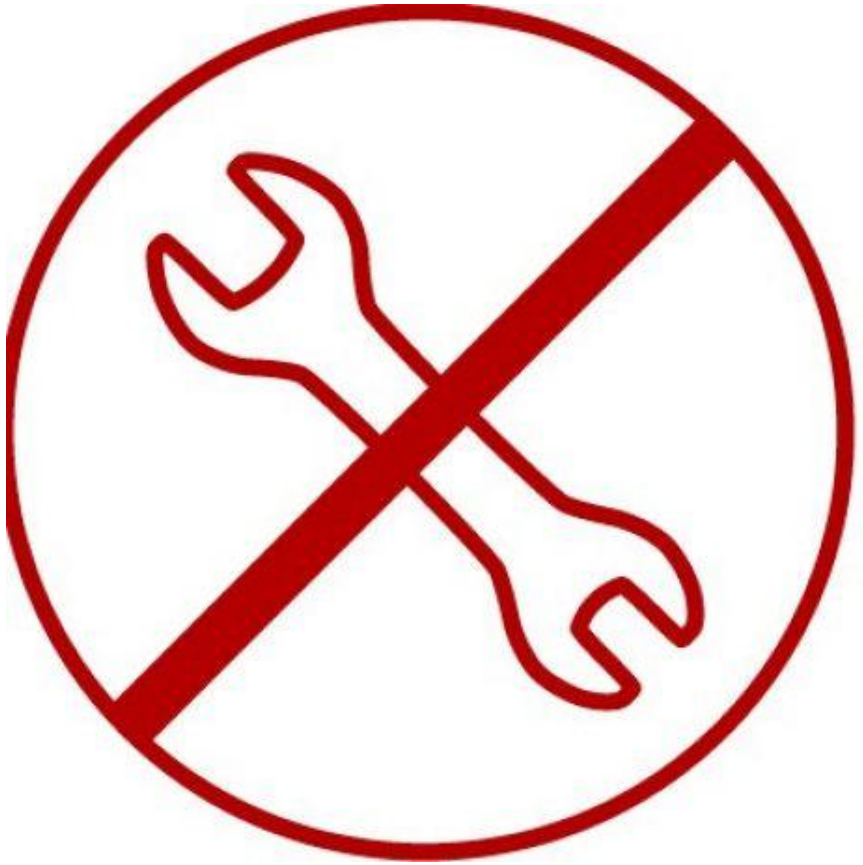



```go
package L

import (
    "fmt"

    "github.com/panicthis/M"
)

func Visit() {
    fmt.Println("L v1.1.0")
    M.Visit()
}
```

```go
package M

import (
    "fmt"

    "github.com/panicthis/L"
)

func Visit() {
    fmt.Println("M v1.0.0")
    L.Visit()
}
```

etcd --> prometheous-client --> prometheus-common --> go-kit -->etcd

不再维护的第三方



总是拉不下来的版本？历史旧账!

```
➜  modfile2 git:(master) ✗ go list -m -versions go.etcd.io/etcd
go.etcd.io/etcd v0.1.0 v0.1.1 v0.1.2 v0.2.0-rc0 v0.2.0-rc1 v0.2.0-rc2 v0.2.0-rc3 v0.2.0-rc4 v0.2.0
v0.3.0 v0.4.0 v0.4.1 v0.4.2 v0.4.3 v0.4.4 v0.4.5 v0.4.6 v0.4.7 v0.4.8 v0.4.9 v0.5.0-alpha.0 v0.5.0-
alpha.1 v0.5.0-alpha.2 v0.5.0-alpha.3 v0.5.0-alpha.4 v0.5.0-alpha.5 v2.0.0-rc.1+incompatible
v2.0.0+incompatible v2.0.1+incompatible v2.0.2+incompatible v2.0.3+incompatible v2.0.4+incompatible
......
v3.3.0+incompatible v3.3.1+incompatible v3.3.2+incompatible v3.3.3+incompatible v3.3.4+incompatible
v3.3.5+incompatible v3.3.6+incompatible v3.3.7+incompatible v3.3.8+incompatible v3.3.9+incompatible
v3.3.10+incompatible v3.3.11+incompatible v3.3.12+incompatible v3.3.13+incompatible
v3.3.15+incompatible v3.3.16+incompatible v3.3.17+incompatible v3.3.18+incompatible
v3.3.19+incompatible v3.3.20+incompatible v3.3.21+incompatible v3.3.22+incompatible
v3.3.24+incompatible v3.3.25+incompatible

➜  modfile2 git:(master) ✗ go get -v  go.etcd.io/etcd@v3.4.16
go get go.etcd.io/etcd@v3.4.16: unrecognized import path "go.etcd.io/etcd": https fetch: Get
"https://go.etcd.io/etcd?go-get=1": dial tcp 172.217.24.19:443: i/o timeout

➜  modfile2 git:(master) ✗ git ls-remote --tags git@github.com:etcd-io/etcd.git
20ca21a3f7122cf7caa91cb0e9b9c69be9279950    refs/tags/0
4ac392680e96b4b640c9e6c70cae21afdfb44fd4    refs/tags/api/v3.5.0
946a5a6f25c3b6b89408ab447852731bde6e6289    refs/tags/api/v3.5.0^{}
357571619c98b2575b48cdbaa31dbd6e2b542f17    refs/tags/api/v3.5.0-alpha.0
......
4a7ec4ab0820d6f6afa1b94e7a605b50a046408b    refs/tags/v3.3.25
2c834459e1aab78a5d5219c7dfe42335fc4b617a    refs/tags/v3.3.25^{}
bcb01e90afa04e641f033e2d34e6e25fd644cada    refs/tags/v3.4.0
898bd1351fcf6e0ebce8d7c8bbbbf3f3e42aa42c    refs/tags/v3.4.0^{}
43b9aa5f9569412d60097cdd07eab7910fd282ab    refs/tags/v3.5.0
946a5a6f25c3b6b89408ab447852731bde6e6289    refs/tags/v3.5.0^{}
e9ab4640c3c6a8f361eed72d37cb4346a7f03f14    refs/tags/v3.5.0-alpha.0
......
```

恼人的v2

```
➜  N git:(master) ls -l
total 8
-rw-r--r--  1 chaoyuepan  staff  42  6 25 10:00 go.mod
drwxr-xr-x  3 chaoyuepan  staff  96  6 25 09:57 v1
drwxr-xr-x  3 chaoyuepan  staff  96  6 25 09:57 v2
```

```go
package main

import (
    v1 "github.com/panicthis/N/v1"
    v2 "github.com/panicthis/N/v2/v2"
)

func main() {
    v1.Visit()
    v2.Visit()
}
```

module source tree too big

https://github.com/golang/go/issues/40780

```
$ go run main.go
go: finding module for package github.com/ikawaha/kagome-dict-ipa-neologd
go: downloading github.com/ikawaha/kagome/v2 v2.0.4
go: downloading github.com/ikawaha/kagome-dict-ipa-neologd v0.0.1
main.go:6:2: create zip: module source tree too large (max size is 524288000 bytes)
```

proxy有点懒

https://github.com/golang/go/issues/46369

```
$ go get github.com/stellar/mddiffcheck@latest
go get: downgraded github.com/stellar/mddiffcheck v0.0.0-20210525162741-cfbe68afa074 => v0.0.0-
20210525064236-6b157d2a519a

$ GOPROXY=direct go get github.com/stellar/mddiffcheck@latest
go get: upgraded github.com/stellar/mddiffcheck v0.0.0-20210525064236-6b157d2a519a => v0.0.0-
20210525162741-cfbe68afa074

$ go get github.com/stellar/mddiffcheck@latest
go get: downgraded github.com/stellar/mddiffcheck v0.0.0-20210525162741-cfbe68afa074 => v0.0.0-
20210525064236-6b157d2a519a

$ go get github.com/stellar/mddiffcheck@v0.0.0-20210525162741-cfbe68afa074
go get: upgraded github.com/stellar/mddiffcheck v0.0.0-20210525064236-6b157d2a519a => v0.0.0-
20210525162741-cfbe68afa074

$ go get github.com/stellar/mddiffcheck@latest
go get: downgraded github.com/stellar/mddiffcheck v0.0.0-20210525162741-cfbe68afa074 => v0.0.0-
20210525064236-6b157d2a519a
```

go mod graph失真

https://github.com/golang/go/issues/46365

Print module requirement graph

Usage:

```
go mod graph
```

Graph prints the module requirement graph (with replacements applied) in text form. Each line in the output has two space-separated fields: a module and one of its requirements. Each module is identified as a string of the form path@version, except for the main module, which has no @version suffix.

See https://golang.org/ref/mod#go-mod-graph for more about 'go mod graph'.

第五部分

Go Module 未来特性

Deprecated

```
go.mod
---
// Deprecated: use github.com/example/modtest/v2 instead.
github.com/example/modtest v1.5.7

$ go get http://github.com/example/modtest

go: module http://github.com/example/modtest is deprecated: use github.com/example/modtest/v2 instead.
```

lazy module loading

Proposal: Lazy Module Loading

Author: Bryan C. Mills (with substantial input from Russ Cox, Jay Conrod, and Michael Matloob)

Last updated: 2020-02-19

Discussion at https://golang.org/issue/36460.

Abstract

We propose to change `cmd/go` to avoid loading transitive module dependencies that have no observable effect on the packages to be built.

The key insights that lead to this approach are:

1. If *no* package in a given dependency module is ever (even transitively) imported by any package loaded by an invocation of the `go` command, then an incompatibility between any package in that dependency and any other package has no observable effect in the resulting program(s). Therefore, we can safely ignore the (transitive) requirements of any module that does not contribute any package to the build.
2. We can use the explicit requirements of the main module as a coarse filter on the set of modules relevant to the main module and to previous invocations of the `go` command.

Based on those insights, we propose to change the `go` command to retain more transitive dependencies in `go.mod` files and to avoid loading `go.mod` files for "irrelevant" modules, while still maintaining high reproducibility for build and test operations.

The End

谢谢