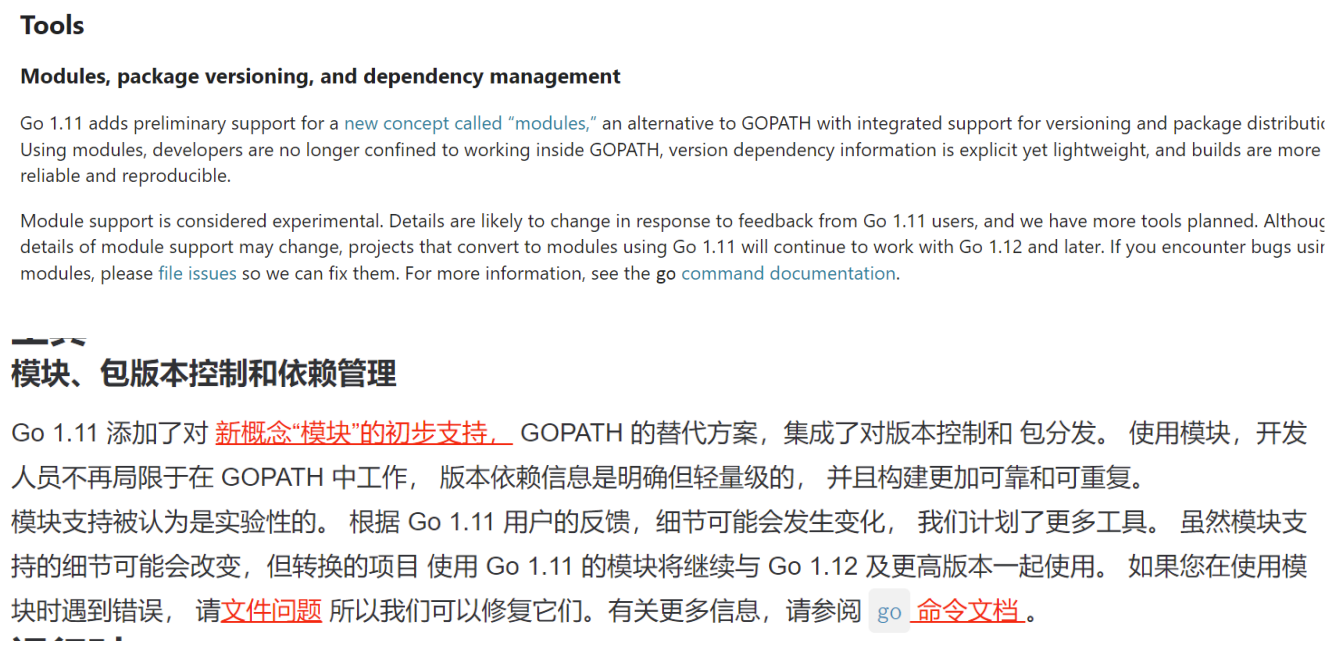
mod

发展历程

go 1.11 (2018-08-24)开始支持 gomod 的方式去管理包的依赖

具体的发行说明参考官方链接：https://go.dev/doc/go1.11 [所有版本的发行说明：https://go.dev/doc/devel/release]



Tools

Modules, package versioning, and dependency management

Go 1.11 adds preliminary support for a new concept called "modules," an alternative to GOPATH with integrated support for versioning and package distributi
Using modules, developers are no longer confined to working inside GOPATH, version dependency information is explicit yet lightweight, and builds are more reliable and reproducible.

Module support is considered experimental. Details are likely to change in response to feedback from Go 1.11 users, and we have more tools planned. Althouc
details of module support may change, projects that convert to modules using Go 1.11 will continue to work with Go 1.12 and later. If you encounter bugs usir
modules, please file issues so we can fix them. For more information, see the go command documentation.

模块、包版本控制和依赖管理

Go 1.11 添加了对 新概念"模块"的初步支持， GOPATH 的替代方案，集成了对版本控制和 包分发。 使用模块，开发人员不再局限于在 GOPATH 中工作， 版本依赖信息是明确但轻量级的， 并且构建更加可靠和可重复。
模块支持被认为是实验性的。 根据 Go 1.11 用户的反馈，细节可能会发生变化， 我们计划了更多工具。 虽然模块支持的细节可能会改变，但转换的项目 使用 Go 1.11 的模块将继续与 Go 1.12 及更高版本一起使用。 如果您在使用模块时遇到错误， 请文件问题 所以我们可以修复它们。有关更多信息，请参阅 `go` 命令文档 。

Q: GOPATH 方式存在的问题？

A 今天 go get pkg1 commit 了提交，第二天 pkg1 更新了，B 去 clone A 的项目执行 go get pkg1，此时 pkg1 变化了

同一个包，不同项目无法引用不同版本

vendor 目录，git 上传整份代码，git 库体积过大，升级包时，文件太多，MR review 不友好

基本概念

GO111MODULE 变量

`GO111MODULE` 环境变量

`GO111MODULE` 是一个环境变量，可以在使用 `go` 更改 Go 导入包的方式时进行设置。第一个痛点是，根据 Go 版本，其语义会发生变化。

`GO111MODULE` 与 Go 1.11 和 1.12

- 即使项目在您的 GOPATH 中， `GO111MODULE = on` 仍将强制使用 Go 模块。需要 `go.mod` 正常工作。
- `GO111MODULE = off` 强制 Go 表现出 GOPATH 方式，即使在 GOPATH 之外。
- `GO111MODULE = auto` 是默认模式。在这种模式下，Go 会表现
  - 当您在 `GOPATH` 外部时， 设置为 `GO111MODULE = on` ，
  - 当您位于 `GOPATH` 内部时，即使存在 `go.mod` , 设置为 `GO111MODULE = off` 。

每当您进入 GOPATH 中，并且您希望执行的操作需要 Go 模块 (例如， `Go get` 特定版本的二进制文件)，您需要执行以下操作：

```
GO111MODULE=on go get github.com/golang/mock/tree/master/mockgen@v1.3.1
```

Go 1.13 下的 `GO111MODULE`

在 Go 1.13 下， `GO111MODULE` 的默认行为 (`auto`) 改变了：

- 当存在 `go.mod` 文件时或处于 GOPATH 外， 其行为均会等同于于 `GO111MODULE=on` 。这意味着在 Go 1.13 下你可以将所有的代码仓库均存储在 GOPATH 下。
- 当处于 GOPATH 内且没有 `go.mod` 文件存在时其行为会等同于 `GO111MODULE=off` 。

所以为什么 `GO111MODULE` 哪里都是？！

我们已经知道了 `GO111MODULE` 对于选择是否开启 Go Modules 行为非常有用了，那么标题所述问题的原因就是：由于 `GO111MODULE=on` 允许你选择一个行为。如果不使用 Go Modules， `go get` 将会从模块代码的 master 分支拉取，而若使用 Go Modules 则你可以利用 Git Tag 手动选择一个特定版本的模块代码。

当我想要在 `gopls` （the Go Language Server，Go 语言服务器）的最新发布（latest 标签对应）版本和最近更新（master 分支对应）版本之间切换时我经常使用 `GO111MODULE=on` ：

```
GO111MODULE=on go get -u golang.org/x/tools/gopls@latest
GO111MODULE=on go get -u golang.org/x/tools/gopls@master
GO111MODULE=on go get -u golang.org/x/tools/gopls@v0.1
GO111MODULE=on go get golang.org/x/tools/gopls@v0.1.8
```

`@latest` 后缀将会使用 gopls 的 latest Git 标签。值得注意的是， `-u` 参数（意思是 'update'）对于 `@v0.1.8` 不再需要（因为这已经是一个固定的版本了，而更新一个固定的版本通常并没有意义）。还有一件事， 如果提供类似于 `@v0.1` 的后缀， `go get` 将会找寻这一版本号所对应的最后一个修订版本号的 tag。

go1.16 GO111MODULE 默认值是 on，并且增加了 retract 功能（版本撤回）

Module撤回

您是否曾经在Module版本准备好之前不小心发布过？或者您是否在版本发布后就发现了一个需要快速修复的问题？发布的版本中的错误是很难纠正的。为了保持Module构建的确定性，一个版本在发布后不能被修改。即使你删除或更改了版本标签，`proxy.golang.org`和其他代理可能已经有了原始版本的缓存。

Module作者现在可以使用`go.mod`中的`retract`指令撤回Module版本。撤回的版本仍然存在，并且可以被下载（所以依赖它的构建不会中断），但在解析`@latest`这样的版本时，`go`命令不会自动选择它，`go get`和`go list -m -u`会打印关于现有使用版本的警告。

例如，假设一个流行库`example.com/lib`的作者发布了`v1.0.5`，然后发现了一个新的安全问题。他们可以在他们的go.mod文件中添加如下指令。

```
// Remote-triggered crash in package foo. See CVE-2021-01234.
retract v1.0.5
```

接下来，作者可以标记并推送`v1.0.6`版本，即新的最高版本。在这之后，已经依赖v1.0.5的用户在检查更新或升级依赖的软件包时，就会被通知版本撤回。通知信息可能会包含来自`retract`指令上方注释的文字。

```
$ go list -m -u all
example.com/lib v1.0.0 (retracted)
$ go get .
go: warning: example.com/lib@v1.0.5: retracted by module author:
    Remote-triggered crash in package foo. See CVE-2021-01234.
go: to switch to the latest unretracted version, run:
    go get example.com/lib@latest
```

关于交互式的、基于浏览器的使用指南，请查看`play-with-go.dev`上的`Retract Module Versions`。可以查看`retract指令文档`以了解语法细节。

指令上面为撤回的原因，后面是具体的版本。如果涉及多版本，可以如下编写：

```
retract (
  v0.1.0
  v0.2.0
)
```

go mod 常用操作命令

用 go mod 初始化一个新项目

C/C++

```
mkdir demo
cd demo
go mod init demo
go: creating new go.mod: module demo
```

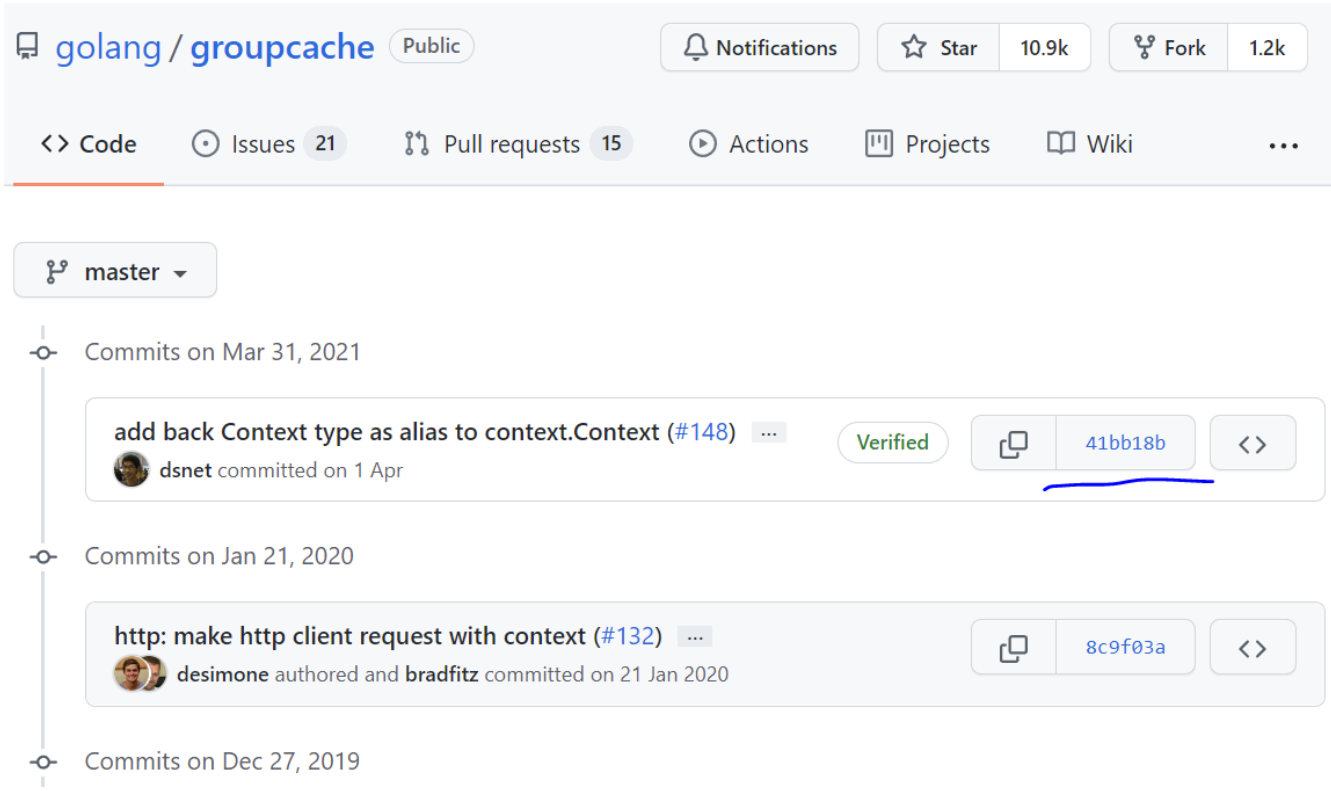
添加一个依赖包

C/C++

```
添加方式可参见上面的图

// 根据commit id添加依赖包的方式
go get github.com/golang/groupcache@41bb18b
```

```
github.com/gogo/protobuf v1.3.2
github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da
github.com/golang/mock v1.5.0
```



此时，go.mod 里会记录好依赖包，和对应的版本名

C/C++

```
1 cat go.mod
```

输出

C/C++

```
module demo

go 1.17

require (
        github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da // indirect
        github.com/golang/protobuf v1.5.2 // indirect
        google.golang.org/protobuf v1.26.0 // indirect
)
```

go.mod 里出现的关键字

- module - go mod init 指令定义的库名
- go - 要求 go 语言的最低版本，会影响到后面依赖库的下载
- require - 必备库，也就是代码中直接 import 的部分

```
github.com/Microsoft/hcsshim v0.8.22
github.com/armon/circbuf v0.0.0-20150827004946-bbbad097214e
github.com/auth0/go-jwt-middleware v1.0.1 // indirect
github.com/aws/aws-sdk-go v1.38.49
github.com/blang/semver v3.5.1+incompatible
github.com/boltdb/bolt v1.3.1 // indirect
github.com/clusterhq/flocker-go v0.0.0-20160920122132-2b8b7259d313
github.com/container-storage-interface/spec v1.5.0
github.com/containernetworking/cni v0.8.1
github.com/coredns/corefile-migration v1.0.14
github.com/coreos/go-oidc v2.1.0+incompatible
github.com/coreos/go-systemd/v22 v22.3.2
```

indirect：间接依赖，即 A -> B -> C 时，A 就是间接依赖 C 了

incompatible - 兼容v2及以上的版本号

上面我们已经讲过，如果一个库的tag为v1以上，如v2，就必须得创建一个v2的目录。例如

```
require example.com/new/thing/v2 v2.3.4
```

这就要求我们在项目example.com/new/thing下新建v2目录，再存放代码。但是，很多库往往只是升级个主版本号，并不会去新建目录、还需要迁移代码。为了兼容这个情况，就会引入+incompatible。例如

```
require example.com/new/thing v2.3.4+incompatible
```

- replace - 替换库，在重构时挺好用（比如某个开源组件有问题，内部 fork 了一版，直接 replace 即可）

```
replace (
        bazil.org/fuse => bazil.org/fuse v0.0.0-20160811212531-371fbbdaa898
        bitbucket.org/bertimus9/systemstat => bitbucket.org/bertimus9/systemstat v0.0.0-20180207000608-0eeff89b0690
        cloud.google.com/go => cloud.google.com/go v0.81.0

          k8s.io/api => ./staging/src/k8s.io/api
          k8s.io/apiextensions-apiserver => ./staging/src/k8s.io/apiextensions-apiserver
          k8s.io/apimachinery => ./staging/src/k8s.io/apimachinery
          k8s.io/apiserver => ./staging/src/k8s.io/apiserver
          k8s.io/cli-runtime => ./staging/src/k8s.io/cli-runtime
          k8s.io/client-go => ./staging/src/k8s.io/client-go
          k8s.io/cloud-provider => ./staging/src/k8s.io/cloud-provider
          k8s.io/cluster-bootstrap => ./staging/src/k8s.io/cluster-bootstrap
          k8s.io/code-generator => ./staging/src/k8s.io/code-generator
```

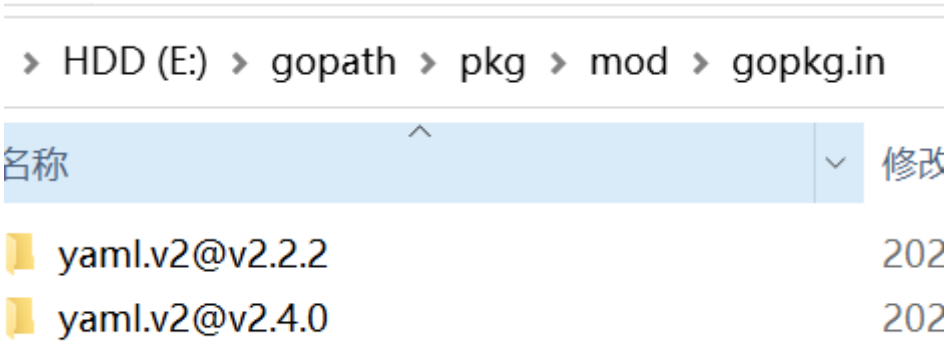
- retract 撤回版本，告诉调用本库的项目，部分版本有严重问题、不要引用

根据 go.mod 下载依赖包

C/C++

```
1 go mod download
```

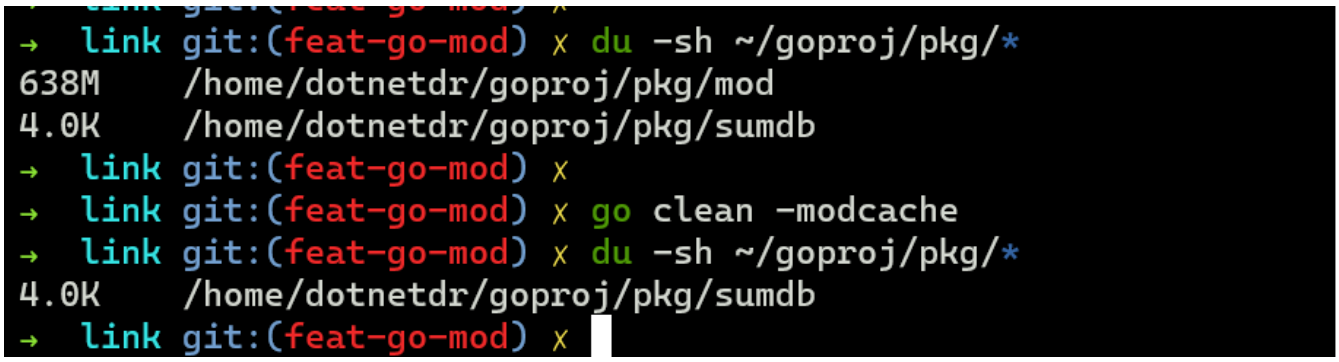
go mod 下载的包缓存在：$GOPATH/pkg/mod



清理下载好的 go pkg 缓存

C/C++

```
1 go clean -modcache
```



清理 go.mod 不在使用的依赖

C/C++

```
1 go mod tidy
```

将 gomod 项目依赖包存放到 vendor 目录，以便在离线环境下使用 GOPATH 方式编译

C/C++

```
1 go mod vendor
```

查看一个包是怎么被引用的

C/C++

```
1 go mod why ${pkg}
```

替换一个包

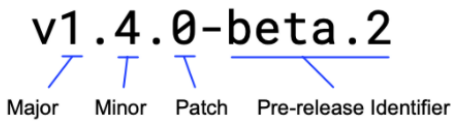
Plain Text

```
1 go mod edit -replace wps.cn/lib=ksogit.kingsoft.net/wps-zhiliao/backend/lib.git@v0.0.2-prep
```

go mod 版本格式

https://go.dev/doc/modules/version-numbers

https://semver.org/spec/v2.0.0.html 译文



The following table describes how the parts of a version number signify a module's stability and backward compatibility.

Version stage	Example	Message to developers
In development	Automatic pseudo-version number v**0**.x.x	Signals that the module is still **in development and unstable**. This release carries no backward compatibility or stability guarantees.
Major version	v**1**.x.x	Signals **backward-incompatible public API changes**. This release carries no guarantee that it will be backward compatible with preceding major versions.
Minor version	vx.**4**.x	Signals **backward-compatible public API changes**. This release guarantees backward compatibility and stability.
Patch version	vx.x.**1**	Signals **changes that don't affect the module's public API** or its dependencies. This release guarantees backward compatibility and stability.
Pre-release version	vx.x.x-**beta.2**	Signals that this is a **pre-release milestone, such as an alpha or beta**. This release carries no stability guarantees.

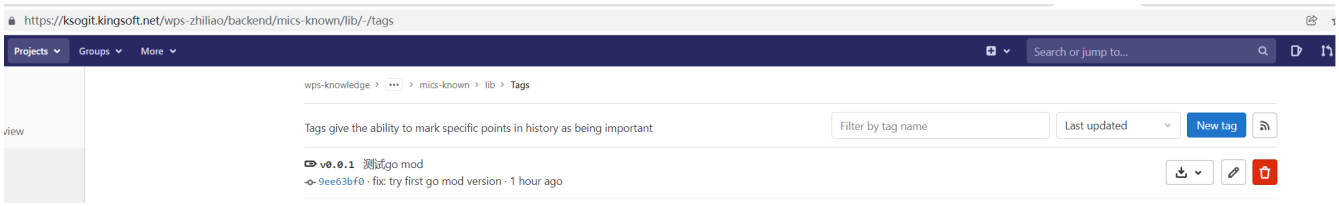
格式：

第一种：v${大版本}.${功能版本}.${fix 版本}。例如：v1.0.0

第二种：v${大版本}.${功能版本}.${fix 版本}-${预发行版本}。例如：v1.0.0-rc.1，v1.0.0-pre，v8.0.0-alpha.0，v1.0.0-分支名 .${整型序号}【目前采用】

第三种：v0.0.0-yyyyMMddHHmmss-${commitId}

然后我们要给对应的稳定代码版本打上 tag，项目 go get 时，可以指定对应的版本。同时打 tag 的好处可以针对引用库的改动做发行说明，留底方便作兼容性记录，为后续问题排查提供依据。



大版本号可以用来做重构以确保一个项目里，可以同时存在 v1,v2 这种情况的引用链。

```
main
  pkgA
     pkgX@v1
  pkgB
     pkgX@v2
这样保证了，pkgA,pkgB可以同时存在，pkgX做了大版本升级，可能有功能是废弃的
```

用以下例子提问

```
main
  pkgA
    pkgX@v1.0.0
  pkgB
    pkgX@v1.2.0
此时，对于main里pkgA，pkgB引用的pkgX是存在1个版本还是2个版本，如果是1个，选那个？
```

这里面涉及一个叫 Minimal Version Selection (mvs) 译文

go mod 版本的标准化做法

最典型的类库是 https://github.com/go-redis/redis 库

github.com/go-redis/redis/v8 指向 v8.x.x 的 tag 号

github.com/go-redis/redis/v7 指向 v7.x.x 的 tag 号

更多细节：https://go.dev/ref/mod

实践

接着需要在本地设置好 go mod 的环境变量

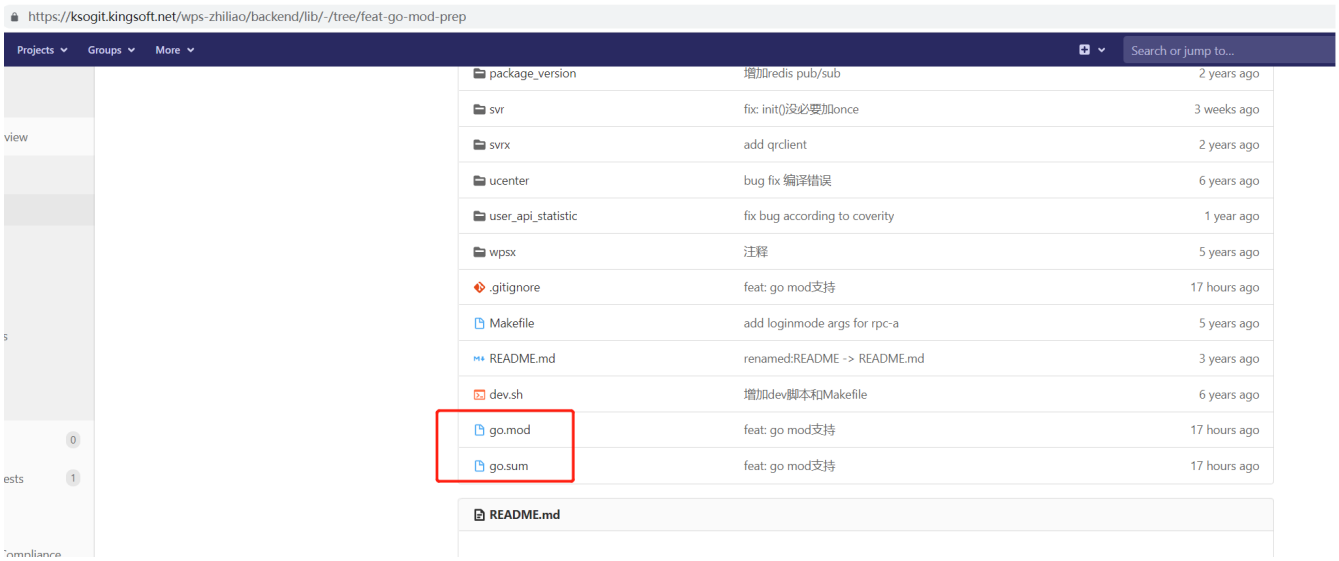
我本地使用的是 go1.17.2，按道理，生产环境所使用的 go1.14.2 也可以支持的

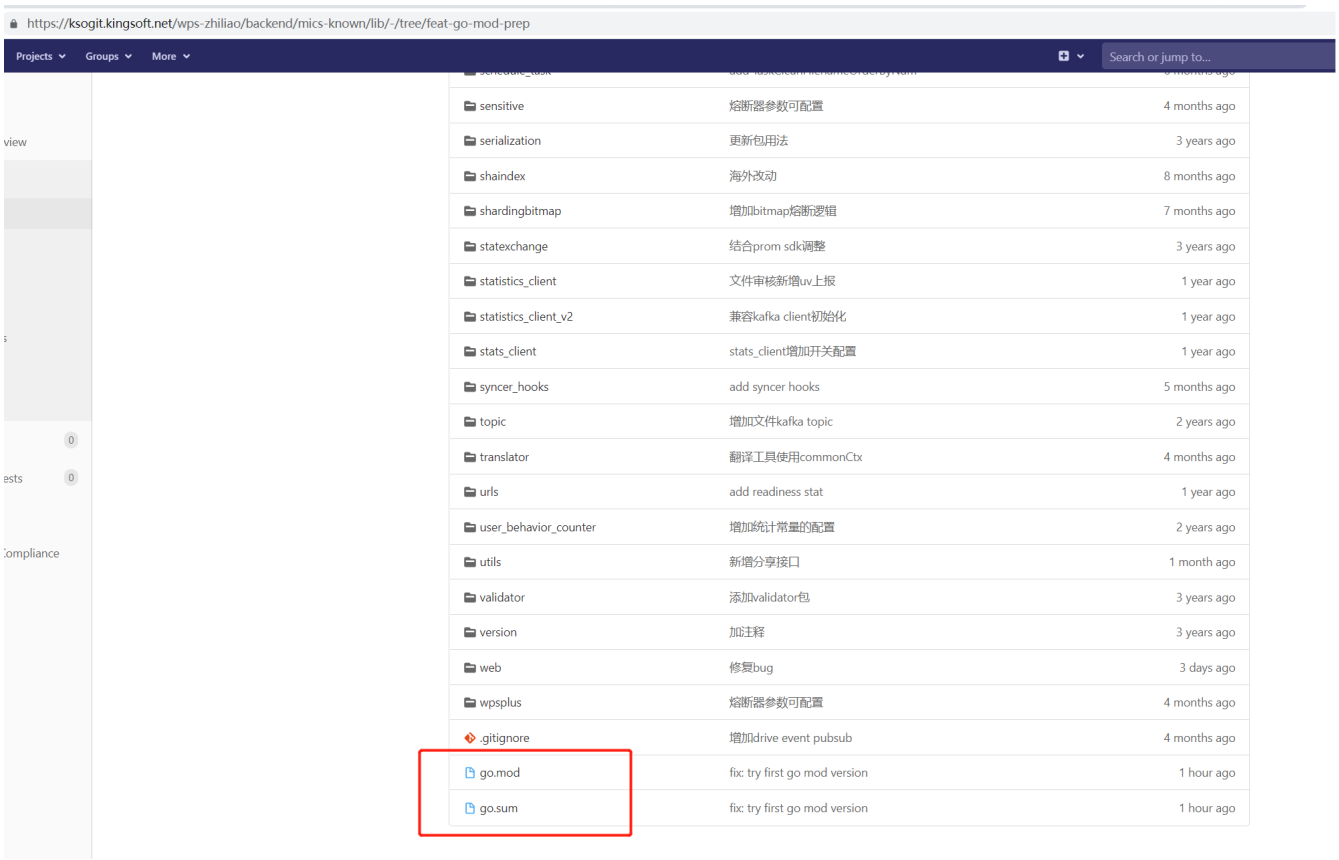
Plain Text

```
export GO111MODULE="on"
export GOPRIVATE="ksogit.kingsoft.net"
export GOSUMDB="off"
export CGO_ENABLED="0"
export GOPROXY="https://mirrors.wps.cn/go/"
export GOPATH="你的GOPATH，不用再添加wps.cn/lib/Godeps/_workspace"
```

关于 GOSUMDB，因为公司内部有私有魔改库，所以这类魔改库用的还是 github.com/xxx 包名，但是他们的 CHECHESUN 是不暴露在公网上的，最终我们需要手动把 SUMDB 给关掉

GOPATH 转换 go.mod 之前，需要确保依赖的 pkg 的根目录里包含 go.mod，go.sum





魔改库

公司内对公网外的所有第三方库的私有魔改版本代码都存放在：https://ksogit.kingsoft.net/cloudlibs

而且 GOPROXY 不要添加 https://goproxy.cn,direct 设置项

思考：魔改库最佳的实现方式是从原库 fork 一份出来，再用 go.mod replace 方式去引用

link 适配 go mod 过程

link 引用了 backend/lib 和 drive/lib 两个库，所以第一步需要做的事情就是将这两个 lib 库增加 go.mod, go.sum

我们在机器上配置好上面提到的环境变量，接着开始对 lib 进行 go mod 化适配

1.backend/lib 的 go mod 化

先给 backend 添加 go.mod

就遇到了 SetNewFileNameStyle 未定义，解决方法是升级到魔改库 github.com/minio/minio-go/v6@v6.0.1-wps

Plain Text

```
# wps.cn/lib/core/storage
core/storage/wps3.go:270:13: policy.SetNewFileNameStyle undefined (type *minio.PostPolicy has no field or method SetNewFileNameStyle)

/go/pkg/mod/github.com/uber/jaeger-client-go@v1.0.0-wps/metrics.go:100:35: too many arguments in call to factory.Namespace
        have (string, nil)
        want (metrics.NSOptions)
```

2.drive/lib 的 go mod 化

go get 包报错 go get disable "terminal prompt" by default

原因：Go get 命令默认禁用 terminal prompt，即终端提示

解决方案：环境设置 GIT_TERMINAL_PROMPT=1

C/C++

```
export GIT_TERMINAL_PROMPT=1
go get XXX
<此时会让你输入账户名和gitlab token>
```

更好的方式是添加 .netrc 文件

3.link 的 go mod 化

Plain Text

```
1 go mod edit -replace wps.cn/lib=ksogit.kingsoft.net/wps-zhiliao/backen
  d/lib.git@v0.0.1
2 go mod edit -replace wps.cn/drive/lib=ksogit.kingsoft.net/wps-zhiliao/b
  ackend/mics-known/lib.git@v0.0.1
```

注意一定要加 .git 结尾

编译通过了，运行的时候发现 grpc 生成的代码 panic

Plain Text

```
panic: mismatching message name: got group.RPCGroupXGetGroupCorpReques
t, want group.RPCGroupXGetGroupCreatorsByUserIdsRequest

goroutine 1 [running]:
google.golang.org/protobuf/internal/impl.legacyLoadMessageDesc(0x15e10a
0, 0x12193c0, 0x13e823b, 0x2f, 0x0, 0x0)
        google.golang.org/protobuf@v1.25.0/internal/impl/legacy_messag
e.go:136 +0x882
google.golang.org/protobuf/internal/impl.legacyLoadMessageInfo(0x15e10a
0, 0x12193c0, 0x13e823b, 0x2f, 0xc0001d57b0)
        google.golang.org/protobuf@v1.25.0/internal/impl/legacy_messag
e.go:48 +0xbd
google.golang.org/protobuf/internal/impl.Export.LegacyMessageTypeOf(0x1
5bcb20, 0x0, 0x13e823b, 0x2f, 0x12191e0, 0xc00020fb80)
        google.golang.org/protobuf@v1.25.0/internal/impl/legacy_export.
go:33 +0xa5
github.com/golang/protobuf/proto.RegisterType(0x15bcb20, 0x0, 0x13e823
b, 0x2f)
        github.com/golang/protobuf@v1.4.3/proto/registry.go:186 +0x4d
wps.cn/drive/lib/grpc/group.init.0()
        wps.cn/drive/lib@v0.0.1-prep/grpc/group/group.rpc.pb.go:3575 +0
x27a9
```

最后发现是分支合的时候，有 2 个提交合并冲突导致。

参考资料：

Go 语言技巧 - 5.【初探 Go Module】Go 语言的版本管理 https://www.bilibili.com/read/cv12121027

Go 模块解惑：到处都是 GO111MODULE ，这到底什么？ https://learnku.com/go/t/39086

最小版本抉择（MVS） https://learnku.com/docs/go-mod/1.17/minimal-version-selection/11441

go package查询网址 https://pkg.go.dev/