GO-10 01: Продвинутая работа с модулями

Описание:

В 3-м модуле вы изучили возможности менеджера зависимостей go mod. Давайте рассмотрим еще несколько особенностей.
Использование нескольких версий
Создаем репозиторий для нового модуля version и клонируем его на локальную машину:

```
$ git clone git@github.com:rebrainme/version.git
```

В директории version создаем файл version.go с содержимым:

```
// version.go
package version

import "fmt"

// Version - print current module version
func Version() {
      fmt.Println("v1.0.0")
}
```

Создаем модуль из пакета version:

```
~/version$ go mod init github.com/version
```

Сохраняем изменения, отправляем правки в удаленный репозиторий. Не забываем присвоить tag версии:

```
~/version$ git add .
~/version$ git commit -m "version 1.0.0 created"
[master 0496fae] version 1.0.0 created
 1 file changed, 8 insertions(+)
 create mode 100644 version.go
~/version$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 416 bytes | 416.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0)
To github.com:rebrainme/version.git
   03b11d8..0496fae  master -> master
~/version$ git tag v1.0.0
~/version$ git push --tags
Total 0 (delta 0), reused 0 (delta 0)
To github.com:rebrainme/version.git
 * [new tag]         v1.0.0 -> v1.0.0
```

Теперь у нас есть первая версия модуля version, которую мы можем использовать в своих проектах.
Создадим новый локальный модуль main для использования version со следующим содержимым:

```
// main.go
package main

import "github.com/rebrainme/version"

func main() {
      version.Version()
}
```

Инициализируем создание модуля в пакете main:

```
~/main$ go mod init main
go: creating new go.mod: module main
```

И запустим файл на выполнение:

```
~/main$ go run main.go
go: finding module for package github.com/rebrainme/version
go: downloading github.com/rebrainme/version v1.0.0
go: found github.com/rebrainme/version in github.com/rebrainme/version
v1.0.0
v1.0.0
```

Как вы заметили, из удаленного репозитория подтянулась зависимость на модуль version и программа успешно вывела версию на экран.
Версия модуля состоит из 3 элементов - МАЖОРНАЯ.МИНОРНАЯ.ПАТЧ:

- МАЖОРНАЯ версия - когда сделаны обратно несовместимые изменения API.
- МИНОРНАЯ версия - когда вы добавляете новую функциональность, не нарушая обратной совместимости.
- ПАТЧ-версия - когда вы делаете обратно совместимые исправления.

Давайте теперь поменяем патч-версию в модуле version и запушим правки с новым tag:

```
package version

import "fmt"

// Version - print current module version
func Version() {
      fmt.Println("v1.0.1")
}
```

Для того чтобы в модуле main подтянулась новая версия version, мы должны отредактировать файл go.mod:

```
// go.mod
module main

go 1.15

require github.com/rebrainme/version v1.0.1 // indirect
```

Следующий запуск подтянет новую версию модуля:

```
~/main$ go run main.go
go: downloading github.com/rebrainme/version v1.0.1
v1.0.1
```

Теперь мы решили модуль version изменить следующим образом - будем выводить версию как возвращаемый аргумент:

```
// version.go
package version

// Version - return current module version
func Version() string {
      return "v2.0.0"
}
```

Мы видим, что здесь нарушается обратная совместимость, следовательно, мы меняем мажорную версию проекта. С точки зрения Go модулей, мажорная версия — это совершенно другой пакет.
Изменим go.mod:

```
~/version$ go mod edit -module github.com/rebrainme/version/v2
```

Необходимо создать новую ветку с именем версии в репозитории, в нее перенесем правки и все запушим с новым tag:

```
~/version$ git checkout -b v2
~/version$ git add .
~/version$ git commit -m "version 2.0.0 created"
~/version$ git push --set-upstream origin v2
~/version$ git tag v2.0.0
~/version$ git push --tags
```

Теперь в модуле main можно использовать обе мажорные версии проекта. Отредактируем файлы следующим образом:

```
// main.go
package main

import (
	"fmt"

	v1 "github.com/rebrainme/version"
	v2 "github.com/rebrainme/version/v2"
)

func main() {
	v1.Version()
	fmt.Println(v2.Version())
}

// go.mod
module main

go 1.15

require github.com/rebrainme/version v1.0.1
require github.com/rebrainme/version/v2 v2.0.0
```

При запуске main.go:

```
~/main$ go run main.go
go: downloading github.com/rebrainme/version/v2 v2.0.0
v1.0.1
v2.0.0
```

Мы видим, что в проект подтянулись обе версии модуля version.

Замена зависимостей

Но что делать, если, к примеру, модуль стал недоступен по старому адресу? Для этого существует директива replace в файле go.mod.

Давайте скопируем папку version в проект main:

```
$ cp -r version main/version
```

Затем отредактируем файл version.go:

```
//version.go
package version

// Version - return current module version
func Version() string {
      return "local v2.0.0"
}
```

И в модуле main выполним команду:

```
~/main$ go mod edit -replace github.com/rebrainme/version/v2=./version
```

При этом в файл go.mod в main добавилась строчка замены:

```
// go.mod
module main

go 1.15

require github.com/rebrainme/version v1.1.0

require github.com/rebrainme/version/v2 v2.0.0

replace github.com/rebrainme/version/v2 => ./version
```

При запуске программы мы видим, что вместо v2 используется локально подменная версия модуля:

```
~/main$ go run main.go
v1.1.0
local v2.0.0
```

Что такое GOPROXY

Благодаря GOPROXY удается гарантировать загрузку модуля Go, даже если версии были уничтожены автором или отредактированы.

При разработке на Go до использования GOPROXY зависимости модули загружались непосредственно из их исходных репозиториев в системах VCS, таких как GitHub, Bitbucket, Bazaar, Mercurial или SVN. Установка GOPROXY перенаправляет запросы загрузки модуля Go в репозиторий кеша.
Возвращая модуль из кеша GOPROXY, он всегда предоставляет один и тот же код для запрошенной версии, даже если модуль недавно был неправильно изменен в репозитории VCS. Существуют разные способы использования GOPROXY, в зависимости от источника зависимостей модулей go, которые используются.
Публичный GOPROXY
Публичный GOPROXY - это централизованный репозиторий, доступный разработчикам Golang по всему миру. Он содержит модули Go с открытым исходным кодом, которые были предоставлены третьими сторонами в общедоступных репозиториях проектов VCS. Большинство из них предоставляются сообществу разработчиков Golang бесплатно.
Чтобы использовать его, необходимо установить:

```
$ export GOPROXY = https://gocenter.io
```

Приведенный выше параметр перенаправляет все запросы на загрузку модуля в GoCenter. Загрузки из общедоступного GOPROXY могут быть намного быстрее, чем напрямую из VCS, путем загрузки файла архива модуля. Помимо выполнения загрузок общедоступный GOPROXY может также предоставить разработчикам GoLang более подробную информацию о содержащихся в нем модулях.
Начиная с версии 1.13, GOPROXY по умолчанию равен https://proxy.golang.org,direct. Это значит, что сначала мы попробуем получить зависимость через proxy.golang.org (официальный прокси), а потом - выкачать напрямую.
Приватный прокси
Обычно проекты GoLang используют зависимости модулей как с открытым исходным кодом, так и частных. Некоторые пользователи применяют переменную среды GOPRIVATE, чтобы указать список путей, которые должны обходить GOPROXY и GOSUMDB (проверка контрольных сумм модулей), и загружать частные модули непосредственно из этих репозиториев VCS. Чтобы использовать общедоступный GOPROXY GoCenter вместе с частными модулями, установите переменные среды Golang:

```
$ export GOPROXY = https: //gocenter.io,direct
$ export GONOSUMDB = *. internal.mycompany.com
```

Также GONOSUMDB отключает проверку контрольных сумм (GONOSUMDB), так как GOSUMDB="sum.golang.org" ничего не знает о наших приватных модулях.

Полезные ссылки:

- Using Go Modules
- Введение в систему модулей Go
- Package Management With Go Modules: The Pragmatic Guide
- Why GOPROXY Matters and Which to Pick

Задание:

1. Создайте у себя модуль с любым названием (go mod init).
2. Создайте файлик main.go со следующим содержимым:

```
package main

import "gitlab.rebrainme.com/golang-workshop-users/greeter"

func main() {
	greeter.Greet()
}
```

3. Если не хотите каждый раз вводить свои логин с паролем - измените протокол доступа к репозиторию модуля с https (используется для наших репозиториев) принудительно на ssh:

```
git config --global url.git@gitlab.rebrainme.com:.insteadOf https://gitlab.rebrainme.com/
```

4. Попробуйте собрать ваш проект. Скорее всего у вас не получится - используйте GOPRIVATE.
5. И у вас, скорее всего, снова не получится - изучите интересный issue в официальном репозитории GitLab(https://gitlab.com/gitlab-org/gitlab/-/issues/29629) - подумайте, как можно применить replace.
6. В ответе пришлите содержимое вашего go.mod и go.sum.