如何使用go module导入本地包

原创 李文周 李文周 2020-02-28

`go module`是Go1.11版本之后官方推出的版本管理工具，并且从`Go1.13`版本开始，`go module`将是Go语言默认的依赖管理工具。最近`Go1.14`版本推出之后`Go module`功能已经被正式推荐在生产环境下使用了。

这几天已经有很多教程讲解如何使用`go module`，以及如何使用`go module`导入gitlab私有仓库，我这里就不再啰嗦了。

最近我发现很多小伙伴在群里问如何使用`go module`导入本地包，毕竟初学者刚开始接触package概念的时候都是先在本地创建一个包，然后尝试本地调用一下，但是在go module模式下调用本地包就很容易被卡住了。

本文就详细介绍下如何使用`go module`导入本地包。

前提

假设我们现在有`moduledemo`和`mypackage`两个包，其中`moduledemo`包中会导入`mypackage`包并使用它的`New`方法。

`mypackage/mypackage.go`内容如下：

```go
package mypackage

import "fmt"

func New(){
    fmt.Println("mypackage.New")
}
```

我们现在分两种情况讨论：

在同一个项目下

注意：在一个项目（project）下我们是可以定义多个包（package）的。

目录结构

现在的情况是，我们在`moduledemo/main.go`中调用了`mypackage`这个包。

```
moduledemo
├── go.mod
├── main.go
└── mypackage
    └── mypackage.go
```

导入包

这个时候，我们需要在`moduledemo/go.mod`中按如下定义：

```
module moduledemo

go 1.14
```

然后在`moduledemo/main.go`中按如下方式导入`mypackage`

```
package main

import (
    "fmt"
    "moduledemo/mypackage"  // 导入同一项目下的mypackage包
)
func main() {
    mypackage.New()
    fmt.Println("main")
}
```

举个例子

举一反三，假设我们现在有文件目录结构如下：

```
└── bubble
```

```
    ├── dao
    │   └── mysql.go
    ├── go.mod
    └── main.go
```

其中`bubble/go.mod`内容如下：

```
module github.com/q1mi/bubble

go 1.14
```

`bubble/dao/mysql.go`内容如下：

```
package dao

import "fmt"

func New(){
    fmt.Println("mypackage.New")
}
```

`bubble/main.go`内容如下：

```
package main

import (
    "fmt"
    "github.com/q1mi/bubble/dao"
)
func main() {
    dao.New()
    fmt.Println("main")
}
```

不在同一个项目下

目录结构

```
├── moduledemo
│   ├── go.mod
│   └── main.go
└── mypackage
    ├── go.mod
    └── mypackage.go

```

导入包

这个时候，`mypackage`也需要进行module初始化，即拥有一个属于自己的`go.mod`文件，内容如下：

```
module mypackage

go 1.14
```

然后我们在`moduledemo/main.go`中按如下方式导入：

```
import (
    "fmt"
    "mypackage"
)
func main() {
    mypackage.New()
    fmt.Println("main")
}
```

因为这两个包不在同一个项目路径下，你想要导入本地包，并且这些包也没有发布到远程的github或其他代码仓库地址。这个时候我们就需要在`go.mod`文件中使用`replace`指令。

在调用方也就是`packagedemo/go.mod`中按如下方式指定使用相对路径来寻找`mypackage`这个包。

```
module moduledemo

go 1.14

```

```
require "mypackage" v0.0.0
replace "mypackage" => "../mypackage"
```

举个例子

最后我们再举个例子巩固下上面的内容。

我们现在有文件目录结构如下：

```
├── p1
│   ├── go.mod
│   └── main.go
└── p2
    ├── go.mod
    └── p2.go
```

`p1/main.go`中想要导入`p2.go`中定义的函数。

`p2/go.mod`内容如下：

```
module liwenzhou.com/q1mi/p2

go 1.14
```

`p1/main.go`中按如下方式导入

```
import (
    "fmt"
    "liwenzhou.com/q1mi/p2"
)
func main() {
    p2.New()
    fmt.Println("main")
}
```

因为我并没有把`liwenzhou.com/q1mi/p2`这个包上传到`liwenzhou.com`这个网站，我们只是想导入本地的包，这个时候就需要用到`replace`这个指令了。

`p1/go.mod` 内容如下：

```
module github.com/q1mi/p1

go 1.14


require "liwenzhou.com/q1mi/p2" v0.0.0
replace "liwenzhou.com/q1mi/p2" => "../p2"
```

此时，我们就可以正常编译`p1`这个项目了。

说再多也没用，赶快自己动手试试吧。