go.mod 文件中的 //indirect

一、前言

在使用 `Go module` 过程中，随着引入的依赖增多，也许你会发现 `go.mod` 文件中部分依赖包后面会出现一个 `// indirect` 的标识。这个标识总是出现在 `require` 指令中，其中 `//` 与代码的行注释一样表示注释的开始，`indirect` 表示间接的依赖。

比如开源软件 Kubernetes（v1.17.0版本）的 go.mod 文件中就有数十个依赖包被标记为 `indirect`：

```
require (
    github.com/Rican7/retry v0.1.0 // indirect
    github.com/auth0/go-jwt-middleware v0.0.0-20170425171159-5493cabe49f7 // indirect
    github.com/boltdb/bolt v1.3.1 // indirect
    github.com/checkpoint-restore/go-criu v0.0.0-20190109184317-bdb7599cd87b // indirect
    github.com/codegangsta/negroni v1.0.0 // indirect
    ...
)
```

在执行命令 `go mod tidy` 时，Go module 会自动整理 `go.mod` 文件，如果有必要会在部分依赖包的后面增加 `// indirect` 注释。一般而言，被添加注释的包肯定是间接依赖的包，而没有添加 `// indirect` 注释的包则是直接依赖的包，即明确的出现在某个 `import` 语句中。

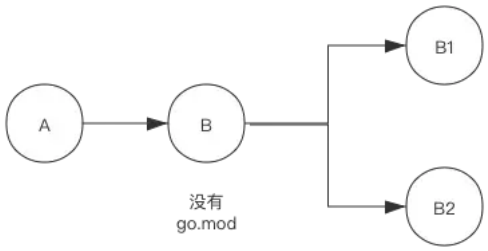
然而，这里需要着重强调的是：并不是所有的间接依赖都会出现在 `go.mod` 文件中。

间接依赖出现在 `go.mod` 文件的情况，可能符合下面所列场景的一种或多种：

- 直接依赖未启用 Go module
- 直接依赖go.mod 文件中缺失部分依赖

二、间接依赖出现情况 1： 直接依赖未启用 Go module

如下图所示，Module A 依赖 B，但是 B 还未切换成 Module，也即没有 `go.mod` 文件，此时，当使用 `go mod tidy` 命令更新A的 `go.mod` 文件时，B的两个依赖B1和B2将会被添加到A的 `go.mod` 文件中（前提是A之前没有依赖B1和B2），并且B1 和B2还会被添加 `// indirect` 的注释。



此时Module A的 `go.mod` 文件中require部分将会变成：

```
require (
    B vx.x.x
```

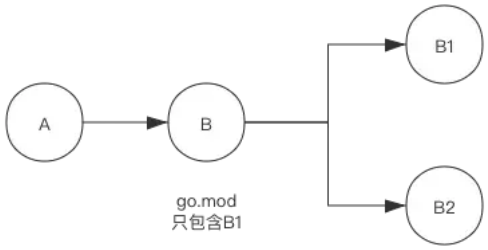
```
    B1 vx.x.x // indirect
    B2 vx.x.x // indirect
)
```

依赖B及B的依赖B1和B2都会出现在 `go.mod` 文件中。

三、间接依赖出现情况 2：直接依赖go.mod 文件中缺失部分依赖（不完整）

如上面所述，如果依赖B没有 `go.mod` 文件，则Module A 将会把B的所有依赖记录到A 的 `go.mod` 文件中。即便B拥有 `go.mod` ，如果 `go.mod` 文件不完整的话，Module A依然会记录部分B的依赖到 `go.mod` 文件中。

如下图所示，Module B虽然提供了 `go.mod` 文件中，但 `go.mod` 文件中只添加了依赖B1，那么此时A在引用B时，则会在A的 `go.mod` 文件中添加B2作为间接依赖，B1则不会出现在A的 `go.mod` 文件中。



此时Module A的 `go.mod` 文件中require部分将会变成：

```
require (
    B vx.x.x
    B2 vx.x.x // indirect
)
```

由于B1已经包含进B的 `go.mod` 文件中，A的 `go.mod` 文件则不必再记录，只会记录缺失的B2。

四、总结

1、为什么要记录间接依赖

在上面的例子中，如果某个依赖B 没有 `go.mod` 文件，在A 的 `go.mod` 文件中已经记录了依赖B及其版本号，为什么还要增加间接依赖呢?

我们知道Go module需要精确地记录软件的依赖情况，虽然此处记录了依赖B的版本号，但B的依赖情况没有记录下来，所以如果B的 `go.mod` 文件缺失了（或没有）这个信息，则需要在A的 `go.mod` 文件中记录下来。此时间接依赖的版本号将会跟据Go module的版本选择机制确定一个最优版本。

2、如何处理间接依赖

综上所述间接依赖出现在 `go.mod` 中，可以一定程度上说明依赖有瑕疵，要么是其不支持Go module，要么是其 `go.mod` 文件不完整。

由于Go 语言从v1.11版本才推出module的特性，众多开源软件迁移到go module还需要一段时间，在过渡期必然会出现间接依赖，但随着时间的推进，在 `go.mod` 中出现 `// indirect` 的机率会越来越低。

出现间接依赖可能意味着你在使用过时的软件，如果有精力的话还是推荐尽快消除间接依赖。可以通过使用依赖的新版本或者替换依赖的方式消除间接依赖。

3、如何查找间接依赖来源

Go module提供了 `go mod why` 命令来解释为什么会依赖某个软件包，若要查看 `go.mod` 中某个间接依赖是被哪个依赖引入的，可以使用命令 `go mod why -m <pkg>` 来查看。

比如，我们有如下的 `go.mod` 文件片断：

```
require (
    github.com/Rican7/retry v0.1.0 // indirect
    github.com/google/uuid v1.0.0
    github.com/renhongcai/indirect v1.0.0
    github.com/spf13/pflag v1.0.5 // indirect
    golang.org/x/text v0.3.2
)
```

我们希望确定间接依赖`github.com/Rican7/retry v0.1.0 // indirect`是被哪个依赖引入的，则可以使用命令`go mod why`来查看：

```
[root@ecs-d8b6 gomodule]# go mod why -m github.com/Rican7/retry
# github.com/Rican7/retry
github.com/renhongcai/gomodule
github.com/renhongcai/indirect
github.com/Rican7/retry
```

上面的打印信息中`# github.com/Rican7/retry`表示当前正在分析的依赖，后面几行则表示依赖链。

`github.com/renhongcai/gomodule`依赖`github.com/renhongcai/indirect`，而`github.com/renhongcai/indirect`依赖`github.com/Rican7/retry`。

由此我们就可以判断出间接依赖`github.com/Rican7/retry`是被`github.com/renhongcai/indirect`引入的。

另外，命令`go mod why -m all`则可以分析所有依赖的依赖链。