20 | 协作开发：模块化管理为什么能够提升研发效能?

任何业务，都是从简单向复杂演进的。而在业务演进的过程中，技术是从单体向多模块、多服务演进的。技术的这种演进方式的核心目的是**复用代码、提高效率**，这一讲，我会为你介绍 Go 语言是如何通过模块化的管理，提升开发效率的。

Go 语言中的包

什么是包

在业务非常简单的时候，你甚至可以把代码写到一个 Go 文件中。但随着业务逐渐复杂，你会发现，如果代码都放在一个 Go 文件中，会变得难以维护，这时候你就需要抽取代码，把相同业务的代码放在一个目录中。在 Go 语言中，这个目录叫作包。

在 Go 语言中，一个包是通过**package 关键字定义**的，最常见的就是**main 包**，它的定义如下所示：

```
package main
```

此外，前面章节演示示例经常使用到的 fmt 包，也是通过 package 关键字声明的。

一个包就是一个**独立的空间**，你可以在这个包里**定义函数**、**结构体**等。这时，我们认为这些函数、结构体是属于这个包的。

使用包

如果你想使用一个包里的函数或者结构体，就需要先**导入这个包**，才能使用，比如常用的 fmt包，代码示例如下所示。

```
package main
import "fmt"
func main() {
   fmt.Println("先导入fmt包，才能使用")
}
```

要导入一个包，需要使用 import 关键字；如果需要同时导入多个包，则可以使用小括号，示例代码如下所示。

```
import (
   "fmt"
   "os"
)
```

从以上示例可以看到，该示例导入了 fmt 和 os 这两个包，使用了小括号，每一行写了一个要导入的包。

作用域

讲到了包之间的导入和使用，就不得不提作用域这个概念，因为只有满足作用域的函数才可以被调用。

- 在**Java 语言**中，通过 public、private 这些修饰符修饰一个类的作用域；
- 但是在**Go 语言**中，并没有这样的作用域修饰符，它是通过首字母是否大写来区分的，这同时也体现了 Go 语言的简洁。

如上述示例中 fmt 包中的**Println 函数**：

- 它的首字母就是大写的 P，所以该函数才可以在 main 包中使用；
- 如果 Println 函数的首字母是小写的 p，那么它只能在 fmt 包中被使用，不能跨包使用。

这里我为你总结下 Go 语言的作用域：

- Go 语言中，所有的定义，比如函数、变量、结构体等，如果首字母是大写，那么就可以被其他包使用；
- 反之，如果首字母是小写的，就只能在同一个包内使用。

自定义包

你也可以自定义自己的包，通过包的方式把相同业务、相同职责的代码放在一起。比如你有一个 util 包，用于存放一些常用的工具函数，项目结构如下所示：

```
ch20
├── main.go
└── util
    └── string.go
```

在 Go 语言中，一个包对应一个文件夹，上面的项目结构示例也验证了这一点。在这个示例中，有一个 util 文件夹，它里面有一个 string.go 文件，这个 Go 语言文件就属于 util 包，它的包定义如下所示：

ch20/util/string.go

```
package util
```

可以看到，Go 语言中的包是代码的一种**组织形式**，通过包把相同业务或者相同职责的代码放在一起。通过包对代码进行归类，便于代码维护以及被其他包调用，提高团队协作效率。

init 函数

除了 main 这个特殊的函数外，Go 语言还有一个特殊的函数——init，通过它可以**实现包级别的一些初始化操作**。

init 函数没有返回值，也没有参数，它**先于 main 函数执行**，代码如下所示：

```
func init() {
   fmt.Println("init in main.go ")
}
```

一个包中可以有多个 init 函数，但是它们的执行顺序并不确定，所以如果你定义了多个 init 函数的话，要确保它们是**相互独立**的，一定**不要有顺序上的依赖**。

那么 init 函数作用是什么呢? 其实就是在导入一个包时，可以对这个包做一些必要的初始化操作，比如数据库连接和一些数据的检查，确保我们可以正确地使用这个包。

Go 语言中的模块

如果包是比较低级的代码组织形式的话，那么模块就是更高级别的，在 Go 语言中，一个模块可以包含很多个包，所以模块是相关的包的集合。

在 Go 语言中：

- 一个模块通常是**一个项目**，比如这个专栏实例中使用的 gotour 项目；
- 也可以是**一个框**架，比如常用的 Web 框架 gin。

go mod

Go 语言为我们提供了 go mod 命令来创建一个模块（项目），比如要创建一个 gotour 模块，你可以通过如下命令实现：

```
➜ go mod init gotour
go: creating new go.mod: module gotour
```

运行这一命令后，你会看到已经创建好一个名字为 gotour 的文件夹，里面有一个 go.mod 文件，它里面的内容如下所示：

```
module gotour
go 1.15
```

- 第一句是该项目的**模块名**，也就是 gotour；
- 第二句表示要编译该模块至少需要**Go 1.15 版本的 SDK**。

> 小提示：模块名最好是以自己的域名开头，比如 flysnow.org/gotour，这样就可以很大程度上保证模块名的唯一，不至于和其他模块重名。

使用第三方模块

模块化为什么可以提高开发效率？最重要的原因就是**复用了现有的模块**，Go 语言也不例外。比如你可以把项目中的公共代码抽取为一个模块，这样就可以供其他项目使用，不用再重复开发；同理，在 Github 上也有很多开源的 Go 语言项目，它们都是一个个独立的模块，也可以被我们直接使用，提高我们的开发效率，比如 Web 框架 gin-gonic/gin 。

众所周知，在使用第三方模块之前，需要先设置下 Go 代理，也就是 GOPROXY，这样我们就可以获取到第三方模块了。

在这里我推荐 goproxy.io 这个代理，非常好用，速度也很快。要使用这个代理，需要进行如下代码设置：

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.io,direct
```

打开终端，输入这一命令回车即可设置成功。

在实际的项目开发中，除了第三方模块外，还有我们**自己开发的模块**，放在了公司的 GitLab上，这时候就要把公司 Git 代码库的域名排除在 Go PROXY 之外，为此 Go 语言提供了GOPRIVATE 这个环境变量帮助我们达到目的。通过如下命令即可设置 GOPRIVATE：

```
# 设置不走 proxy 的私有仓库，多个用逗号相隔（可选）
go env -w GOPRIVATE=*.corp.example.com
```

以上域名只是一个示例，实际使用时你要改成自己**公司私有仓库的域名**。

一切都准备好就可以使用第三方的模块了，假设我们要使用 Gin 这个 Web 框架，首先需要安装它，通过如下命令即可安装 Gin 这个 Web 框架:

```
go get -u github.com/gin-gonic/gin
```

安装成功后，就可以像 Go 语言的标准包一样，通过 import 命令导入你的代码中使用它，代码如下所示：

```
package main
import (
   "fmt"
   "github.com/gin-gonic/gin"
)
func main() {
   fmt.Println("先导入fmt包，才能使用")
   r := gin.Default()
   r.Run()
}
```

以上代码现在还**无法编译通过**，因为还没有同步 Gin 这个模块的依赖，也就是没有把它添加到go.mod 文件中。通过如下命令可以添加缺失的模块：

```
go mod tidy
```

运行这一命令，就可以把缺失的模块添加进来，同时它也可以移除不再需要的模块。这时你再查看 go.mod 文件，会发现内容已经变成了这样：

```
module gotour
go 1.15
require (
   github.com/gin-gonic/gin v1.6.3
   github.com/golang/protobuf v1.4.2 // indirect
   github.com/google/go-cmp v0.5.2 // indirect
   github.com/kr/text v0.2.0 // indirect
   github.com/modern-go/concurrent v0.0.0-20180306012644-bacd9c7ef1dd // indirect
   github.com/modern-go/reflect2 v1.0.1 // indirect
   github.com/niemeyer/pretty v0.0.0-20200227124842-a10e7caefd8e // indirect
   github.com/stretchr/testify v1.6.1 // indirect
   golang.org/x/sys v0.0.0-20201009025420-dfb3f7c4e634 // indirect
   golang.org/x/xerrors v0.0.0-20200804184101-5ec99f83aff1 // indirect
   gopkg.in/check.v1 v1.0.0-20200227125254-8fa46927fb4f // indirect
   gopkg.in/yaml.v2 v2.3.0 // indirect
)
```

所以我们不用手动去修改 go.mod 文件，通过 Go 语言的工具链比如 go mod tidy 命令，就可以帮助我们自动地维护、自动地添加或者修改 go.mod 的内容。

总结

在 Go 语言中，包是同一目录中，编译在一起的源文件的集合。包里面含有函数、类型、变量和常量，不同包之间的调用，必须要首字母大写才可以。

而模块又是相关的包的集合，它里面包含了很多为了实现该模块的包，并且还可以通过模块的方式，把已经完成的模块提供给其他项目（模块）使用，达到了代码复用、研发效率提高的目的。

所以对于你的项目（模块）来说，它具有**模块 ➡ 包 ➡ 函数类型**这样三层结构，同一个模块中，可以通过包组织代码，达到代码复用的目的；在不同模块中，就需要通过模块的引入，达到这个目的。

编程界有个谚语：不要重复**造轮子**，使用现成的轮子，可以提高开发效率，降低 Bug 率。Go 语言提供的模块、包这些能力，就可以很好地让我们使用现有的轮子，在多人协作开发中，更好地提高工作效率。

最后，为你留个作业：基于模块化拆分你所做的项目，提取一些公共的模块，以供更多项目使用。相信这样你们的开发效率会大大提升的。