Go+ v1.x 设计第二期

import 过程与 Go+ 模块管理

许式伟@七牛

内容概要

- Go+ 的 import 过程
- Go+ 模块管理

Go+ 的 import 过程

import 语法

- import "pkgPath"
  - import Go standard package
    - import "fmt"
  - import Go+ standard package（Go+ 引入的）
    - import "gop/ast"
  - import customer package
    - import "<domain>/pkgPath"
  - import local package（Go+ 还没有实现的）
    - import "./localPkgPath"
    - import "../localPkgPath"
- import pkgName "pkgPath"
  - import _ "pkgPath"
  - import . "pkgPath"

ast & parser

- ast.Package (包)
  - Files map[string]*ast.File (文件列表)
    - Decls []ast.Decl (全局声明, interface)
      - *ast.FuncDecl
      - *ast.GenDecl (通用声明)
        - Tok token.Token // token.IMPORT, token.TYPE, token.VAR, token.CONST
        - Specs []ast.Spec (规格, interface)
          - *ast.TypeSpec // represents a type declaration
          - *ast.ValueSpec // represents a constant or variable declaration
          - *ast.ImportSpec // represents a single package import
            - Name *ast.Ident // local package name (including "."); or nil
            - Path *ast.BasicLit // import path
- parser
  - 略

cl & gox

- gox
  - func NewPackage(pkgPath, name string, conf *Config) *Package
  - func (*Package) Import(pkgPath string) *PkgRef
  - type PkgRef
    - Types *types.Package
  - func (*PkgRef) Ref(name string) Ref
  - func (*PkgRef) MarkForceUsed()
- cl
  - Go+ AST 转为对 gox DOM Writer 的函数调用

例子 Hello world

```
import "fmt"

func main() {
    fmt.Println("Hello world")
}
```

cl 对 gox 的调用序列

```
pkg := gox.NewPackage("main", "main", nil)
fmt := pkg.Import("fmt")

pkg.NewFunc(nil, "main", nil, nil, false).BodyStart(pkg).
    Val(fmt.Ref("Println")).
    Val("Hello world").
    Call(1).
End()
```

gox Import 过程

- (*Package).Import(pkgPath string) *PkgRef
  - 这个 import 过程是延迟加载的
    - 如果这个包没用被引用，这个 import 什么都不会发生

- (*PkgRef).Ref(name string)
  - 如果这个包还没有加载，则真正去加载它
    - 由于延迟加载的原因，多个包可能会一起被加载（实际上我们鼓励更多包一起加载）
  - 如果已经加载，查找相关的符号信息（lookup by name）

- (*PkgRef).MarkForceUsed()
  - 对于 import _ "pkgPath" 这种情况，虽然这个 pkgPath 被 import 而没被使用，但是还是要强制加载，这通过 MarkForceUsed 来实现
    - pkgRef := pkg.Import("pkgPath")
    - pkgRef.MarkForceUsed()

gox Import 过程（续）

- 包加载过程发生了什么？为什么鼓励多个包一起加载？
- golang.org/x/tools/go/packages
  - func Load(cfg *Config, patterns ...string) ([]*Package, error)
    - patterns 是要加载的 pkgPath 列表，叫 patterns 是因为支持通配符如 "go/..."
    - 为什么支持多个包一起加载？因为不同包依赖的基础包大同小异，中间有很多工作量是重复的，而当前 packages.Load 函数没缓存，速度很慢
  - Package
    - Imports map[string]*Package
      - 包的依赖关系
    - Types *types.Package
      - 依赖包的核心信息，通过它可以构建出 *gox.PkgRef 实例

packages.Load 设计上的问题

- 重复加载的开销
  - 单次 packages.Load 同时加载多个包可以一定程度上避免重复加载问题，但多次 packages.Load 调用之间会有很多重复工作
    - 所以 Load(nil, "go/token"); Load(nil, "go/ast"); Load(nil, "go/parser") 比合并为单次 Load(nil, "go/token", "go/ast", "go/parser") 会慢很多
    - 怎么解决加载慢的问题？
- 多次 packages.Load 导致相同的包有多份实例
  - 这导致不能简单用 T1 == T2 来判断两个类型是否是同一个类型
    - 例如 Load(nil, "go/token"); Load(nil, "go/ast"); Load(nil, "go/parser") 分开调用的话，parser.ParseDir 依赖的 fset *token.FileSet，我们如果是用单独调用 Load(nil, "go/token") 构造出来的 token.FileSet 类型实例，那么类型匹配会失败

怎么解决 packages.Load 问题？

- 怎么解决加载慢的问题？
  - 我们引入 packages.Load 的缓存
    - 但只要有一个包没被缓存住产生了一次 packages.Load 调用，就会比较慢
      - 原因还是多次 packages.Load 没办法共享依赖的包
      - 例如，标准库 fmt 包依赖 9 个其他的包
- 怎么解决多次 packages.Load 导致相同的包有多份实例？
  - 我们对 packages.Load 的结果进行 dedup，确保相同类型只有一个实例
    - 详见 gox/dedup.go
  - 当然还有更彻底的改法：直接改 packages.Load 本身，让多次 Load 之间可以共享依赖包
    - 欢迎大家去尝试这个改法

packages.Load 缓存

- 基本逻辑
  - packages.Load 前先查询要加载的包是否已经缓存过
    - 如果缓存过，直接返回结果
    - 如果没缓存过，先 packages.Load，再 dedup (确保相同类型只有一个实例)，再保存到缓存中
    - 详见 gox/import.go 的 func (*LoadPkgsCached) Load 函数
  - 程序退出时对所有依赖包的缓存进行持久化
    - 先序列化为 json，然后 zip 压缩，最后保存为 $ModRoot/.gop/gop.cache 文件
    - 详见 gox/persist.go
- 缓存更新
  - 怎么判断缓存已经过时，需要更新？

gop cache 更新问题

- 如果依赖包是 Go 标准库，认为不会变化
- 否则，计算**依赖包的指纹**，如果指纹发生变化就认为有依赖包变化

```
type pkgFingerp struct {
        files       []string // files to generate fingerprint
        fingerp     string   // package code fingerprint, or empty (delay calc)
        updated bool     // dirty flag is valid
        dirty       bool
}
```

- 怎么计算指纹？
  - 如果依赖包属于本 Module 内的（代码在 $ModRoot 下）
    - 根据该依赖包的 files（文件列表）每个文件的最后更新时间计算指纹
      - 详见 gox/import.go 的 func calcFingerp 函数
  - 如果依赖包不属于本 Module 内的（**暂未实现**）
    - 需要读 go.mod 文件检查该依赖包的版本，如果版本没变就认为没变
    - 当然还要考虑 replace 的情形，如果被 replace 为本地代码了，视同该依赖包属于本 Module 内的依赖来处理

解决 gop cache 更新的临时方案

- 发现 gop 编译时**依赖包的信息过时了**怎么办?
  - 手工删除 gop cache 文件:

```
rm $ModRoot/.gop/gop.cache
```

Go+ 模块管理

模块（Module）是什么？

- 模块（Module）不同于 包（Package）
  - 一个模块（Module）通常包含一个或多个包（Package）

- 模块（Module）是 Go/Go+ **代码发布**的单元
- 模块（Module）是 Go/Go+ **代码版本管理**的单元
  - 很自然一个道理：有**发布**才有**版本管理**

内容概要

- 如何 import 一个 Go+ 的包（Package）
  - 依赖一个包本质上是依赖一个模块
- 如何管理 Go+ 模块（Module）

如何 import 一个 Go+ 的包？

- 思考一个问题
  - 在 gox Import 过程中，传给 packages.Load 的 pkgPath 不是一个 Go 包而是一个 Go+ 的包，会怎样？
    - packages.Load 并不能识别 Go+ 包并对其进行加载

- 那么，该怎么解决这个问题？

如何 import 一个 Go+ 的包?

- 实现一个 Go+ 版本的 packages.Load
  - 详见 gop/cl/packages.go 的 func (*PkgsLoader) Load 函数

- 基本逻辑
  - 先调用 packages.Load 来 import 依赖包
    - 如果出错，error 信息会包含哪些包加载失败（这有点像 CPU 内存管理的缺页）
  - 将加载失败的 Go+ 包编译成 Go 包
    - 通过在 Go+ 包所在目录下生成 gop_autogen.go 文件
  - 重新调用 packages.Load 来 import 该依赖包
    - 此时它已经是一个合法的 Go 包，所以 packages.Load 可以成功

如何 import 一个 Go+ 的包？

- 问题：如果依赖的 Go+ 包还没有下载怎么办？
  - go mod tidy 可以下载所有 Go **依赖模块**（依赖包所在的模块）
    - 但它不能识别依赖的 Go+ 模块
  - gop mod tidy（**还没有实现**）

如何管理 Go+ 模块?

- Go+ 模块管理机制的选择
  - 基于 Go Module (go.mod)
  - 自己实现 Go+ Module (gop.mod)

基于 Go Module (go.mod)

- 当前我们就是这样干的

- 优势：简单
- 劣势：要把对 Go+ 标准库的依赖也加到 go.mod
  - 用起来非常不方便，很容易出现各种奇怪的问题

实现 Go+ Module (gop.mod)

- 是我们更希望的方式
- 基本逻辑（**尚未实现**）
  - 通过 gop.mod 来自动生成 go.mod
    - 每次 gop.mod 文件更新时，重新生成一次 go.mod
    - 对于 Go+ 模块，这个文件不需要入代码库
  - 自动生成的 go.mod 依赖的 Go+ 标准库是本地的
    - 通过 replace 指令来实现

练习题

上一讲练习题的状态更新

- 进阶练习（已完成）
  - for range start:end:step (https://github.com/goplus/gop/issues/865)

基础练习

- gop cache 更新问题
  - $ModRoot/.gop/gop.cache 当前只能感知到本Module 内的更新
    - 对于模块外的依赖如果发生了改变，并不能正确检测到
      - 临时方案：手工删除 gop.cache 文件
    - See https://github.com/goplus/gop/issues/891
- import "pkgPath"
  - 需要真去 import 这个 pkgPath 对于的包，并得到 pkgName
  - 当前简单先 pkgName = path.Base(pkgPath)
    - See https://github.com/goplus/gop/issues/881
- import local package
  - 实际生产环境很少有人用到，要实现它也只是从兼容性角度考虑
    - See https://github.com/goplus/gop/issues/814

进阶练习

- 实现 gop mod tidy
  - See https://github.com/goplus/gop/issues/889
- 实现 Go+ Module (gop.mod)
  - See https://github.com/goplus/gop/issues/861
- 改 packages.Load 本身，让多次 Load 之间可以共享依赖包
  - See https://github.com/goplus/gop/issues/810