Go+ v1.x 设计
第1篇

许式伟@七牛

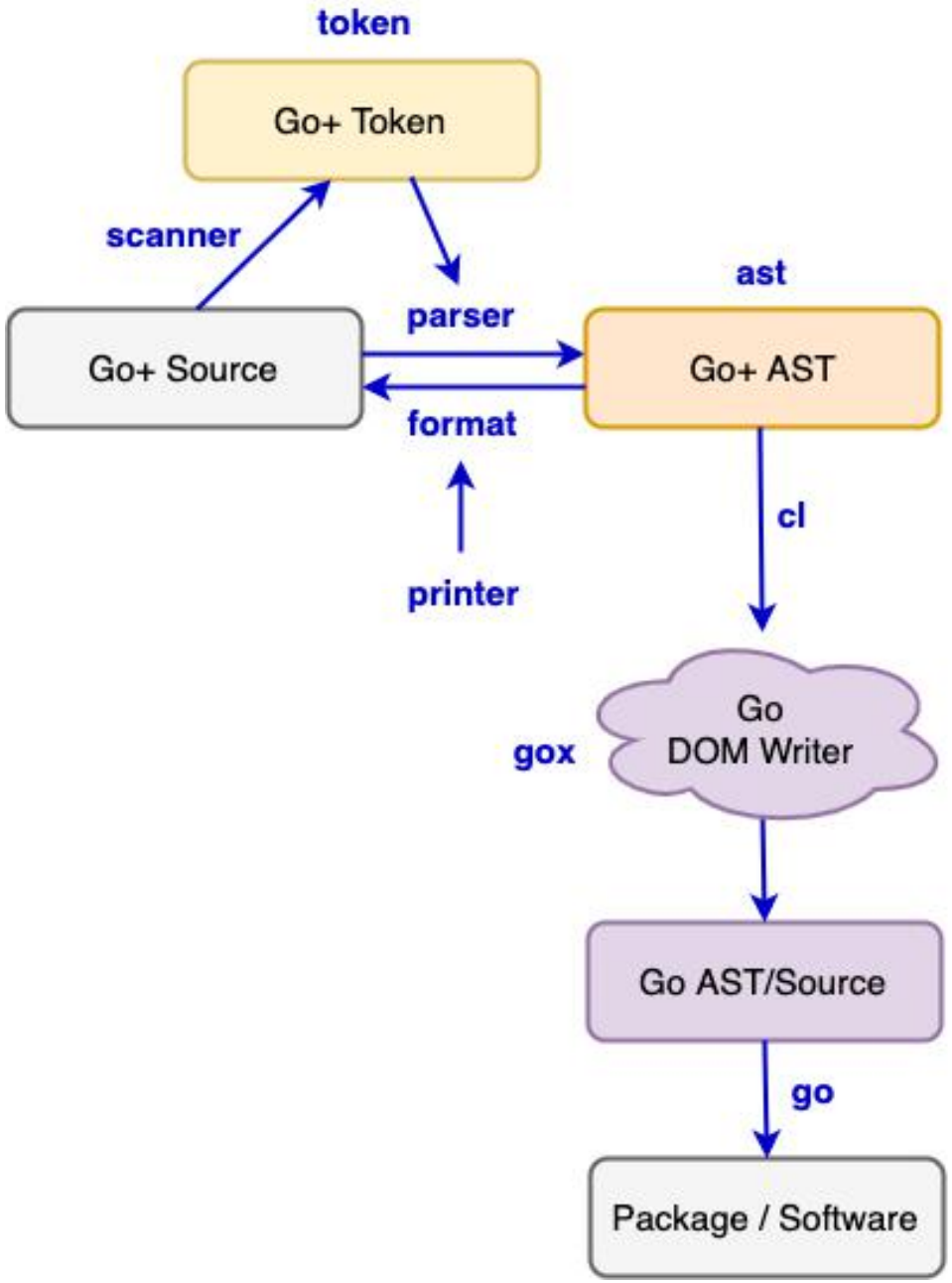
内容概要

- Go+ 整体架构
- 怎么给 Go+ 增加功能

Go+ 整体架构

Hello world

```
println "Hello world"
```



编译过程涉及：

- gop/token
  - github.com/goplus/gop/token
- gop/scanner
  - github.com/goplus/gop/scanner
- gop/parser
  - github.com/goplus/gop/parser
- gop/ast
  - github.com/goplus/gop/ast
- gop/cl
  - github.com/goplus/gop/cl
- gox
  - github.com/goplus/gox
- go
  - github.com/golang/go

token & scanner

- github.com/goplus/gop/token
  - token 是编译原理中的概念，类似于词
  - 源代码是字节流：[]byte
  - 需要通过 lex 过程变成 token 流：[]token

- github.com/goplus/gop/scanner
  - 即编译原理中的 lex 过程（词法分析）
  - 它通常不需要我们主动调用，而是由 parser 调用

ast & parser

- github.com/goplus/gop/ast
  - AST 的全称是抽象语法树（abstract syntax tree），是语言的 DOM
  - DOM 在文本处理中是很经典的模式
    - XML/json 等通用文档都有自己的 DOM，这和语言有自己的 AST 实质相同

- github.com/goplus/gop/parser
  - 编译原理中的 parser 过程（语法分析），负责将 token 流转为 DOM
    - 即：[]token => AST

parser 的使用方式

- func ParseDir(
  fset *token.FileSet,
  path string,
  filter func(os.FileInfo) bool,
  mode Mode) (pkgs map[string]*ast.Package, first error)

- fset: 主要用来记录文件偏移(offset)与行列号(line:col)之间的关系
- path: 源代码所在目录
- filter: 过滤文件用途，可以传 nil
- mode: 一些控制 parser 过程的 flags，可以简单传 0

- pkgs: 得到的 AST（因为一个目录下可能有多个pkg，所以是一个map）
- first: 在 parser 出错的时候，发生的第一个错误

cl & gox

- github.com/goplus/gop/cl
  - 编译（语义分析）：负责把 Go+ AST 转换为对 gox DOM Writer 的调用
  - 这意味着 cl 实现的是从 Go+ AST DOM 到另一种 DOM 的变换
    - 所以它本质上就是一种数据格式的变换

- github.com/goplus/gox
  - 是一个 Go 语言的 DOM Writer 组件，用来生成 Go AST
  - 所以 cl + gox 完成了 Go+ AST 到 Go AST 的转换

cl 的使用方式

- func NewPackage(
  pkgPath string,
  pkg *ast.Package,
  conf *Config) (p *gox.Package, err error)

- pkgPath: 要编译的目标 Go+ pkg 的 import 路径
- pkg: 要编译的目标 Go+ pkg 的 AST
- conf: 编译用的配置

- p: 生成的 gox DOM Writer，已经调用其接口完成了格式转换
- err: 在编译过程中如果发生错误，则返回所有的编译错误

gox 的使用方式

- 分两部分：一部分给 cl 灌数据（实现格式转换的），一部分是用来生成 Go AST/Source 的

- 实现格式转换的 (gox 大部分功能都是这一类)
  - func NewPackage(pkgPath, name string, conf *Config) *Package
    - NewFunc, NewType, NewVar, NewConstDecl, etc.
    - ...

- 生成 Go AST/Source 的
  - func ASTFile(pkg *Package, testingFile bool) *ast.File
    - 通过 pkg *gox.Package 得到 *go/ast.File（注意不是 Go+ AST 而是 Go AST 了）
    - testingFile: 一个 gox.Package 里面包含两个 DOM，一个是正常代码，一个是测试代码
  - func WriteFile(file string, pkg *Package, testingFile bool) (err error)
    - 将 pkg *gox.Package 写到磁盘生成 Go 源代码文件

go

- 有了 Go AST/Source 后，就可以用 go tools 来进行编译了
- 当然，也可以用一个 Go 的解释器去解析执行
  - 如 github.com/traefik/yaegi

gop run . 核心流程 (Hello world)

```
fset := token.NewFileSet()
pkgGops, err := parser.ParseDir(fset, ".", nil, 0)
pkgGox, err := cl.NewPackage("main", pkgGops["main"], nil)
err = gox.WriteFile("gop_autogen.go", pkgGox, false)
err = exec.Command("go", "run", "gop_autogen.go").Run()
```

怎么给 Go+ 增加功能

假设我们要新增语法

- 比如，实现三目运算符（cond ? expr1 : expr2）
  - 当然这只是一个例子，我们并没有实现三目运算符的计划

- 判断是否改变了 Go+ AST
  - 如果是，需要修改 gop/ast、gop/parser
    - 改 gop/ast 是增加新 AST 节点，比如叫 TernaryExpr
    - 改 gop/parser 是为了将文本 cond ? expr1 : expr2 转成 ast.TernaryExpr

- 修改编译器 gop/cl
  - cl 负责将新增的 ast.TernaryExpr 转为对 gox 的调用最后变成 Go AST
    - 如果是表达式，通常改 cl/expr.go
    - 如果是语句，通常改 cl/stmt.go

添加 ast.TernaryExpr

```
type TernaryExpr struct { // Cond ? X : Y
        Cond     Expr
        Question token.Pos
        X        Expr
        Colon    token.Pos
        Y        Expr
}

func (p *TernaryExpr) Pos() token.Pos { return p.Cond.Pos() }
func (p *TernaryExpr) End() token.Pos { return p.Y.End() }
func (*TernaryExpr) exprNode() {}
```

修改 gop/parser

[]token => ast.TernaryExpr

- https://github.com/goplus/gop/blob/main/parser/parser.go#L1973
- 实现：略

修改 gop/cl

- 考虑 ast.TernaryExpr 转换后的 Go 代码

```
// Cond ? X : Y

func() T {  // T 类型需要自动推导
        if Cond {
                return X
        }
        return Y
}()
```

cl.compileTernaryExpr

```
func compileTernaryExpr(ctx *blockCtx, v *ast.TernaryExpr) {
        ret := ctx.pkg.NewAutoParam("")  // gox 生成 DOM 有自动类型推导能力
        ctx.cb.NewClosure(nil, types.NewTuple(ret), false).BodyStart(ctx.pkg)
                ctx.cb.If()
                        compileExpr(ctx, v.Cond)
                        ctx.cb.Then()
                        compileExpr(ctx, v.X)
                        ctx.cb.Return(1)
                ctx.cb.End()
                compileExpr(ctx, v.Y)
                ctx.cb.Return(1)
        ctx.cb.End().Call(0)
}
```

写测试案例

- gop/ast: 不需要测试，基本上就是定义新数据结构而已
- gop/parser: 测试是否可以正确把 Go+ 代码转成期望的 AST
- gop/cl: 测试是否可以正确将 Go+ 代码转成期望的 Go 代码

如何测试 gop/parser

- 对于 parser 我们不需要写测试代码，只需要补充测试用例
  - 在 gop/parser 下有 _testdata 目录添加测试用例
  - 每个测试用例是一个目录（比如我们取 ternary），其中包含两个文件
    - 一个是 .gop 后缀的文件，例如我们叫 ternary.gop
    - 另一个文件必须叫 parser.expect，其内容是我们期望的 AST dump 结果
  - 例如，lambda 表达式我们加了三个测试用例
    - https://github.com/goplus/gop/blob/main/parser/_testdata/lambda1
    - https://github.com/goplus/gop/blob/main/parser/_testdata/lambda2
    - https://github.com/goplus/gop/blob/main/parser/_testdata/lambda3
- 谁在执行这些测试用例？
  - https://github.com/goplus/gop/blob/main/parser/parserdir_test.go
    - func TestFromTestdata(t *testing.T)
- 怎么调试测试用例？
  - TestFromTestdata 第一行 sel := "" 表示执行 _testdata 目录下所有测试用例
  - 可以将其改为 sel := "ternary"，这表示只执行我们添加的测试用例

如何测试 gop/cl

- 找到 gop/cl 下有一个叫 compile_test.go 的文件，在其中添加测试函数，比如 TestTernaryExpr

```
func TestTernaryExpr(t *testing.T) {
    gopCITest(t, `Go+ code`, `Go code`)
}
```

- 可以看出编译器 cl 的测试比 parser 还要更为简单，只需要把 Go+ 代码和期望生成的 Go 代码给出来就好了

添加功能的完整流程

- 访问 https://github.com/goplus/gop
- fork Go+ 代码到 https://github.com/your-id/gop
- clone Go+ 代码到本地
  - git clone git@github.com:your-id/gop.git
- cd gop
- 创建新功能分支
  - git checkout -b new-feature-name
- 添加功能代码
  - 修改 gop/ast, gop/parser, gop/cl 并测试通过
- 提交代码
  - git commit -a -m "new feature description"
  - git push
- 回到 https://github.com/your-id/gop，提交 pull request 给 Go+

划重点

- 通过以上例子可以看出：
  - 增加新功能通常需要修改 gop/ast, gop/parser 和 gop/cl
  - 由于 gox DOM Writer 强大的类型自动推导和 Go AST 生成能力，给 Go+ 增加新功能时 cl 过程变得轻松很多

练习题

- 基础练习
  - 实现三目运算符（cond ? expr1 : expr2）

- 进阶练习
  - import local package (https://github.com/goplus/gop/issues/814)
  - for range start:end:step (https://github.com/goplus/gop/issues/865)
  - support cgo (https://github.com/goplus/gop/issues/809)

欢迎反馈

- 如果大家在联系过程中遇到问题，欢迎通过以下途径反馈：
  - Go+ 用户组
    - 如果你在某个 Go+ 用户组的微信群，可以直接在群里面反馈问题
  - 知识星球搜索《Go+ 公开课》，并在其中提问：
    - https://wx.zsxq.com/dweb2/index/group/48844528441888