Go generate

The good, the bad, the better

Intro

- Introduced in Go 1.4
- Write code that writes code
- It is for package authors, not clients
- Abusing comments?
  - https://dave.cheney.net/2013/10/12/how-to-use-conditional-compilation-with-the-go-build-tool
- New concept ?

Syntax

```
// Code generated by go generate; DO NOT EDIT.

//go:generate cmd

$ go generate
```

Use cases

- Other sources: gRPC, Swagger
  - Makefile, bash ?
- Complements existing code
  - golang.org/x/tools/cmd/stringer
- I need XYZ language feature
  - where are generics in go ?! :D :D

Ugly ?

machine-generated code is often ugly ? can it be human friendly?

```
func (p *Mailer) Delete(index int) error {
    if index < 0 || index >= len(p.data) {
        return errors.ErrFetch
    }
    copy(p.data[index:], p.data[index+1:])
    p.data[len(p.data)-1] = types.Mailer{}
    p.data = p.data[:len(p.data)-1]
    return nil
}
```

Problem

Problem

github.com/haproxytech/config-parser

~100 parsers

```
type ParserInterface interface {
    Init()
    Parse(line string, parts, previousParts []string, comment string) (changeState string, err error)
    GetParserName() string
    Get(createIfNotExist bool) (common.ParserData, error)
    GetOne(index int) (common.ParserData, error)
    Delete(index int) error
    Insert(data common.ParserData, index int) error
    Set(data common.ParserData, index int) error
    Result() ([]common.ReturnResultLine, error)
}
```

The bigger the interface,
the weaker the abstraction.

— Rob Pike, Go Proverbs

Problem

- Same interface, most of the methods looks the same
- Two types
  - one-liner - only one line to parse
  - multiline parsers - can consume multiple lines
- Much or the methods can share same code
- Only Parse() and Result() are unique to each parser
- Go: No inheritance for methods, just embedding

Architecture

- Not ideal for go generate
- Types in separate package/folder
- Tests in separate package
  - Black box testing
    - Workaround for cyclic dependencies in tests
    - Use package from ‘outside’, more like end user / client
    - Test are hard to write -> tests are hard to use

Concept of go generate

- Go generate - usage limited only to one package/folder

README.md

use
$ *go run go-generate.go $(pwd) ?!*

- Not a `go generate` way, but similar

Define type, structure, tests in one place

types

types.go

parsers

tests

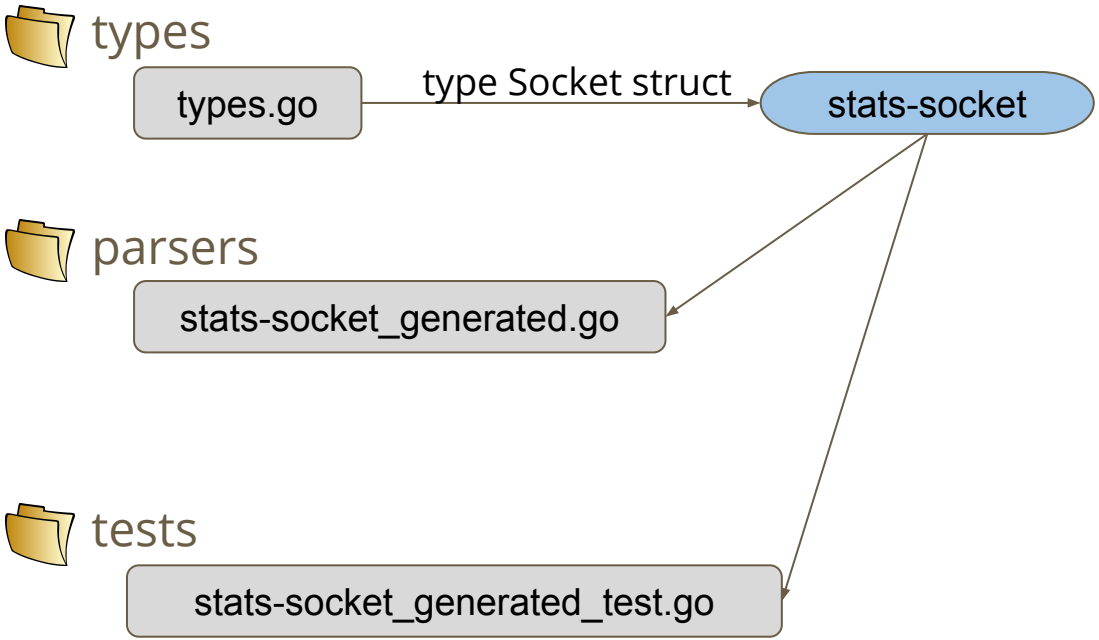
Define type, structure, tests in one place

types

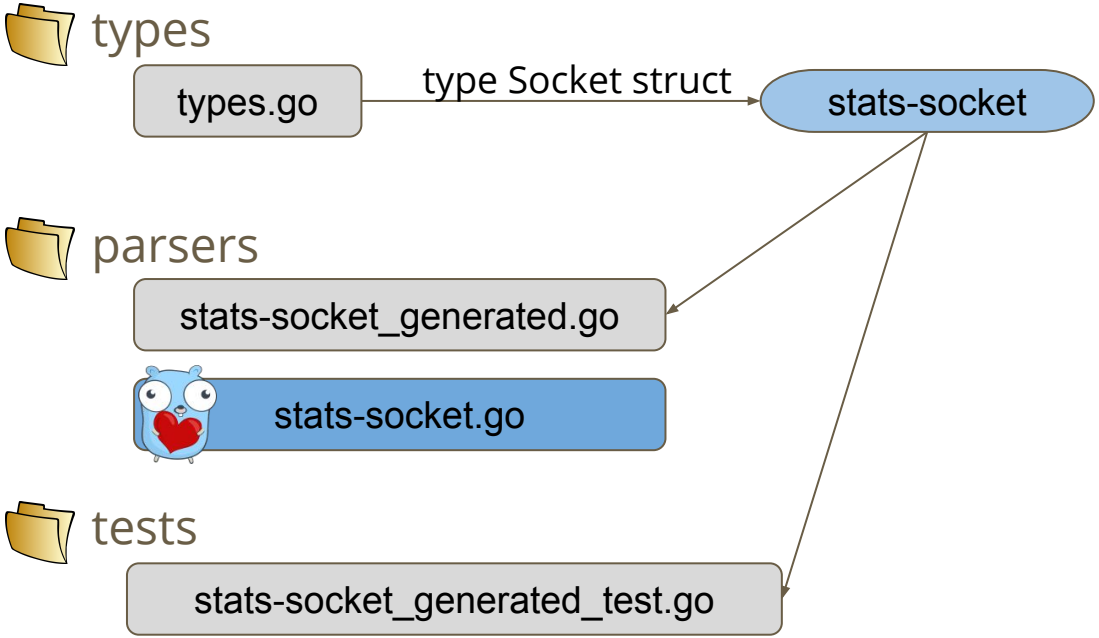
parsers

tests

Define type, structure, tests in one place



Define type, structure, tests in one place



Code

Example

Use comments in similar manner as go

```
//sections:backend
//name:stats socket
//is-multiple:true
//test:ok:stats socket 127.0.0.1:8080
//test:ok:stats socket 127.0.0.1:8080 mode admin
//test:ok:stats socket /some/path/to/socket
//test:ok:stats socket /some/path/to/socket mode admin
//test:fail:stats socket /some/path/to/socket mode
//test:fail:stats socket
type Socket struct {
    Path    string //can be address:port
    Params  []params.BindOption
    Comment string
}
```

go-generate.go

- Start with *go run go-generate.go $(pwd)*

```
// +build ignore
...
func main() {
    ...
    dir, err := filepath.Abs(filepath.Dir(os.Args[0]))
    ...
    generateTypes(dir)
    ...
}
```

go-generate.go

```
type Data struct {
    ParserMultiple      bool
    ParserName          string
    ParserSecondName    string
    StructName          string
    ParserType          string
    ParserTypeOverride  string
    NoInit              bool
    NoParse             bool
    NoGet               bool
    IsInterface         bool
    Dir                 string
    ModeOther           bool
    TestOK              []string
    TestFail            []string
}
```

Multiple flags to customize parsers

go-generate.go

```
func generateTypes(dir string) {
    dat, err := ioutil.ReadFile("types/types.go")
    if err != nil {
        log.Println(err)
    }
    lines := common.StringSplitIgnoreEmpty(string(dat), '\n')

    parserData := Data{}
    for _, line := range lines {
        if strings.HasPrefix(line, "//sections:") {
            //log.Println(line)
        }
        ...
```

go-generate.go

```
        ...
        filePath := path.Join(dir, "parsers", cleanFileName(filename)+"_generated.go")
        ...
        err = typeTemplate.Execute(f, parserData)
        ...
        ...
        filePath = path.Join(dir, "tests", cleanFileName(filename)+"_generated_test.go")
        ...
        err = testTemplate.Execute(f, parserData)
        ...
    }
}
```

Results

- Auto generated tests
  - Write only what you want to test, code for test is automated
- Most of methods for interface are already written
  - Avoid bugs on copy paste
- Speed
  - Development is faster

?

Zlatko Bratkovic

github.com/oktalz
gophers.slack.com -> #croatia