Go系统测试覆盖率统计及应用实践

姬长军@七牛云

MTSC2018
第四届中国移动互联网测试开发大会
TesterHome | IT大咖说
TesterHome

目录

背景 - Go语言的先行者

- 使用Go语言做自动化基础支撑的质量保证先行者团队
  - 全员测试开发
  - 完善的测试框架
    - Ginkgo+Gomega高效组织用例及断言
    - Go天然的数据驱动支持
    - 全面的日志追踪
    - 高效的并发

5000+ 自动化用例

- 以质取胜，追求精准系统测试覆盖率

目录

探索 - Go Cover Story

- Go1.2

Instrument the binary
- 技术难度大
- 不可移植性

VS

Rewrite the package's source code before compilation to add instrumentation, compile and run the modified source, and dump the statistics

```
func Size(a int) string {
    GoCover.Count[0] = 1
    switch {
    case a < 0:
        GoCover.Count[2] = 1
        return "negative"
    case a == 0:
        GoCover.Count[3] = 1
        return "zero"
    case a < 10:
        GoCover.Count[4] = 1
        return "small"
    case a < 100:
        GoCover.Count[5] = 1
        return "big"
    case a < 1000:
        GoCover.Count[6] = 1
        return "huge"
    }
    GoCover.Count[1] = 1
    return "enormous"
}
```

```
var GoCover = struct {
    Count     [13]uint32
    Pos       [3 * 13]uint32
    NumStmt   [13]uint16
} {
```

git:(master) ✗ go test -cover

探索 - Go Cover 不足

- Go本身没有提供系统测试覆盖率的支持
- 业界也没有成熟的方案
  - Elastic公司的一篇blog给了我们启发
    - 为每个main方法写个test文件，通过测试入口调用程序入口，借用go test -cover命令的能力来输出系统测试覆盖率

探索 - Elastic的方案

第一步: 创建Main Test File

```
// Test started when the test binary is started.
func TestSystem(t *testing.T) {
  if *systemTest {
     main()
  }
}
```

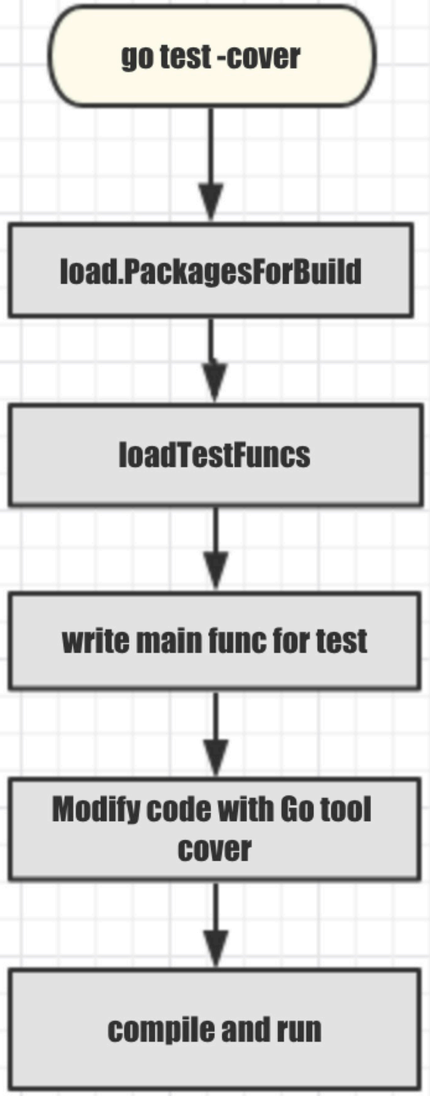
第二步: 修改源码, 将flag
全局定义

```
var configDirPath string
// Init config path flag
func init() {
  flag.StringVar(&configDirPath, "configDir", "", "pa
configuration directory with .yml files")
}
```


第三步: 执行 go test -c 编译出二进制, 在系统测试环境执行此二进制, 输出覆盖率结果

结论: 可行，但不通用

探索 - Go test with cover



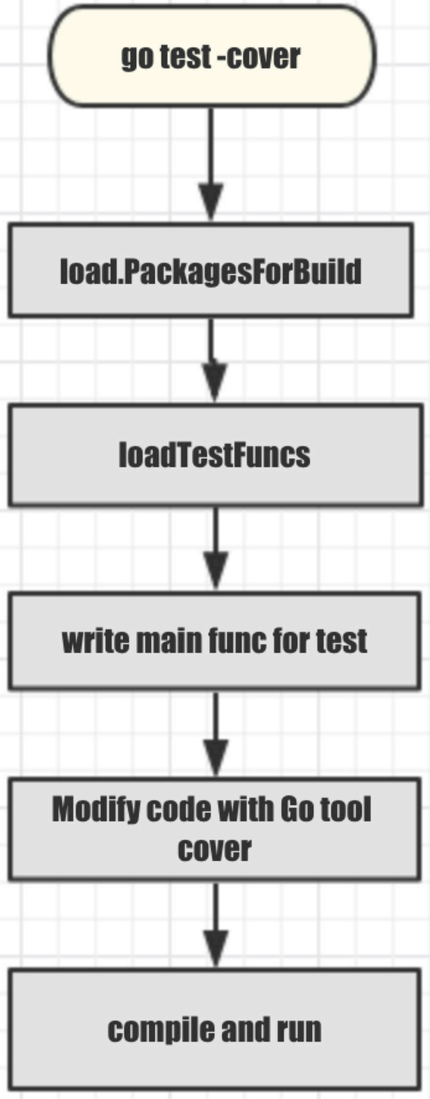
test main template:

```
func main() {
{{if .CoverEnabled}}
    testing.RegisterCover(testing.Cover{
        Mode: {{printf "%q" .CoverMode}},
        Counters: coverCounters,
        Blocks: coverBlocks,
        CoveredPackages: {{printf "%q" .Covered}},
    })
{{end}}
    m := testing.MainStart(testdeps.TestDeps{}, tests, benchmarks, examples)
{{with .TestMain}}
    {{.Package}}.{{.Name}}(m)
{{else}}
    os.Exit(m.Run())
{{end}}
}
```

go1.4引入:

```
func TestMain(m *testing.M)
```

探索 - Go test with cover




```go
// Run runs the tests. It returns an exit code to pass to os.Exit.
func (m *M) Run() int {
    // Count the number of calls to m.Run.
    // We only ever expected 1, but we didn't enforce that,
    // and now there are tests in the wild that call m.Run multiple times
    // Sigh. golang.org/issue/23129.
    m.numRun++

    // TestMain may have already called flag.Parse.
    if !flag.Parsed() {
        flag.Parse()
    }

    if *parallel < 1 {
        fmt.Fprintln(os.Stderr, a: "testing: -parallel can only be given
        flag.Usage()
        return 2
    }

    if len(*matchList) != 0 {
        listTests(m.deps.MatchString, m.tests, m.benchmarks, m.examples)
        return 0
    }

    parseCpuList()

    m.before()
    defer m.after()
    m.startAlarm()
    haveExamples = len(m.examples) > 0
    testRan, testOk := runTests(m.deps.MatchString, m.tests)
    exampleRan, exampleOk := runExamples(m.deps.MatchString, m.examples)
    m.stopAlarm()
    if !testRan && !exampleRan && *matchBenchmarks == "" {
        fmt.Fprintln(os.Stderr, a: "testing: warning: no tests to run")
```

探索 - 方案

为被测程序自动化的创建TestMain(m *testing.M)文件, 然后同样借助go test -c -cover命令, 将被测程序插桩, 之后部署到系统测试环境。之后执行测试用例, 就可以精确得到系统测试覆盖率结果

实现 - 插桩工具

- 自动生成TestMain文件
- 上报自身地址到控制层
- 基于业务需求插桩源码包
- 编译服务

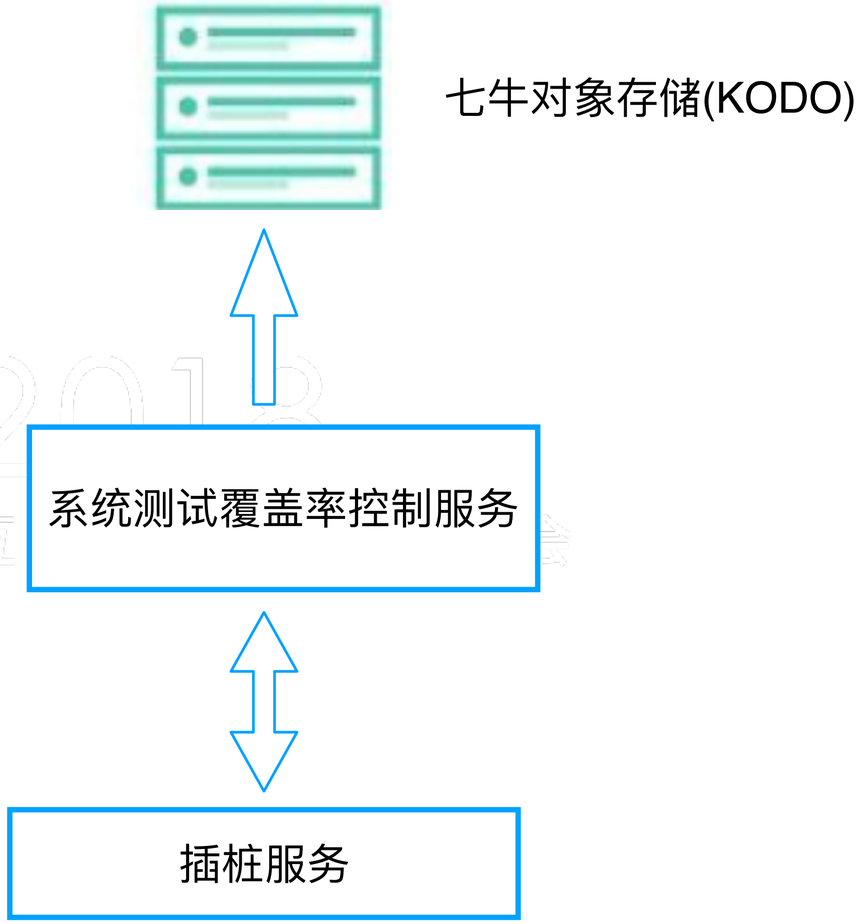
```
func TestMain(m *testing.M) {
        go func() {
            main()
        }()

        time.Sleep(time.Second)
        if !flag.Parsed() {
            flag.Parse()
        }
        stop := make(chan bool)
        go func() {
            registerKiller(stop)
        }()

        <-stop
        os.Exit(m.Run())
    }
}
```

实现 - 控制层

- 覆盖率结果的收集与上报
  - 定义团队与服务之间的关系
  - 接受被测服务的注册
  - 停止被测服务

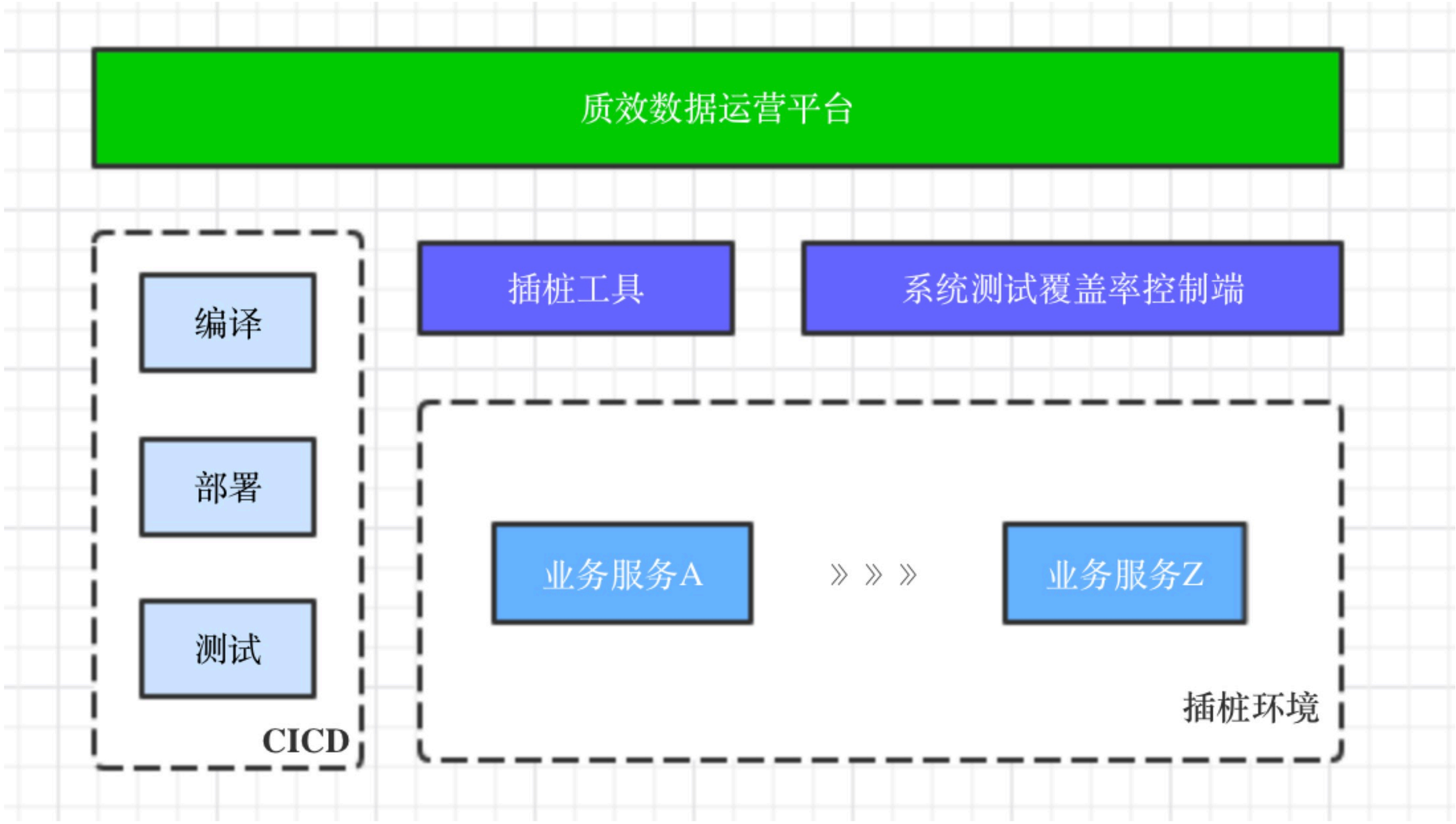


实现 - 产品化(质效平台)

- 覆盖率结果汇总与计算
  - 合并同个服务的多个实例
  - 基于团队, 合并多个服务
- 基于各种维度展示覆盖率结果及趋势
  - 团队
  - 服务
  - 包与文件

服务名	覆盖率 (%)	
pili-forwardg-stcovstest	56.77%	197 / 347
pili-streamd-stcovstest	48.01%	5032 / 10482
pili-zeusd-stcovstest	38.02%	4689 / 12333

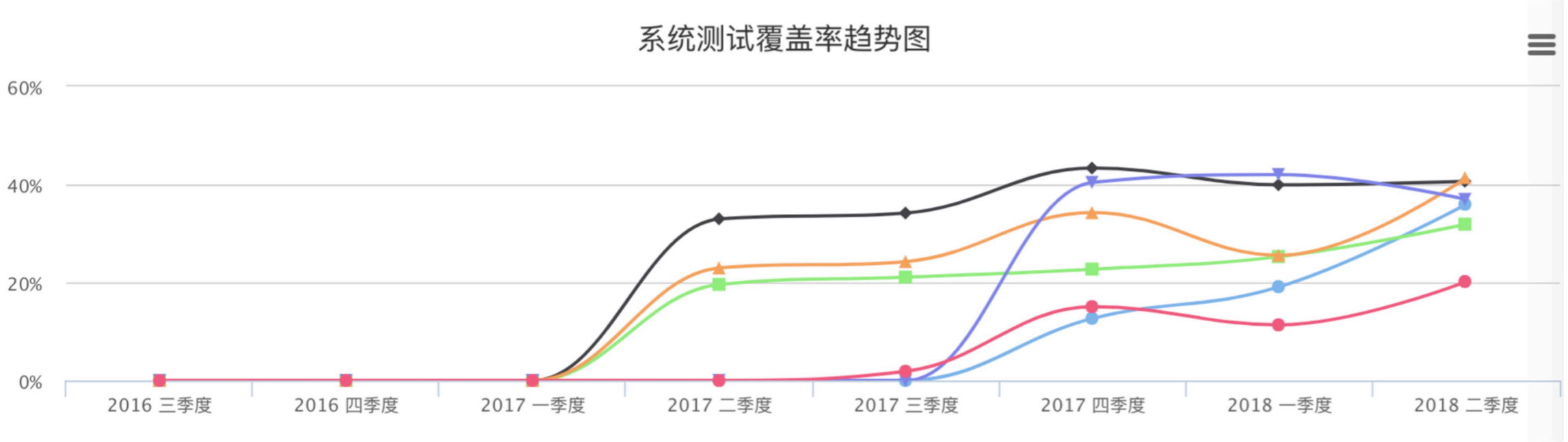
实现 - 整体架构



目录

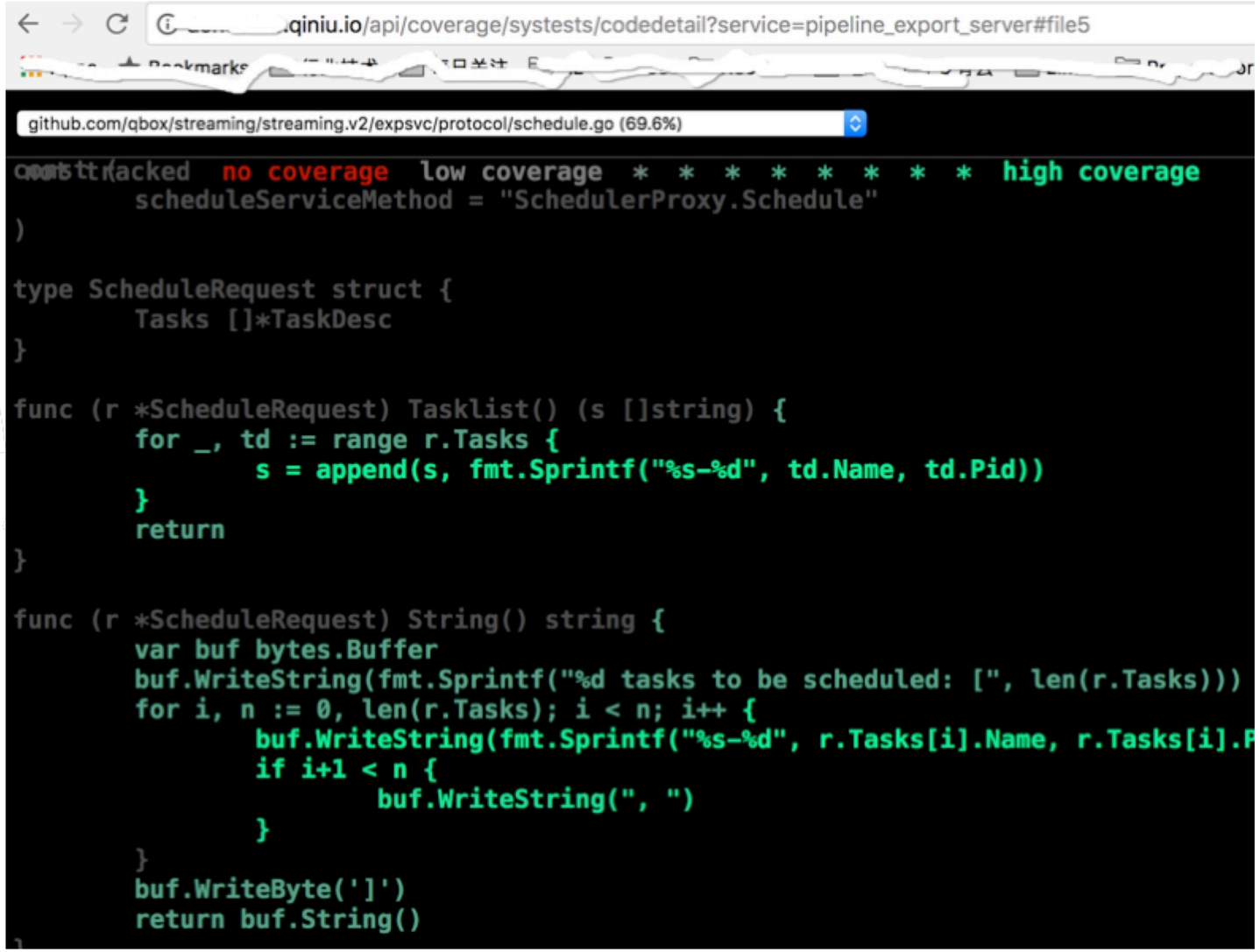
实践场景 - 量化

- 指标量化, 可衡量
- 目标清晰, 过程透明
- 质量运营, 激励正向竞争



实践场景 - 精准测试

- 精准测试辅助日常迭代



目录

后续计划

- 嵌入到内部容器化CICD项目(SPOCK), 优化部署及环境冲突隐忧

谢谢大家

MTSC2018

第四届中国移动互联网测试开发大会

TesterHome | IT大咖说

TesterHome