Golang Tutorial #3

- unit test
- benchmark
- debug on vscode

unit test tool

- assert
- dockertest
- gock

unit test flow control

```
// init_test.go
func TestMain(m *testing.M) {
        log.SetOutput(os.Stdout)
        log.SetFlags(log.LstdFlags)
        var p *int
        retCode := 0
        p = &retCode
        BeforeTest()
        defer AfterTest(p)
        *p = m.Run()
}
```

Assert

```
func TestGetMongoDBInfo(t *testing.T) {
        mongoConfig := getMongoDBInfo()
        assert.Equal(t, "testt", mongoConfig.Name)
}
```

```
--- FAIL: TestGetMongoDBInfo (0.00s)
    .../main_test.go:54:
                Error Trace:    main_test.go:54
                Error:          Not equal:
                                expected: "testt"
                                actual  : "test"
                Test:           TestGetMongoDBInfo
FAIL
```

HTTP mock

```
defer gock.Off() // Flush pending mocks after test execution
gock.InterceptClient(httpClient)
defer gock.RestoreClient(httpClient)
apDomain := "http://test.com"
path := "/test"
gock.New(apDomain).
    Get(path).
    Reply(200).
    JSON(map[string]string{
        "id": "123",
    })
```

static function mock

```
var (
        getProvisionFunc = provision.GetProvision
)

// test
        getProvisionFunc = func(ctx goctx.Context, service string, rp *redispool.Pool) (bool, gopkg.CodeError) {
                if service == "bad" {
                        return false, nil
                }
                return true, nil
        }
```

dockertest run mongo

```
var (
        dockerPool     *dockertest.Pool
        dockerResource *dockertest.Resource
)

dockerPool, err = dockertest.NewPool("")
dockerResource, err = dockerPool.Run("mongo", "3.4", nil)
dockerResource.GetPort("27017/tcp")
```

dockertest teardown

```
func AfterTest(ret *int) {
        if e := recover(); e != nil {
                dockerPool.Purge(dockerResource)
                os.Exit(1)
        }
        dockerPool.Purge(dockerResource)
        os.Exit(*ret)
}
```

- sometimes teardown fail, please use `docker system prune -a`

go benchmark #1

- `go test -benchmem -run=xxx` (test cpu time and memory alloc)
- used when compared two or more syntax/function

```
func BenchmarkIfLt1(b *testing.B) {
        count := 0
        test := ""
        for n := 0; n < b.N; n++ {
                if len(test) < 1 {
                        count++
                }
        }
        fmt.Println("lt1:", count)
}
```

go benchmark result

```
BenchmarkIfLt1-4        lt1: 100
lt1: 10000
lt1: 1000000
lt1: 100000000
lt1: 2000000000
2000000000               0.64 ns/op            0 B/op          0 allocs/op
}
```

go debug on vscode

- update go tools of vscode
- `go get -u github.com/go-delve/delve/cmd/dlv` make sure delve version support go version
- lazy: open main.go and launch debugger
- check your debugger launch point must in `main` package
- all check point could be debugged

go profiling

- https://github.com/davecheney/gophercon2018-performance-tuning-workshop
- `go tool pprof`
- http pprof
- `go tool pprof -http=":8011"`
- `http://localhost:10201/debug/pprof/profile?seconds=30`

HTTP pprof example

```
import (
        "log"
        "net/http"
        _ "net/http/pprof"
)

// ActivateProfile runs the profiling endpoint
func ActivateProfile() {
        log.Println("Start profiling")
        go http.ListenAndServe(":10201", http.DefaultServeMux)
}
```