

# Golang Tutorial #3

- unit test
- benchmark
- go-grpc
- context

# 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",
    })
```

# 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`
- 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 #2

```
func BenchmarkIfEq0(b *testing.B) {  
    count := 0  
    test := ""  
    for n := 0; n < b.N; n++ {  
        if len(test) == 0 {  
            count++  
        }  
    }  
    fmt.Println("Eq0:", count)  
}
```

# go gRPC implementation

- [grpc intro](#)
- [go-grpc](#)
- `protoc --go_out=plugins=grpc:. *.proto`
- [go grpc example](#)
- server and client struct implement interface
- `RegisterXXXServiceServer` `NewXXXServiceClient`

# go context

- built-in library [context](#)

```
type cancelCtx struct {  
    Context  
  
    mu      sync.Mutex           // protects following  
    done    chan struct{}          // created lazily, c  
    children map[canceler]struct{} // set to nil by the  
    err      error                // set to non-nil by
```

# go context cancel

```
ctx, cancel := context.WithCancel(context.Background())  
  
// after complete somethings...  
cancel()
```

# go context deadline/timeout

```
ctx, cancel := context.WithDeadline(context.Background(), d)

// Even though ctx will be expired, it is good practice to
// cancelation function in any case. Failure to do so may
// context and its parent alive longer than necessary.
defer cancel()

select {
case <-time.After(1 * time.Second):
    fmt.Println("overslept")
case <-ctx.Done():
    fmt.Println(ctx.Err())
}
```