OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Продвинутое тестирование в Go

Романовский Алексей

Преподаватель OTUS

Слак Otus: ищите по имени
Телеграм: @alexus1024
GitHub: https://github.com/alexus1024

Преподаватель

Романовский Алексей

Сейчас - разработчик Resolver Inc (go, Node)

Более 5 последних лет - бекенд на Go
Ранее, 10 лет - фуллстек - MS .Net, C#

Часовой пояс: -4 GMT

Правила вебинара

Активно участвуем

Off-topic обсуждаем в канале в Телеграме

Задаем вопросы в чат

Вопросы вижу в чате, могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или задайте вопрос

Цели вебинара

После занятия вы сможете

1. Работать с моками
2. Узнать как использовать ряд продвинутых приёмов тестирования

Смысл

Зачем вам это уметь

1. Лучше обеспечивать качество кода в сложных проектах

Приёмы в тестировании

1. Blackbox

```go
package router

func New(...) *Router {
    return &Router{
        // ...
    }
}
func (r *Router) run() { // приватный
метод будет недоступен
    // ...
}
```

```go
package router_test

import (
    // тестируемый пакет импортируется
    "testing"
    "github.com/me/myprj/internal/router"
)

func TestRouter(t *testing.T) {
    r := router.New(...)
    // ....
}
```

3. Suite

https://pkg.go.dev/github.com/stretchr/testify/suite

```go
type UsersTestSuite struct {
   suite.Suite
   db UsersDB
}

func (s *UsersTestSuite) SetupTest() {
   s.db = NewUsersDB(connstr)
}
func (s *UsersTestSuite) TestAddUser() {
   user1 := fakeUser()
   s.db.AddUser(user1)
   u, err := s.db.FindUser(user1.id)
   s.Require().NoError(err)
   s.Require().Equal(user1, u)
}
func TestUsersTestSuite(t *testing.T) {
   suite.Run(t, new(UsersTestSuite))
}
```

фичи:

- запускает код перед\после каждого теста без копипасты
- паника в тесте не отменит следующий тест (в отличие от цепочки t.Run(...))
- статические данные хранятся в структуре, а не как глобальные переменные - не видны из соседних тестовых файлов

2. Dependency injection (DI)

```
type UserStore struct { // будем тестировать эту структуру
    db UsersDB
}
func (s *UserStore) GetUserName(userID string) (string, error) { /* … */ }
func NewUserStore(db UsersDB) *UserStore { /* … */ }

func main() {
    repo := NewUserStore(NewDB(connstr))
    newID, err := store.GetUserName(user1.ID)
}
```

```
func TestProcessAndStore() { // тест
   // ...
   store := NewUserStore(mockDB) // подставляем другую имплементацию UsersDB
   // ...
   newID, err := store.GetUserName(user1.ID)
}
```

- Не создаём зависимости внутри
- Получаем зависимости снаружи
- Делаем зависимости интерфейсами

Моки
Mocks

Отличие моков от стабов

- Стаб — это заглушка, "простая" реализация интерфейса
  - Может хранить состояние
- Мок фиксирует вызовы интерфейса.
  - Позволяет проверить правильность его использования.

https://martinfowler.com/articles/mocksArentStubs.html

Пакеты для mock-ов

- https://pkg.go.dev/github.com/stretchr/testify/mock
- https://github.com/golang/mock

Моки: код, который тестируем

```
type MyRepo struct {
  db *dbdriver.Connection
}

func (o *MyRepo) SavePersonDetails(firstname, lastname string, age int) (int, error) {
  return db.Exec("SOME SQL HERE", firstname,lastname,age)
}
```

```
type Repo interface{
  SavePersonDetails(firstname, lastname string, age int) (int, error)
}

func BusinessLogicFunc(repo Repo){ // <- we are testing this one
  f,l,a := parsePerson()
  return repo.SavePersonDetails(f,l,a)
}
```

Моки: код тестов

```
type MyMock struct {
  mock.Mock
}
func (o *MyMock) SavePersonDetails(firstname, lastname string, age int) (int, error) {
  args := o.Called("SavePersonDetails",firstname, lastname, age)
  return args.Int(0), args.Error(1)
}
```

```
func TestWithMock(t *testing.T) {
  m := &MyMock{}
  m.On("SavePersonDetails", "John", "Doe", 30).Return(1, nil)

   // test
  affected, err := BusinessLogicFunc(m)
  require.NoError(t, err)

  m.AssertExpectations(t)
}
```

Практика!
Отладьте тест:
https://go.dev/play/p/tlAvE4AMDuV

Перед выполнением, выберите что будете менять:

- **либо только тест**
- **либо только код приложения**

Mockery

https://github.com/vektra/mockery

- можно в докере

  `docker run -v "$PWD":/src -w /src vektra/mockery -all`
- можно ставить себе
  - `go install github.com/vektra/mockery/v2@v2.36.0`
  - (mac) `brew install mockery` (нет выбора версии)

- есть проблема с vendor
- долго, если для вообще всего
- --with-expecter, constructor(auto cleanup)

```
rm -rf internal/mocks
mockery --all --case underscore --keeptree --dir internal/api --output internal/mocks/api --log-level warn
mockery --all --case underscore --keeptree --dir internal/logic --output internal/mocks/logic --log-level warn
```

Практика!

Давайте рефакторить.

Моки для времени

- https://github.com/cabify/timex
- https://github.com/benbjohnson/clock

```
type Application struct {
  Clock clock.Clock
}
func StartApp() &Application{
  var app := &Application{}
  app.Clock = clock.New()
  return app
}

func TestApplication_DoSomething(t *testing.T) {
  mock := clock.NewMock()
  app := Application{Clock: mock}
  // ...
  mock.Add(2 * time.Hour)
}
```

Стаб для ФС

- https://github.com/spf13/afero

```
var AppFs = afero.NewMemMapFs()

var AppFs = afero.NewOsFs()
```

Генерация тестовых данных

Faker

https://github.com/bxcodec/faker
https://go.dev/play/p/BqOcrrUCZAn

```
type Person struct {
   Name   string `faker:"username"`
   Phone  string `faker:"phone_number"`
   Answer int64  `faker:"answer"`
}

func CustomGenerator() {
   faker.AddProvider("answer", func(v reflect.Value) (interface{}, error) {
       return int64(42), nil
   })
}
func main() {
   CustomGenerator()
   var p Person
   faker.FakeData(&p)
   fmt.Printf("%+v\n", p)
}
```

Faker: seed

https://github.com/bxcodec/faker
https://go.dev/play/p/BqOcrrUCZAn

В тестах важна повторяемость.
Нужно выводить seed в вывод теста.
Это позволит повторить тест с теми же данными

```
var seed int64 = time.Now().UnixNano()
s.T().Logf("rand seed: %d", seed)
rand.Seed(seed)
s.genFakeData()
```

```
--- FAIL: TestStoreSuire (0.00s)
--- FAIL: TestStoreSuire/TestDuplicate (0.00s)
store_test.go:45: rand seed: 1599764658164627786
```

```
var seed int64 = 1599764658164627786 //time.Now().UnixNano()
```

Fuzzing

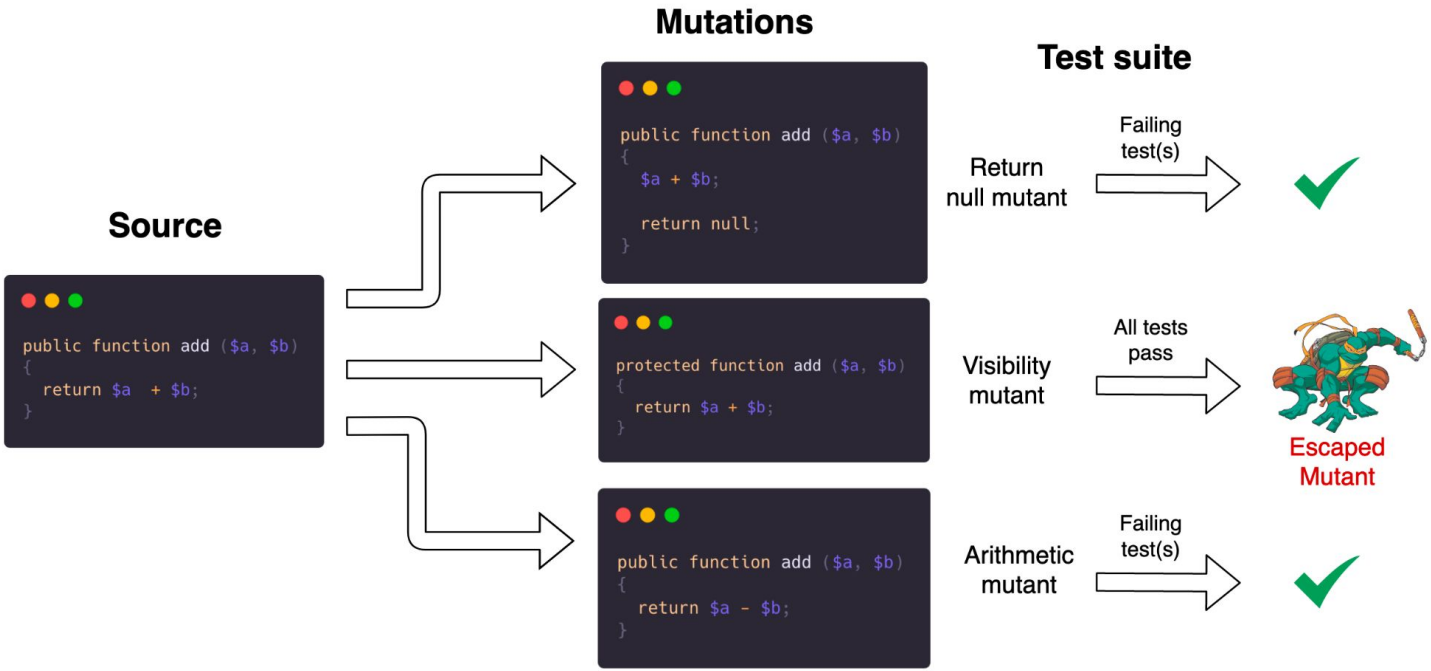
https://go.dev/security/fuzz/
https://towardsdatascience.com/fuzzing-tests-in-go-96eb08b7694d

- постоянно долбит функцию со случайными параметрами
- ведёт запись “интересных” наборов параметров
  - интересные - это те, которые меняют покрытие
- go 1.18

Ещё приёмчики

Тестирование мутированием

(знакомимся с концепцией)



https://github.com/zimmski/go-mutesting

Golden files

```go
var update = flag.Bool("update", false, "update golden files")

func TestSomething(t *testing.T) {
    actual := doSomething()
    golden := filepath.Join("test-fixtures", "expected.golden")
    if *update {
        ioutil.WriteFile(golden, actual, 0644)
    }
    expected, _ := ioutil.ReadFile(golden)
    require.Equal(t, expected, actual)
}
```

```
> go test -update=true
```

Пригодится для тестирования кода, который генерирует файл. Или апи.

Примеры кода

https://github.com/OtusGolang/webinars_practical_part/tree/master/03-unit-testing

Big picture

- Blackbox
- DI
- Suite
- Mocks
  - для времени и ФС
- Мутанты
- Золотишко

Вопросы?

Следующий вебинар

23 августа 2023

Интерфейсы изнутри

- Ссылка на вебинар будет в ЛК за 15 минут
- Материалы к занятию в ЛК — можно изучать
- Обязательный материал обозначен красной лентой

Заполните, пожалуйста, опрос о занятии по ссылке https://otus.ru/polls/116798/