Тестирование Golang приложений

18 ноября 2016

Владимир Ананьев
АО "Программный Регион"

Какое бывает тестирование?

Есть различные классификации тестирования (Wikipedia (https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D

)
В нашем мире речь идет обычно о следующих

По степени автоматизации:

- Ручное тестирование
- Автоматизированное тестирование
- Полуавтоматизированное тестирование

По степени изолированности компонентов:

- Модульное тестирование (отдельные функции и модули)
- Интеграционное тестирование (взаимодействие отдельно взятых сервисов между собой)
- Системное тестирование (имитация работы всей системы)

Демо: иллюстрация структуры системы

Unit-тестирование

- **Модульное тестирование**, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.
- Идея состоит в том, чтобы писать тесты для **каждой нетривиальной функции или метода**. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к **регрессии**, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Unit-тесты в Go

- Все просто - используем пакет testing (https://golang.org/pkg/testing/)
- Тестируется функция на прием различных **входных параметров**
- В идеале должны быть протестированы все **"ветвления"** в функции

```
func Add(a, b int) (int, error) {
    if a < 0 || b < 0 {
        return 0, errors.New("Values are negative")
    }
    return a + b, nil
}
func TestAdd(t *testing.T) { // Эта функция в логике программы не участвует
    a, b := 1, 2
    sum, err := Add(a, b)
    if err != nil {
        t.Fatal(err)
    }
    if sum != 3 {
        t.Fatal(err)
    }
}
```

Демо: пример теста с возвращаемой ошибкой

Где писать тесты и как запускать?

- Хорошая практика - в отдельных файлах, именуемых с суффиксом **_test**
- Хорошая практика - в отдельном **package**

```
user.go
user_test.go

tests/
  |-- user.go
```

- Запускается с помощью встроенной утилиты **test**

```
go test               // Запустятся все тесты в текущем каталоге
go test -v            // Будет вывод дополнительной информации
go test -v -cover     // Будет выведен процент покрытия кода
go test -run ABC      // Будет выполнено тестирование только функции TestABC()
go test -run "AB[CD]" // Регулярное выражение, будут выполнены функции TestABC() и TestABD()
```

- Есть еще много дополнительных параметров для **go test**

```
go test --help
```

Что увидим после выполнени команды go test?

```
go test -v -cover
...
=== RUN   TestGetAction
--- PASS: TestGetAction (0.00s)
=== RUN   TestGetActionInvalidUrlPars
--- PASS: TestGetActionInvalidUrlPars (0.00s)
=== RUN   TestGetActionInvalidActionId
--- PASS: TestGetActionInvalidActionId (0.00s)
=== RUN   TestGetActionInvalidContentId
--- PASS: TestGetActionInvalidContentId (0.00s)
=== RUN   TestGetFullActionInvalidPath
--- PASS: TestGetFullActionInvalidPath (0.00s)
=== RUN   TestGetFullActionInvalidHex
--- PASS: TestGetFullActionInvalidHex (0.00s)
--- PASS: TestGetFullActionInvalidMsg (0.00s)
=== RUN   TestGetFullActionInvalidActionId
--- PASS: TestGetFullActionInvalidActionId (0.00s)
=== RUN   TestSerializeDeserialize
--- PASS: TestSerializeDeserialize (0.00s)
```

- **Важно:** все тестовые функции выполняются параллельно, т.е. **не нужно** завязывать логику одной на логику другой

TestMain функция

- Инициализация объектов, необходимых в разных тестовых функциях

```
// Tests entry point
func TestMain(m *testing.M) {
    ...
    // Connect to mongo storage
    sess, err = GetSession(cfg.MongoDB)
    if err != nil {
        panic(err)
    }
    // Run tests
    os.Exit(m.Run())
}
```

TDD (Test Driven Development)

- Демонстрация презентации **"TDD - Way to the best code"** (Файл **tdd.pdf** в репозитории)

Benchmark (тесты производительности)

- Делается аналогично добавлению функций тестирования
- Добавляются функции **benchmark**
- Основной смысл - запускать несколько раз требуемую функцию и смотреть параметры при ее работе: время работы, выделяемая память.

```
// Benchmarks string concatenation with fmt package
func BenchmarkConcatFmt(b *testing.B) {
    b.ReportAllocs()                                // Показывать выделение памяти
    for i := 0; i < b.N; i++ {                      // go test сам выбирает, какой N выбрать
        key := fmt.Sprintf("%v%v%v%v%v%v%v%v",
            "prefix",
            stats.KeySep,
            rec.Timestamp,
            stats.KeySep,
            rec.DomainGroup,
            stats.KeySep,
            rec.SiteId,
            stats.KeySep)
        fmt.Sprintf("r_%v", key)
    }
}
```

Запуск бенчмарков и просмотр результатов

- Запускается с помощью команды **go test**

```
go test -bench=.                // Точка - регулярное выражение, берутся все функции бенчмарка
go test -bench=. concat_test.go // Из конкретного файла
```

- Вывод команды

```
BenchmarkConcatFmt-8     2000000  983 ns/op  232 B/op  12 allocs/op
BenchmarkConcatBytes-8   2000000  728 ns/op  304 B/op  9 allocs/op
BenchmarkConcatStrings-8 3000000  463 ns/op  240 B/op  7 allocs/op
PASS
ok      command-line-arguments    7.094s
```

Материалы

- golang.org/pkg/testing/ (https://golang.org/pkg/testing/)
- Тестирование ПО

  (https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0
- Testable examples in Go (https://blog.golang.org/examples)
- Tips & Tricks (https://medium.com/@matryer/5-simple-tips-and-tricks-for-writing-unit-tests-in-golang-619653f90742#.q68ol57q2)

Задание

Benchmark тест для разных способов **конкатенации** (сложение строк)

- Есть структура данных **Record**
- В ней есть набор полей **A, B, C, D, E, F**
- Нужно сгенерить строку вида **a_b_c_d_e_f**, где **a,b,c,d,e,f** - значения соответствующих полей
- Необходимо сравнить производительность разных подходов в такой конкатенации:
- С помощью **fmt.Sprintf**
- С помощью **bytes.Buffer**
- С помощью **strings.Join**
- Результаты теста записать в виде комментариев в конец **_test.go** файла

Thank you

18 ноября 2016

Владимир Ананьев
АО "Программный Регион"
vladimir.ananyev@regium.com (mailto:vladimir.ananyev@regium.com)