Advanced Testing with Go

2017-01-25

Povilas Versockas
Software Engineer, Uber

Agenda

- News
- Intro to Go testing framework
- Tools
- Advanced testing tips & tricks

[1]

Based on

- Andrew Gerrand - Testing Techniques (https://talks.golang.org/2014/testing.slide)
- Mitchell Hashimoto - Advanced Testing with Go (https://www.youtube.com/watch?v=yszygk1cpEc)
- Ben Johnson - Structuring Tests in Go (https://medium.com/@benbjohnson/structuring-tests-in-go-46ddee7a25c#.q88391hne)
- Dave Cheney - Writing Table Driven Tests in Go (https://dave.cheney.net/2013/06/09/writing-table-driven-tests-in-go)
- Dave Cheney - Test Fixtures in Go (https://dave.cheney.net/2016/05/10/test-fixtures-in-go)
- Peter Bourgon - Go: Best Practices for Production Environments (https://peter.bourgon.org/go-in-production/) [2]

News

- Go 1.8 RC2 is released!
- Go 1.8 is going to to launched in February 2017.

Features:

- HTTP Server Graceful Shutdown: `Server.Shutdown()` / `Server.Close()`.
- Go now supports a "plugin" build mode for generating plugins written in Go.
- Better concurrent Map misuse detection.
- Garbage collection pauses should be significantly shorter.
- The `GOPATH` environment variable now has a default value if it is unset. It defaults to $HOME/go

[3]

Go Testing Basics

[4]

Testing framework

Go has a built-in testing framework.
It is provided by the `testing` package and the go `test` command, which automates execution of test functions.

```
func TestXxx(*testing.T)
```

To write a new test suite, create a file whose name ends **_test.go** that contains the `TestXxx` functions

```
github.com/user/
  http/
      http.go              # package source
      http_test.go         # test source
```

[5]

*testing.T

T is a type passed to Test functions to manage test state and support formatted test logs.

```
t.Errorf("got bar = %v, want %v", got, want)
t.Fatalf("Frobnicate(%v) returned error: %v", arg, err)
t.Logf("iteration %v", i)
```

- **Errorf** logs and marks function as having failed but continues execution.
- **FatalF** logs and marks function as having failed and stops its execution.

Parallel tests:

```
t.Parallel()
```

Skipping tests:

```
t.Skip()
```

Testing Main

Sometimes necessary for a test program to do extra setup or teardown before or after testing.

```
func TestMain(m *testing.M) {
  // setup
  m.Run()
  // teardown
}
```

[7]

Testing framework

Test files that declare a package with the suffix **_test** will be compiled as a separate package, and then linked and run with the main test binary.

```
package http_test
```

The go tool will ignore a directory starting with **dot**, **underscore** or named **testdata**, making it available to hold ancillary data needed by the tests. [8]

Table driven tests

Go's struct literal syntax makes it easy to write table-driven tests:

```
func TestIndex(t *testing.T) {
    var tests = []struct {
        s   string
        sep string
        out int
    }{
        {"", "", 0},
        {"", "a", -1},
        {"fo", "foo", -1},
        {"foo", "foo", 0},
        {"oofofoofooo", "f", 2},
        // etc
    }
    for _, test := range tests {
        actual := strings.Index(test.s, test.sep)
        if actual != test.out {
            t.Errorf("Index(%q,%q) = %v; want %v", test.s, test.sep, actual, test.out)
        }
    }
}
```

[9]

Table driven tests

- Low overhead to add new test cases.
- Makes testing exhaustive scenarios simple.
- Follow the pattern even for single cases, if its possible to grow.
- Do this pattern *a lot*.

Examples:

https://golang.org/src/time/time_test.go (https://golang.org/src/time/time_test.go)

https://golang.org/src/math/all_test.go (https://golang.org/src/math/all_test.go)

Tools

[11]

Generate tables

https://github.com/cweill/gotests (https://github.com/cweill/gotests)

```
$ gotests [options] PATH ...
```

Plugins also exist for Sublime Text 3, Emacs, Vim, Atom Editor, Visual studio code. [12]

Generate tables

render.go

```
58      for _, name := range bindata.AssetNames() {
59          tmpls = template.Must(tmpls.Parse(string(bindata.MustAsset(name)))
                ))
60      }
61  }
62
63  func fieldName(f *models.Field) string {
64      var n string
65      if f.IsNamed() {
66          n = f.Name
67      } else {
68          n = f.Type.String()
69      }
70      return "r" + n
71  }
72
73  func receiverName(f *models.Receiver) string {
74      var n string
75      if f.IsNamed() {
76          n = f.Name
77      } else {
78          n = f.ShortName()
79      }
80      if reserved[n] {
81          return "r" + n
82      }
83      return n
84  }
85
86  func parameterName(f *models.Field) string {
87      var n string
88      if f.IsNamed() {
89          n = f.Name
90      } else {
91          n = fmt.Sprintf("in%v", f.Index)
92      }
93      if reserved[n] {
94          return "p" + n
95      }
96      return n
97  }
98
99  func wantName(f *models.Field) string {
100     var n string
101     if f.IsNamed() {
102         n = "want" + strings.Title(f.Name)
103     } else if f.Index == 0 {
```

render_test.go

```
20      for _, tt := range tests {
21          if got := unexport(tt.s); got != tt.want {
22              t.Errorf("unexport(%v) = %v, want %v", tt.s, got, tt.want
                    )
23          }
24      }
25  }
26
27  func TestFieldName(t *testing.T) {
28      tests := []struct {
29          // Test description.
30          name string
31          // Parameters.
32          f *models.Field
33          // Expected results.
34          want string
35      }{
36      // TODO: Add test cases.
37      }
38      for _, tt := range tests {
39          if got := fieldName(tt.f); got != tt.want {
40              t.Errorf("%q. fieldName() = %v, want %v", tt.name, got,
                    tt.want)
41          }
42      }
43  }
44
45  func TestReceiverName(t *testing.T) {
46      tests := []struct {
47          // Test description.
48          name string
49          // Parameters.
50          f *models.Receiver
51          // Expected results.
52          want string
53      }{
54      // TODO: Add test cases.
55      }
56      for _, tt := range tests {
57          if got := receiverName(tt.f); got != tt.want {
58              t.Errorf("%q. receiverName() = %v, want %v", tt.name, got
                    , tt.want)
59          }
60      }
61  }
62
```

Test runner

http://goconvey.co/ (http://goconvey.co/)

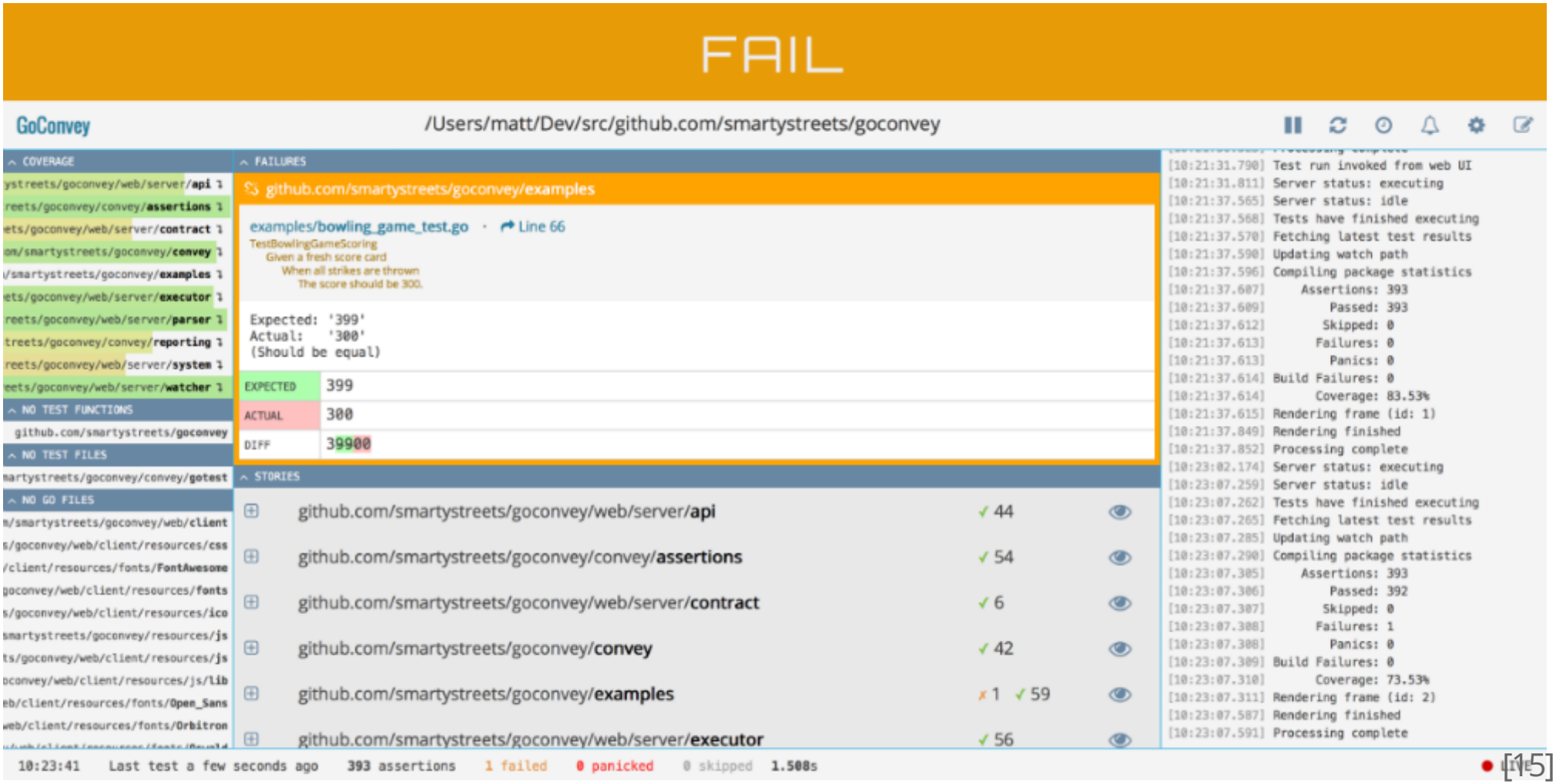
- Tests run automatically
- Web UI/terminal
- Coverage report
- Notifications

```
$ goconvey
```

- I usually run it like this (default port is 8080):

```
$ goconvey -port=8081
```

GoConvey


FAIL
GoConvey
/Users/matt/Dev/src/github.com/smartystreets/goconvey
COVERAGE
NO TEST FUNCTIONS
NO TEST FILES
NO GO FILES
FAILURES
github.com/smartystreets/goconvey/examples
examples/bowling_game_test.go · Line 66
TestBowlingGameScoring
Given a fresh score card
When all strikes are thrown
The score should be 300.
Expected: '399'
Actual: '300'
(Should be equal)
EXPECTED 399
ACTUAL 300
DIFF 39900
STORIES
github.com/smartystreets/goconvey/web/server/api ✓ 44
github.com/smartystreets/goconvey/convey/assertions ✓ 54
github.com/smartystreets/goconvey/web/server/contract ✓ 6
github.com/smartystreets/goconvey/convey ✓ 42
github.com/smartystreets/goconvey/examples ✗ 1 ✓ 59
github.com/smartystreets/goconvey/web/server/executor ✓ 56
[10:21:31.790] Test run invoked from web UI
[10:21:31.811] Server status: executing
[10:21:37.565] Server status: idle
[10:21:37.568] Tests have finished executing
[10:21:37.570] Fetching latest test results
[10:21:37.590] Updating watch path
[10:21:37.596] Compiling package statistics
[10:21:37.607] Assertions: 393
[10:21:37.609] Passed: 393
[10:21:37.612] Skipped: 0
[10:21:37.613] Failures: 0
[10:21:37.613] Panics: 0
[10:21:37.614] Build Failures: 0
[10:21:37.614] Coverage: 83.53%
[10:21:37.615] Rendering frame (id: 1)
[10:21:37.849] Rendering finished
[10:21:37.852] Processing complete
[10:23:02.174] Server status: executing
[10:23:07.259] Server status: idle
[10:23:07.262] Tests have finished executing
[10:23:07.265] Fetching latest test results
[10:23:07.285] Updating watch path
[10:23:07.290] Compiling package statistics
[10:23:07.305] Assertions: 393
[10:23:07.306] Passed: 392
[10:23:07.307] Skipped: 0
[10:23:07.308] Failures: 1
[10:23:07.308] Panics: 0
[10:23:07.309] Build Failures: 0
[10:23:07.310] Coverage: 73.53%
[10:23:07.311] Rendering frame (id: 2)
[10:23:07.587] Rendering finished
[10:23:07.591] Processing complete
10:23:41 Last test a few seconds ago 393 assertions 1 failed 0 panicked 0 skipped 1.508s

Advanced testing tips

Tip 1. Don't use frameworks

Ben Johnson's tip.

- Go proverb "A little copying is better than a little dependency".
- Ben Johnson has published these functions (MIT license):

```
func assert(tb testing.TB, condition bool, msg string)
func ok(tb testing.TB, err error)
func equals(tb testing.TB, exp, act interface{})
```

- go `test` is an incredible workflow tool
- Write a custom framework within go `test`, rather than a separate test harness.

https://github.com/benbjohnson/testing (https://github.com/benbjohnson/testing) [17]

Tip 2. Use the "underscore test" package

Ben Johnson's tip.

- Using a separate `myapp_test` package means that you cannot access the unexported fields and functions in `myapp` package.
- This makes you check if exported API is usable and complete.

[18]

Tip 3. Avoid global state

Mitchell Hashimoto's tip.

- Instead of global state, try to make whatever is global a configuration option using global state as the default, allowing tests to modify it.

```
// Not good on its own
const port = 1000

// Better
var port = 1000

// Best
const defaultPort = 1000

type ServerOpts {
    Port int // default it to defaultPort somewhere
}
```

[19]

Advanced testing tricks

Trick 1. Test fixtures

Dave Cheney/Mitchell Hashimoto's trick.

```
func TestSomeFixture(t *testing.T) {
    data := filepath.Join("testdata", "somefixture.json")
    // ... Do something with data
}
```

- go `test` sets pwd as package directory
- Use relative path "testdata" directory as a place to store data
- Very useful for loading config, model data, binary data, etc.

Trick 2. Golden files

Mitchell Hashimoto's trick.

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

func TestSomething(t *testing.T) {
    // ... table

    for _, tc := range cases {
        actual := doSomething(tc)
        golden := filepath.Join("testdata", tc.Name+".golden")
        if *update {
          ioutil.WriteFile(golden, actual, 0644)
        }
        expected, _ := ioutil.ReadFile(golden)
        if !bytes.Equal(actual, expected) {
            // FAIL!
        }
    }
}
```

Trick 2. Golden files

```
$ go test
...



$ go test -update
...
```

- Test complex output without manually hardcoding it
- Human eyeball the generated golden data. If it is correct, commit it.
- Very scalable way to test complex structures (write a String() method)

Trick 3. Test Helpers

Mitchell Hashimoto's trick.

```
func testTempFile(t *testing.T) string {
    tf, err := ioutil.TempFile("", "test")
    if err != nil {
        t.Fatalf("err: %s", err)
    }
    tf.Close()

    return tf.Name()
}
```

- Never return errors. Pass in *testing.T and fail.
- By not returning errors, usage is much prettier since error checking is gone.
- Used to make tests clear on what they're testing vs what is boilerplate

Trick 3. Test Helpers

```
func testChdir(t *testing.T, dir string) func() {
    old, err := os.Getwd()
    if err != nil {
        t.Fatalf("err: %s", err)
    }

    if err := os.Chdir(dir); err != nil {
        t.Fatalf("err: %s", err)
    }

    return func() { os.Chdir(old) }
}

func TestThing(t *testing.T) {
    defer testChdir(t, "/other")()
    // ...
}
```

Trick 4. Subprocessing: Real

Mitchell Hashimoto's trick.

- Actually executing the subprocess is nice.
- Guard the test for the existence of the binary.
- Make sure side effects don't affect any other test.

```
var testHasGit bool

func init() {
    if _, err := exec.LookPath("git"); err == nil {
        testHasGit = true
    }
}

func TestGitGetter(t *testing.T) {
    if !testHasGit {
        t.Log("git not found, skipping")
        t.Skip()
    }
    // ...
}
```

Trick 5. Subprocessing: Mock

Andrew Gerrand's / Mitchell Hashimoto's trick.

```
func Crasher() {
    fmt.Println("Going down in flames!")
    os.Exit(1)
}
```

Invoke the test binary itself as a subprocess:

```
func TestCrasher(t *testing.T) {
    if os.Getenv("BE_CRASHER") == "1" {
        Crasher()
        return
    }
    cmd := exec.Command(os.Args[0], "-test.run=TestCrasher")
    cmd.Env = append(os.Environ(), "BE_CRASHER=1")
    err := cmd.Run()
    if e, ok := err.(*exec.ExitError); ok && !e.Success() {
        return
    }
    t.Fatalf("process ran with err %v, want exit status 1", err)
}
```

Trick 6. Put mocks, helpers into testing.go files

Mitchell Hashimoto's trick.

- Newer HashiCorp projects have adopted the practice of making a "testing.go" or "testing_*.go" files.
- These are exported APIs for the sole purpose of providing mocks, test harnesses, helpers, etc.
- Allows other packages to test using our package without reinventing the components needed to meaningful use our package in a test.

[28]

Trick 7. Take care of slow running tests

Peter Bourgon trick.

- Add build tag to the `*_integration_test.go` file

```
// +build integration
```

- Run these tests manually with go `test -tags=integration`
- I use this:

```
alias gtest="go test \$(go list ./... | grep -v /vendor/) -tags=integration"
```

- Usage:

```
$ gtest
...
$ gtest -v
...
```

Thank you

2017-01-25

Povilas Versockas
Software Engineer, Uber

@PofkeVe (http://twitter.com/PofkeVe)