

Testing

Tuesday, January 17, 2023 9:33 PM

```
Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go test -cover .
ok      PrimeApp      0.032s  coverage: 54.5% of statements

Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go test -coverprofile=coverage.out
PASS
coverage: 54.5% of statements
ok      PrimeApp      0.037s

Akash@LAPTOP-0S70H5K5 MINGW64 /d/Personal/Programming/Go/Introduction to Testing in Go (Golang)/Testing Practise/PrimeApp
$ go tool cover -html=coverage.out
```

```
PrimeApp/main.go (54.5%) ▾ not tracked  not covered  covered

package main

import "fmt"

func main() {
    no := 8
    _, msg := isPrime(no)
    fmt.Printf(msg)
}

func isPrime(n int) (bool, string) {
    if n == 0 || n == 1 {
        return false, fmt.Sprintf("%d is not prime, by definition", n)
    }

    if n < 0 {
        return false, fmt.Sprintf("%d prime no cannot be negative", n)
    }

    for i := 2; i <= n/2; i++ {
        if n%2 == 0 {
            return false, fmt.Sprintf("%d is not prime no as it is divisible by %d", n, i)
        }
    }

    return true, fmt.Sprintf("%d is a prime number", n)
}
```

go test -coverprofile=coverage.out && go tool cover -
html=coverage.out

To run only one test

```
9     "testing"
10    )
11
12    run test | debug test
D 12 func Test_isPrime(t *testing.T) {
13     primeTests := []struct {
14         name      string
15         testNum   int
16         expected  bool
17         msg       string
18     }{
19         {"prime", 7, true, "7 is a prime number!"},
20         {"not prime", 8, false, "8 is not a prime number"}

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

- tcs@mac-studio primeapp % go test -run Test_isPrime
PASS
ok primeapp 0.091s
- tcs@mac-studio primeapp % go test -v -run Test_isPrime
== RUN Test_isPrime
--- PASS: Test_isPrime (0.00s)
PASS
ok primeapp 0.088s
- tcs@mac-studio primeapp %

To run group of test (test suites)

```
11      run test | debug test
> 12 func Test_alpha_isPrime(t *testing.T) {
13     primeTests := []struct {
14         name      string
15         testNum   int
16         expected  bool
17         msg       string
18     }{
19         {"prime", 7, true, "7 is a prime number!"},
20         {"not prime", 8, false, "8 is not a prime number because it is divisible by 2!"},
21         {"zero", 0, false, "0 is not prime, by definition!"},

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

- tcs@mac-studio primeapp % go test -run Test_alpha
PASS
ok primeapp 0.247s
- tcs@mac-studio primeapp % go test -v -run Test_alpha
--- RUN Test_alpha_isPrime ---
--- PASS: Test_alpha_isPrime (0.00s)
--- RUN Test_alpha_prompt ---
--- PASS: Test_alpha_prompt (0.00s)
PASS
ok primeapp 0.086s
- tcs@mac-studio primeapp % █

All the test cases which has naming convention as Test_alpha will run.

Learn by example situation

I hate long examples to explain simple concepts...

- “I need to mock a function”
- “I need to mock a method on a type”
- “I have a large interface, I need to mock a small set of its methods”
- “I need to mock an HTTP call”

```
}

func OpenDB(user, password, addr, db string) (*sql.DB, error) {
    conn := fmt.Sprintf(format: "%s:%s@%s/%s", user, password, addr, db)
    return sql.Open(driverName: "mysql", conn)
}
```

This is the trouble maker

So go to the function definition

```
// function should be called just once. It is rarely necessary to
// close a DB.
func Open(driverName, dataSourceName string) (*DB, error) {
    driversMu.RLock()
    driveri, ok := drivers[driverName]
    driversMu.RUnlock()
    if !ok {
        return nil, fmt.Errorf("sql: unknown driver %q (forgotten import?)", driverName)
    }

    if driverCtx, ok := driveri.(driver.DriverContext); ok {
        connector, err := driverCtx.OpenConnector(dataSourceName)
        if err != nil {
            return nil, err
        }
        return OpenDB(connector), nil
    }

    return OpenDB(dsnConnector{dsn: dataSourceName, driver: driveri}), nil
}

func (db *DB) pingDC(ctx context.Context, dc *driverConn, release func(error)) error {
    var err error
    if pinger, ok := dc.ci.(driver.Pinger); ok {
```

```
6
7
8 type sqlOpener func(string, string) (*sql.DB, error)
9
10 func OpenDB(user, password, addr, db string, opener sqlOpener) (*sql.DB, error) {
11     conn := fmt.Sprintf(format: "%s:%s@%s/%s", user, password, addr, db)
12     return opener("mysql", conn)
13 }
14
```

```
9
10 func TestOpenDB(t *testing.T) {
11     mockError := errors.New(text: "uh oh")
12     subtests := []struct {
13         name      string
14         u, p, a, db string
15         sqlOpener func(string, string) (*sql.DB, error)
16         expectedErr error
17     }{
18         {
19             name: "happy-path",
20             u:    "u",
21             p:    "p",
22             a:    "a",
23             db:   "db",
24             sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
25                 if source != "u:p@a/db" {
26                     return db: nil, errors.New(text: "wrong connection string")
27                 }
28                 return db: nil, err: nil
29             },
30         },
31         {
32             name: "error-from-sqlOpener",
33             sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
34                 return db: nil, mockError
35             },
36             expectedErr: mockError,
37         },
38     }
39     for _, subtest := range subtests {
40         t.Run(subtest.name, func(t *testing.T) {
41             _, err := sqlexample.OpenDB(subtest.u, subtest.p, subtest.a, subtest.db, subtest.sqlOpener)
42             if !errors.Is(err, subtest.expectedErr) {
43                 t.Errorf(format: "expected error (%v), got error (%v)", subtest.expectedErr, err)
44             }
45         })
46     }
47 }
```

Higher Order Functions

Good

- Clear and proximal to function under test
- Stateless

Not so good....

- Parameter list can get ugly (using a *type* for the function can help)
- Think of dependency graph

Monkey Patching

```
var SQLOpen = sql.Open

func OpenDB(user, password, addr, db string) (*sql.DB, error) {
    conn := fmt.Sprintf("%s:%s@%s/%s", user, password, addr, db)
    return SQLOpen(driverName: "mysql", conn)
}
```

```
func TestOpenDB(t *testing.T) {
    mockError := errors.New("uh oh")
    subtests := []struct {
        name      string
        u, p, a, db string
        sqlOpener func(string, string) (*sql.DB, error)
        expectedErr error
    }{
        {
            name: "happy-path",
            u:    "u",
            p:    "p",
            a:    "a",
            db:   "db",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                if source != "u:p@a/db" {
                    return db: nil, errors.New("wrong connection string")
                }
                return db: nil, err: nil
            },
        },
        {
            name: "error-from-sqlOpener",
            sqlOpener: func(driver string, source string) (db *sql.DB, err error) {
                return db: nil, mockError
            },
            expectedErr: mockError,
        },
    }
    for _, subtest := range subtests {
        t.Run(subtest.name, func(t *testing.T) {
            sqlexample.SQLOpen = subtest.sqlOpener
            _, err := sqlexample.OpenDB(subtest.u, subtest.p, subtest.a, subtest.db)
            if !errors.Is(err, subtest.expectedErr) {
                t.Errorf("expected error (%v), got error (%v)", subtest.expectedErr, err)
            }
        })
    }
}
```

Monkey Patching

Good

- Don't need to modify function signature

Not so good...

- Allergic to parallelism (stateful)
- Have to make variable public (if testing from _test package)

Mock a method on a type

```
func ReadFileContents(f *os.File, numBytes int) ([]byte, error) {
    defer f.Close()
    data := make([]byte, numBytes)
    _, err := f.Read(data)
    if err != nil {
        return nil, err
    }
    return data, nil
}
```

So here we need to mock a file

We don't want to do the operation on the file

- 1) Checking the file implementation

```
3 func Getpagesize() int { return syscall.Getpagesize() }
4
5 // File represents an open file descriptor.
6 type File struct {
7     *file // os specific
8 }
9
10 // A FileInfo describes a file and is returned by Stat and Lstat.
11 type FileInfo interface {
12     Name() string      // base name of the file
13     Size() int64        // length in bytes for regular files; system-dependent for other
14     Mode() FileMode    // file mode bits
15     ModTime() time.Time // modification time
16     IsDir() bool        // abbreviation for Mode().IsDir()
17     Sys() interface{}   // underlying data source (can return nil)
18 }
19
20 // A FileMode represents a file's mode and permission bits.
21 // The bits have the same definition on all systems, so that
22 // information about files can be moved from one system
23 // to another portably. Not all bits apply to all systems.
```

It is a concrete type we need to use interface instead of this

```
)
```

2)

```
func ReadFileContents(rc io.ReadCloser, numBytes int) ([]byte, error) {
    defer rc.Close()
    data := make([]byte, numBytes)
    _, err := rc.Read(data)
    if err != nil {
        return nil, err
    }
    return data, nil
}
```

```
1 type mockReadCloser struct {
2     expectedData []byte
3     expectedErr  error
4 }
5
6 // allow test table to fill out expectedData and expectedErr field
7 func (mrc *mockReadCloser) Read(p []byte) (n int, err error) {
8     copy(p, mrc.expectedData)
9     return n, mrc.expectedErr
10 }
11
12 // do nothing on close
13 func (mrc *mockReadCloser) Close() error { return nil }
14
15
16 func TestReadContents(t *testing.T) {
17     errRead := errors.New("uh oh")
18     testTable := []struct {
19         name      string
20         readCloser io.ReadCloser
21         numBytes   int
22         expectedData []byte
23         expectedErr error
24     }{
25         {
26             name: "happy path",
27             readCloser: &mockReadCloser{
28                 expectedData: []byte(`hello`),
29                 expectedErr: nil,
30             },
31             numBytes: 5,
32             expectedData: []byte(`hello`),
33             expectedErr: nil,
34         },
35         {
36             name: "error-from-read",
37             readCloser: &mockReadCloser{
38                 expectedData: nil,
39                 expectedErr: errRead,
40             },
41             numBytes: 0,
42             expectedData: nil,
43             expectedErr: errRead,
44         },
45     }
46 }
```

Interface Substitution

- “Accept interfaces, return structs”
- 4)
- We used *io.ReadCloser*, but don’t be afraid to define your own interface

If I have a large Interface and I have to mock a small set of its methods

```
func GetItems(client *dynamodb.DynamoDB) (*dynamodb.GetItemOutput, error) {
    return client.GetItem(&dynamodb.GetItemInput{
        AttributesToGet: []*string{aws.String("v:bar")},
        Key: map[string]*dynamodb.AttributeValue{
            "foo": {
                BOOL: aws.Bool(v: true),
            },
        },
    })
}

func GetItems(client dynamodbiface.DynamoDBAPI) (*dynamodb.GetItemOutput, error) {
    return client.GetItem(&dynamodb.GetItemInput{
        AttributesToGet: []*string{aws.String("v:bar")},
        Key: map[string]*dynamodb.AttributeValue{
            "foo": {
                BOOL: aws.Bool(v: true),
            },
        },
    })
}
```

Dynamodb provides an interface for mocking

```
DescribeLimitsWithContext(aws.Context, *dynamodb.DescribeLimitsInput, ...request.Option) (*dynamodb.DescribeLimitsOutput)
DescribeLimitsRequest(*dynamodb.DescribeLimitsInput) (*request.Request, *dynamodb.DescribeLimitsOutput)

DescribeTable(*dynamodb.DescribeTableInput) (*dynamodb.DescribeTableOutput, error)
DescribeTableWithContext(aws.Context, *dynamodb.DescribeTableInput, ...request.Option) (*dynamodb.DescribeTableOutput)
DescribeTableRequest(*dynamodb.DescribeTableInput) (*request.Request, *dynamodb.DescribeTableOutput)

DescribeTableReplicaAutoScaling(*dynamodb.DescribeTableReplicaAutoScalingInput) (*dynamodb.DescribeTableReplicaAutoScalingOutput)
DescribeTableReplicaAutoScalingWithContext(aws.Context, *dynamodb.DescribeTableReplicaAutoScalingInput, ...request.Option) (*dynamodb.DescribeTableReplicaAutoScalingOutput)
DescribeTableReplicaAutoScalingRequest(*dynamodb.DescribeTableReplicaAutoScalingInput) (*request.Request, *dynamodb.DescribeTableReplicaAutoScalingOutput)

DescribeTimeToLive(*dynamodb.DescribeTimeToLiveInput) (*dynamodb.DescribeTimeToLiveOutput, error)
DescribeTimeToLiveWithContext(aws.Context, *dynamodb.DescribeTimeToLiveInput, ...request.Option) (*dynamodb.DescribeTimeToLiveOutput)
DescribeTimeToLiveRequest(*dynamodb.DescribeTimeToLiveInput) (*request.Request, *dynamodb.DescribeTimeToLiveOutput)

GetItem(*dynamodb.GetItemInput) (*dynamodb.GetItemOutput, error)
GetItemWithContext(aws.Context, *dynamodb.GetItemInput, ...request.Option) (*dynamodb.GetItemOutput, error)
GetItemRequest(*dynamodb.GetItemInput) (*request.Request, *dynamodb.GetItemOutput)

ListBackups(*dynamodb.ListBackupsInput) (*dynamodb.ListBackupsOutput, error)
ListBackupsWithContext(aws.Context, *dynamodb.ListBackupsInput, ...request.Option) (*dynamodb.ListBackupsOutput)
ListBackupsRequest(*dynamodb.ListBackupsInput) (*request.Request, *dynamodb.ListBackupsOutput)

ListContributorInsights(*dynamodb.ListContributorInsightsInput) (*dynamodb.ListContributorInsightsOutput, error)
ListContributorInsightsWithContext(aws.Context, *dynamodb.ListContributorInsightsInput, ...request.Option) (*dynamodb.ListContributorInsightsOutput)
ListContributorInsightsRequest(*dynamodb.ListContributorInsightsInput) (*request.Request, *dynamodb.ListContributorInsightsOutput)
```

```
12 of type mockDynamoDBClient struct {
13     dynamodbiface.DynamoDBAPI
14     output *dynamodb.GetItemOutput
15     err    error
16 }
17
18 of func (m *mockDynamoDBClient) GetItem(_ *dynamodb.GetItemInput) (*dynamodb.GetItemOutput, error) {
19     return m.output, m.err
20 }
21
22
23 G func TestGetItems(t *testing.T) {
24     exampleErr := errors.New("uh oh")
25     exampleResponse := &dynamodb.GetItemOutput{
26         Item: map[string]*dynamodb.AttributeValue{
27             "foo": {
28                 BOOL: aws.Bool(false),
29             },
30         },
31     }
32
33     subtests := []struct {
34         name      string
35         client    *mockDynamoDBClient
36         response  *dynamodb.GetItemOutput
37         err       error
38     }{
39         {
40             name: "happy-path",
41             client: &mockDynamoDBClient{
42                 output: exampleResponse,
43                 err: nil,
44             },
45             response: exampleResponse,
46             err: nil,
47         },
48         {
49             name: "error-from-call",
50             client: &mockDynamoDBClient{
51                 err: exampleErr,
52             },
53             err: exampleErr,
54         },
55     }
56 }
```

I need to mock an Http call

```
9  var ErrBadStatusCode = errors.New( text: "bad status code from HTTP call")
1
2  type Response struct {
3      ID      int     `json:"id"`
4      Name    string  `json:"name"`
5      Description string `json:"description"`
6  }
7
8  func MakeHTTPCall(url string) (*Response, error) {
9      resp, err := http.Get(url + "/resource/55/foo")
10     if err != nil {
11         return nil, err
12     }
13     if resp.StatusCode != http.StatusOK {           I
14         return nil, ErrBadStatusCode
15     }
16     body, err := ioutil.ReadAll(resp.Body)
17     if err != nil {
18         return nil, err
19     }
20     r := &Response{}
21     if err := json.Unmarshal(body, r); err != nil {
22         return nil, err
23     }
24     return r, nil
25 }
```

```

12 func TestMakeHTTPCall(t *testing.T) {
13     testTable := []struct {
14         name          string
15         server        *httptest.Server
16         expectedResponse *httpexample.Response
17         expectedErr    error
18     }{
19         {
20             name: "happy-server-response",
21             server: httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
22                 w.WriteHeader(http.StatusOK)
23                 w.Write([]byte(`{"id": 1, "name": "kyle", "description": "novice gopher"}`))
24             })),
25             expectedResponse: &httpexample.Response{
26                 ID:           1,
27                 Name:         "kyle",
28                 Description: "novice gopher",
29             },
30             expectedErr: nil,
31         },
32         {
33             name: "happy-server-response",
34             server: httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
35                 w.WriteHeader(http.StatusInternalServerError)
36                 w.Write([]byte(`EXPLOSION`))
37             })),
38             expectedResponse: nil,
39             expectedErr: httpexample.ErrBadStatusCode,
40         },
41     }
42     for _, tc := range testTable {
43         t.Run(tc.name, func(t *testing.T) {
44             defer tc.server.Close()
45             resp, err := httpexample.MakeHTTPCall(tc.server.URL)
46             if !reflect.DeepEqual(resp, tc.expectedResponse) {
47                 t.Errorf("expected (%v), got (%v)", tc.expectedResponse, resp)
48             }
49             if !errors.Is(err, tc.expectedErr) {
50                 t.Errorf("expected (%v), got (%v)", tc.expectedErr, err)
51             }
52         })
53     }
54 }

```

net/http/httptest

We made one test server per subtest, but you could make one server with many routes, just like a normal server!