



GopherChina2018



# Engineering Practices in Tantan using Golang

Henry Ren



# Engineering Practices in Tantan using Golang



Henry Ren  
Product Backend Team Lead  
探探科技  
[henry@tantanapp.com](mailto:henry@tantanapp.com)

GopherChina2018





# Agenda

---

- Overview of Tantan
- Go in Tantan Backend
- Engineering Practices
  - Daily Development
  - Testing Go Code with ❤️
  - Building RESTful API
- Architecture Evolvement
- Conclusion





# Overview of Tantan





# #1 How Tantan works





# #2 Daily Swipes

---

1+ Billion

RESTful API

**PUT** /users/me/relationships/:uid





# Go in Tantan Backend





# #1 Building Tantan Backend

---

- **Goals**

- **Clean Code** Simplicity, Readability, Maintainability, Testability
- **Development Efficiency** Ease of Development & Deployment
- **Scalability** Vertical & Horizontal
- **Performant**

- **Challenges**

- **Complexities** Product Requirements
- **Software Quality**
- **Continuous Iterations** Development & Deployment
- **Small Team** 3 Backend Developers in the Beginning







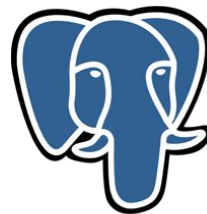
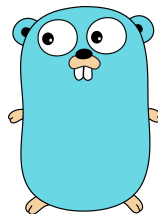
# #2 Tech Stacks

---

- Team members in May, 2014
  - Developers with different background (PHP, C#, Python, C)
  - Developers speak different languages (English, Finnish, Swedish, Chinese)
  - Build first version within 4-5 weeks

- Tech stacks

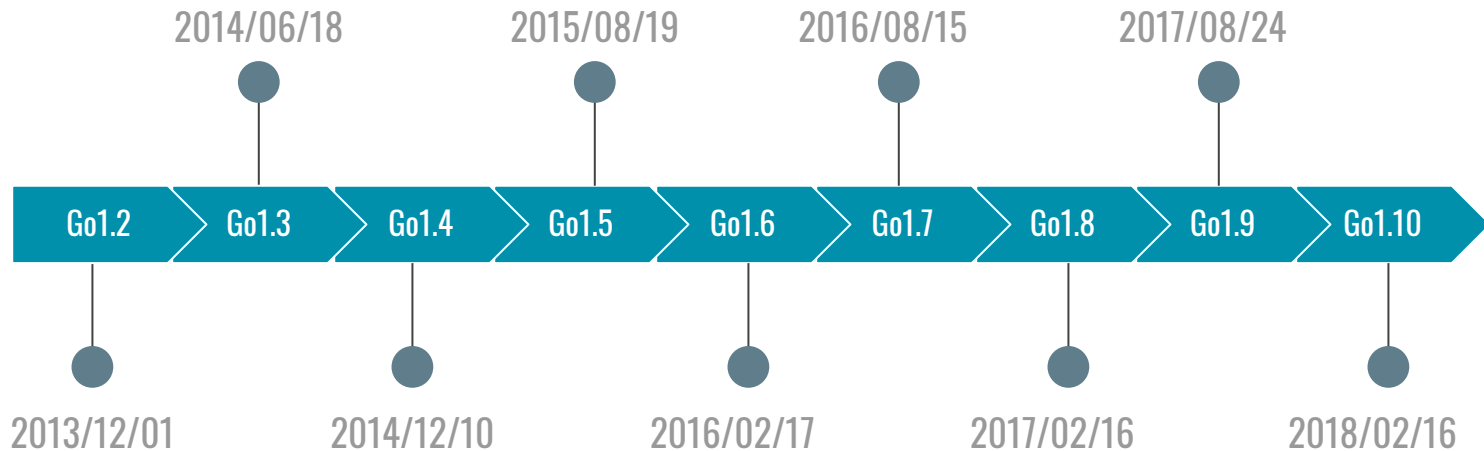
- Golang
  - Simplicity
  - Performant
  - Fun
- PostgreSQL
  - PostGIS Extension for Location Based Services
  - Advanced Data Types, Partial Indexes, Stored Procedures etc.





# #3 Go versions

---





# #4 Lines of Go Code

---



Language	files	blank	comment	code
Go	1683	33520	6096	187130

Lines of Go code in Backend Services Repository





# #5 Backend Services Built in Go

---

- HTTP web servers
  - Using **net/http** package
  - RESTful API Services
  - Media (Image/Video/Audio) Uploading & Downloading Service
- RPC servers
  - Using **net/rpc** package
- Web apps
  - Using **Revel** web framework
- Cli programs
- Cron jobs
- etc.





# #6 Engineering Practices

---

- Daily Development
- Testing Go Code with ❤️
- Building RESTful API





# Engineering Practices: Daily Development





# #1 Repository

---

- Single repository using Github
- Repository organization
  - cmd/: all main packages or files
  - app/: specific app implementations
  - version/: binary versioning
  - vendor/: vendoring packages
  - doc/: documentation in Markdown
  - db/: database changes
  - etc.
- Workflow
  - Fork -> Pull Request -> Code Review

```
▶ app
▶ build
▶ cmd
▶ config
▶ db
▶ doc
▶ domain
▶ vendor
▶ version
≡ .gitignore
≡ Gopkg.lock
≡ Gopkg.toml
≡ Makefile
M↓ README.md
```





# #2 Go Tools

---

- go fmt
- go test
- go build / go install / go run
  - “It’s a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.”
- go doc
  - go doc json / go doc json.Decoder / go doc json.Decoder.Decode
  - godoc -http=:8080 // check go documentation and standard libraries while golang.org not available
- go vet
  - report likely mistakes in packages
- Complete list: <https://golang.org/cmd/>







# #3 IDEs

---

- Vim, Emacs, Sublime Text, GoLand, VS Code, etc.
- Development Plugins with different Go tools integrated (vscode-go for example)
  - Completion Lists (using gocode)
  - Build-on-save (using go build and go test)
  - Lint-on-save (using golint or gometalinter)
  - Format on save as well as format manually (using goreturns or goimports or gofmt)
  - Add Imports (using gopkgs)
  - Add/Remove Tags on struct fields (using gomodifytags)
  - Run Tests under the cursor, in current file, in current package, in the whole workspace (using go test)
  - Show code coverage
  - etc.





# #4 Packages

---

- Standard Packages

- net/http
- encoding/json
- context cancellation, timeouts, request scoped data etc
- reflect
- etc.

- External Packages

- github.com/julienschmidt/httprouter
- github.com/braintree/manners
- github.com/revel/revel
- github.com/stretchr/testify/assert
- etc.

```
203 fmt
188 time
143 strings
136 log
108 net/http
107 errors
104 encoding/json
96 flag
94 strconv
93 os
74 sync
73 io/ioutil
70 bytes
59 io
43 context
```

```
go list -f '{{ join .Imports "\n" }}' ./...
```





# #5 Versioning

---

- Compile version info into Go binary

- `go build -o tantan-swipe-service \`
- `-ldflags "-X tantan/version.version=master-21a5f142fe3041a1ef6ec17d86a15423829d5ddc \`
- `-X tantan/version.date=2018-04-05T20:44:34+0800" \`
- `cmd/tantan-swipe-service/main.go`
- `./tantan-swipe-service -version`
- Version: master-21a5f142fe3041a1ef6ec17d86a15423829d5ddc
- Binary: ./tantan-swipe-service
- Compile date: 2018-04-05T20:44:34+0800

*-X importpath.name=value*  
*Set the value of the string variable*  
*in importpath named name to value.*

*<https://golang.org/cmd/link/>*





```
package version
```

```
import (  
    "flag"  
    "fmt"  
    "io"  
    "os"  
)
```

```
var showVersion = flag.Bool("version", false, "Print version of this binary")
```

```
var (  
    version string  
    date    string  
)
```





```
func init() {  
    if !flag.Parsed() {  
        flag.Parse()  
    }  
    if showVersion != nil && *showVersion {  
        printVersion(os.Stdout, version, date)  
        os.Exit(0)  
    }  
}  
  
func printVersion(w io.Writer, version string, date string) {  
    fmt.Fprintf(w, "Version: %s\n", version)  
    fmt.Fprintf(w, "Binary: %s\n", os.Args[0])  
    fmt.Fprintf(w, "Compile date: %s\n", date)  
}
```





# #6 Profiling

---

- Profiling Types
  - cpu profile
  - mem profile
  - blocking profile, goroutine profile etc.
- Profiling Generation
  - runtime/pprof
  - net/http/pprof
  - go test . -bench . -cpuprofile prof.cpu
- Profiling Visualization
  - go tool pprof cpu.prof
  - Flame graph profiler for Go programs: <https://github.com/uber/go-torch>





# #7 Code Review

---

- Coding standards
- Go Code Review Comments

<https://github.com/golang/go/wiki/CodeReviewComments>





# #8 Others

---

- **Common issues**
  - for ... range loop variable reuse
  - http response body not closed
  - goroutine lifetime
  - etc.
- **Concurrency patterns**
  - goroutine: execution
  - channel: communication message queuing
  - select: coordination
- **Reflection: runtime reflection, allows a program to manipulate objects with arbitrary types**
- **Monitoring: Integration with Prometheus**





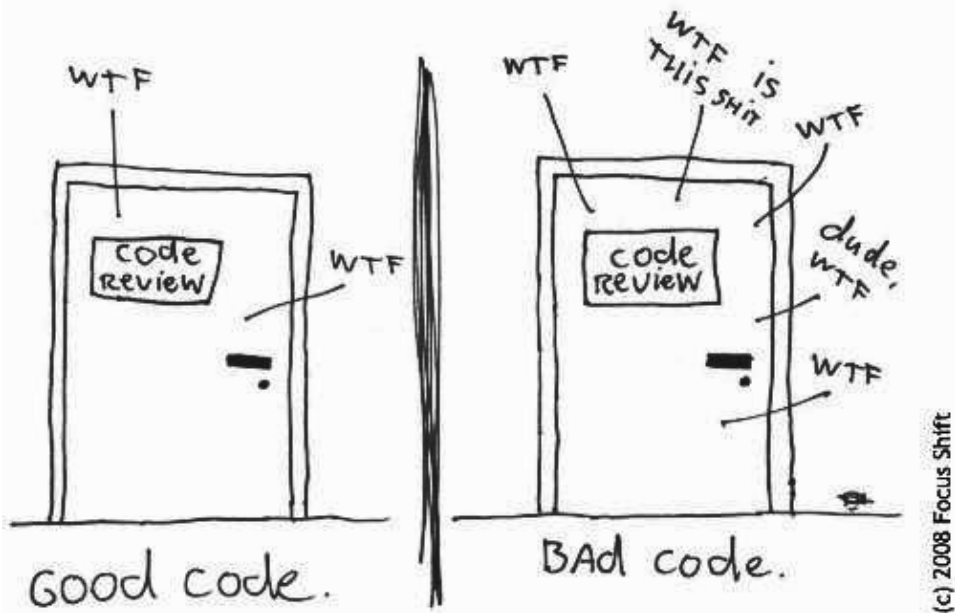


# Engineering Practices: Testing Go Code





# The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE



[http://www.osnews.com/story/19266/WTFs\\_m](http://www.osnews.com/story/19266/WTFs_m)





# #1 Testable Code

---

- Why it matters?
  - Development Time vs. Maintenance Time
  - Make iterations or CI/CD easier (bug fixes etc.)
- Code Quality
  - Readability
  - Maintainability
  - **Testability**
- Continuous Integration
  - Automatic Testing

```
Henry Ren, 4 years ago | 1 author (Henry Ren)  
67 type Dispatcher struct {  
68     server *Server  
69 }
```





# #2 Testing in Go

- A lightweight test framework using:
  - `go test` command
  - `testing` package
- Go test files: `_test.go`

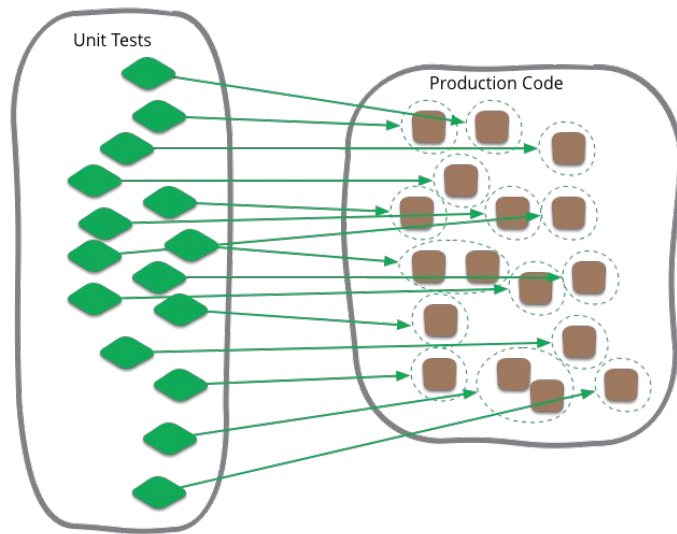
```
// foo_test.go (excluded from go build)
```

```
import "testing"
```

```
func TestXxx(t *testing.T) {}
```

```
func ExampleXxx() {}
```

```
func BenchmarkXxx(b *testing.B) {}
```



<https://martinfowler.com/bliki/UnitTest.html>





# testing.TB interface

---

- Implemented by **testing.T** and **testing.B** structs

```
type TB interface {  
    Error(args ...interface{})  
    Errorf(format string, args ...interface{})  
    Fail()  
    FailNow() // signal failures  
    Failed() bool  
    Fatal(args ...interface{})  
    // ...  
}
```





# testdata

---

- "The go tool will ignore a directory named **testdata**", making it available to hold ancillary data needed by the tests."





# #3 Testing Strategies

---

- Packages to place Go test files
- Same Package
  - White Box testing
  - Tests can access unexported variables, functions etc.
- Separate Package
  - Black Box testing
  - Tests can access only exported variables, functions etc.
  - Mandatory sometimes due to import cycles:
    - **testing** package imports **strings**
    - **strings** tests need to import **testing** package
    - Separate **strings\_test** package is used

```
// foo.go  
package foo
```

```
// foo_test.go (white box testing)  
package foo
```

```
// foo_test.go (black box testing)  
package foo_test
```

```
import . "bar"
```





# #4 Example Tests

---

- Examples on how to use your code
- Examples source code defined by **testing** package
- Examples functions can be run by **go test** command
- Output is compared with function standard output
- A separate `example_test.go` file in Go built-in libraries
  - `func ExampleXxx() {}`

```
func ExampleCompare() {  
    fmt.Println(strings.Compare("a", "b"))  
    fmt.Println(strings.Compare("a", "a"))  
    fmt.Println(strings.Compare("b", "a"))  
    // Output:  
    // -1  
    // 0  
    // 1  
}
```







```
$ cd /usr/local/go/src/strings
$ go test -v example_test.go
=== RUN   ExampleFields
--- PASS: ExampleFields (0.00s)
=== RUN   ExampleFieldsFunc
--- PASS: ExampleFieldsFunc (0.00s)
=== RUN   ExampleCompare
--- PASS: ExampleCompare (0.00s)
...
PASS
ok  command-line-arguments 0.009s
```





# #5 Benchmark Tests

---

- Profiling
  - CPU
  - Mem
  - Goroutine block
- `go test -bench=.`





```
$ cd /usr/local/go/src/strings
$ go test -v -bench='BenchmarkToUpper' -run='^$' strings_test.go
goos: darwin
goarch: amd64
BenchmarkToUpper/#00-8          300000000          5.74 ns/op
BenchmarkToUpper/ONLYUPPER-8    100000000          14.1 ns/op
BenchmarkToUpper/abc-8          300000000          40.8 ns/op
BenchmarkToUpper/AbC123-8       300000000          48.0 ns/op
BenchmarkToUpper/azAZ09_-8      300000000          47.1 ns/op
BenchmarkToUpper/longStringwitHmixofsmaLLandcAps-8      20000000          89.7 ns/op
BenchmarkToUpper/longestringewithenonasciï¼chars-8    5000000            314 ns/op
BenchmarkToUpper/æææææ-8        5000000            268 ns/op
PASS
ok      command-line-arguments    13.429s
```





# #6 HTTP Testing

---

- `net/http/httptest` Package
- Testing HTTP Clients and Servers





```
func requestHandleFunc(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json; charset=utf-8")
    w.WriteHeader(http.StatusOK)
    w.Write([]byte(`{"status": "ok"}`))
}

func TestRequestHandleFunc(t *testing.T) {
    req, _ := http.NewRequest("GET", "/", nil)
    w := httptest.NewRecorder() // response recorder for later inspection

    requestHandleFunc(w, req)
    if status := w.Code; status != http.StatusOK {
        t.Errorf("expected status code: %d, got: %d", http.StatusOK, status)
    }
}
```





# #7 TestMain

---

- Runs in the main goroutine
- Set up or tear down things before or after **multiple tests**
- `setUp()` and `tearDown()` are run only once for multiple tests
- Use cases
  - Initialize and close db connection before and after multiple tests
  - Subprocess testing

```
func TestMain(m *testing.M) {  
    setUp()  
    code := m.Run()  
    tearDown()  
    os.Exit(code)  
}
```





# #8 Multiple Test Cases

---

- Table Driven Tests
- Define separate functions
  - Code duplication
- Use **subtests** in **testing** package
  - Better table driven tests with hierarchy
  - Better testing output format
  - Could add `setup` and `tearDown` before and after subtests
  - `func (t *T) Run(name string, f func(t *T)) bool {}`
  - `func (b *B) Run(name string, f func(b *B)) bool {}`





```
// net/http/httpptest/server_test.go
func TestServer(t *testing.T) {
    for _, name := range []string{"NewServer", "NewServerManual"} {
        t.Run(name, func(t *testing.T) {
            newServer := newServers[name]
            t.Run("Server", func(t *testing.T) { testServer(t, newServer) })
            t.Run("GetAfterClose", func(t *testing.T) { testGetAfterClose(t, newServer) })
            t.Run("ServerCloseBlocking", func(t *testing.T) { testServerCloseBlocking(t, newServer)
        })
            t.Run("ServerCloseClientConnections", func(t *testing.T) {
testServerCloseClientConnections(t, newServer) })
            t.Run("ServerClientTransportType", func(t *testing.T) { testServerClientTransportType(t,
newServer) })
        })
    }
    ...
}
```







```
$ go test -v -run='^TestServer$' net/http/httptest/.
=== RUN   TestServer
=== RUN   TestServer/NewServer
=== RUN   TestServer/NewServer/Server
=== RUN   TestServer/NewServer/GetAfterClose
=== RUN   TestServer/NewServer/ServerCloseBlocking
...
--- PASS: TestServer (0.01s)
    --- PASS: TestServer/NewServer (0.00s)
        --- PASS: TestServer/NewServer/Server (0.00s)
        --- PASS: TestServer/NewServer/GetAfterClose (0.00s)
        --- PASS: TestServer/NewServer/ServerCloseBlocking (0.00s)
...
PASS
ok  net/http/httptest      0.019s
```





# #9 Skipping Tests

---

- Skip tests explicitly using **t.Skip()** method
  - Based on flags, environment variables, conditions etc.
- Skip tests using **-run** flag
  - regular expression pattern
- Skip tests using **-short** flag
- Skip tests using **-timeout** flag





```
$ export DEBUG_MODE="true"
$ go test -v skipping_tests_test.go
=== RUN   TestSkip
--- SKIP: TestSkip (0.00s)
    skipping_tests_test.go:21: test skipped
PASS
ok      command-line-arguments  0.019s
```

```
import (
    "os"
    "testing"
)

func TestSkip(t *testing.T) {
    if os.Getenv("DEBUG_MODE") == "true" {
        t.Skipf("test skipped")
    }
}
```





```
$ go test -v -run='^TestContainsAny$' strings_test.go
```

```
=== RUN    TestContainsAny
```

```
--- PASS: TestContainsAny (0.00s)
```

```
PASS
```

```
ok      command-line-arguments 0.085s
```





```
$ go test -v skipping_tests_test.go
=== RUN   TestSkipUsingShort
--- FAIL: TestSkipUsingShort (0.00s)
FAIL
FAIL    command-line-arguments  0.042s

$ go test -v -short skipping_tests_test.go
=== RUN   TestSkipUsingShort
--- SKIP: TestSkipUsingShort (0.00s)
    skipping_tests_test.go:7: test skipped
PASS
ok     command-line-arguments  0.020s
```

```
import "testing"

func TestSkipUsingShort(t *testing.T) {
    if testing.Short() {
        t.Skip("test skipped")
    }
    t.FailNow()
}
```





```
$ go test -v -timeout 1s skipping_tests_test.go
=== RUN   TestSkipUsingTimeout
panic: test timed out after 1s
...
FAIL     command-line-arguments  1.010s
```

```
$ go test -v -timeout 5s skipping_tests_test.go
=== RUN   TestSkipUsingTimeout
--- PASS: TestSkipUsingTimeout (3.00s)
PASS
ok      command-line-arguments  3.007s
```

```
import (
    "testing"
    "time"
)

func TestSkipUsingTimeout(t *testing.T) {
    time.Sleep(time.Second * 3)
}
```





# #10 Running Tests in Parallel

---

- Tests for specific package are executed sequentially by default
- Run tests in Parallel using `t.Parallel()` method





```
$ go test -v parallel_tests_test.go
=== RUN   TestParallelSleepOneSecond
--- PASS: TestParallelSleepOneSecond (1.00s)
=== RUN   TestParallelSleepTwoSecond
--- PASS: TestParallelSleepTwoSecond (2.00s)
=== RUN   TestParallelSleepThreeSecond
--- PASS: TestParallelSleepThreeSecond (3.00s)
PASS
ok      command-line-arguments  6.018s
```

```
import (
    "testing"
    "time"
)

func TestParallelSleepOneSecond(t *testing.T) {
    time.Sleep(time.Second)
}

func TestParallelSleepTwoSecond(t *testing.T) {
    time.Sleep(time.Second * 2)
}

func TestParallelSleepThreeSecond(t *testing.T) {
    time.Sleep(time.Second * 3)
}
```







```
$ go test -v parallel_tests_test.go
=== RUN   TestParallelSleepOneSecond
=== PAUSE TestParallelSleepOneSecond
=== RUN   TestParallelSleepTwoSecond
=== PAUSE TestParallelSleepTwoSecond
=== RUN   TestParallelSleepThreeSecond
=== PAUSE TestParallelSleepThreeSecond
=== CONT  TestParallelSleepOneSecond
=== CONT  TestParallelSleepThreeSecond
=== CONT  TestParallelSleepTwoSecond
--- PASS: TestParallelSleepOneSecond (1.00s)
--- PASS: TestParallelSleepTwoSecond (2.00s)
--- PASS: TestParallelSleepThreeSecond (3.00s)
PASS
ok      command-line-arguments  3.017s
```

```
import (
    "testing"
    "time"
)

func TestParallelSleepOneSecond(t *testing.T) {
    t.Parallel()
    time.Sleep(time.Second)
}

func TestParallelSleepTwoSecond(t *testing.T) {
    t.Parallel()
    time.Sleep(time.Second * 2)
}

func TestParallelSleepThreeSecond(t *testing.T) {
    t.Parallel()
    time.Sleep(time.Second * 3)
}
```





# #11 Testing Output

---

- `go test -v .`
  - go2xunit: <https://github.com/tebeka/go2xunit>
  - Convert "go test" output to xunit compatible (used in Jenkins/Hudson)
- `go test -v -json .`
  - json flag introduced in Go1.10





```
$ cd /usr/local/go/src/strings
```

```
$ go test -v -json example_test.go
```

```
{"Time":"2018-04-15T11:37:24.804212777+08:00","Action":"run","Package":"command-line-arguments","Test":"ExampleFields"}
```

```
{"Time":"2018-04-15T11:37:24.804431761+08:00","Action":"output","Package":"command-line-arguments","Test":"ExampleFields","Output":"=== RUN ExampleFields\n"}
```

```
{"Time":"2018-04-15T11:37:24.804457082+08:00","Action":"output","Package":"command-line-arguments","Test":"ExampleFields","Output":"--- PASS: ExampleFields (0.00s)\n"}
```

```
....
```

```
{"Time":"2018-04-15T11:37:24.810771405+08:00","Action":"pass","Package":"command-line-arguments","Elapsed":0.012}
```





# #12 Testing Coverage

---

- Statement test coverage
  - 100% vs. 0%
- Go test coverage implementation
  - <https://blog.golang.org/cover>
- Test coverage report
  - `go test -cover {pkg}`
  - `go test -cover -coverprofile=cover.out {pkg}`
  - `go tool cover -func=cover.out`
- Test coverage visualization
  - `go tool cover -html=cover.out -o coverage.html`
  - IDE VSCode “Toggle Test Coverage”

```
$ go test -cover encoding/json
ok  encoding/json  0.995s  coverage: 90.7%
of statements

$ go test -cover context
ok  context  2.617s  coverage: 97.2% of
statements

$ go test -cover net/http
ok  net/http  35.131s  coverage: 78.9% of
statements
```





```
context/context.go (97.2%) not tracked not covered covered
func (e *emptyCtx) String() string {
    switch e {
    case background:
        return "context.Background"
    case todo:
        return "context.TODO"
    }
    return "unknown empty Context"
}

var (
    background = new(emptyCtx)
    todo       = new(emptyCtx)
)

// Background returns a non-nil, empty Context. It is
// values, and has no deadline. It is typically used b
// initialization, and tests, and as the top-level Con
// requests.
func Background() Context {
    return background
}
```





# #13 Integration Testing

---

- Background: automate testing process by making several sequential API requests
- Implemented as a complete framework with “go test” command and testing package
  - Automatic RESTful API tests using “go test”
  - JSON Schema Validation
  - Simulating HTTP Clients
- External packages we used
  - [github.com/stretchr/testify](https://github.com/stretchr/testify)
  - [github.com/xeipuuv/gojsonschema](https://github.com/xeipuuv/gojsonschema)





```
$ go test api/service/moments_test.go -v -debug=false -baseUrl=https://example.com/v1
=== RUN   TestGetMoments
--- PASS: TestGetMoments (0.42s)
=== RUN   TestGetMomentsPagination
--- PASS: TestGetMomentsPagination (1.06s)
=== RUN   TestPostMoments
--- PASS: TestPostMoments (0.11s)
=== RUN   TestPatchMoments
--- PASS: TestPatchMoments (0.10s)
=== RUN   TestPutMoments
--- PASS: TestPutMoments (0.11s)
=== RUN   TestDeleteMoments
--- PASS: TestDeleteMoments (0.09s)
...
PASS
ok      command-line-arguments      2.570s
```

**DEMO**  
(if time permits)





# #14 In Summary

---

- More scenarios in using **go test** command and **testing** package
  - Dependency Injection using Interfaces
  - Concurrency Testing
  - Mocking
- Writing tests in Go has a lot of fun
  - Fast Build
  - Test and Cover
- Great examples and idiomatic practices in Go built-in packages
- A lot of testing library/frameworks
  - Toolkit with common assertions and mocks <https://github.com/stretchr/testify>
  - BDD Testing Framework for Go <https://github.com/onsi/ginkgo>
  - etc.







# Engineering Practices: Building RESTful API





# #1 REST

---

- “Representational state transfer (REST) is the software architectural style of the World Wide Web” - *wikipedia*
  - Resources (resource name, eg. URI )
  - Representation (an Internet media type for the data between client and server, eg. JSON,XML)
  - State Transfer (standard HTTP methods, eg. GET, POST etc)





# #2 RESTful API Design

---

- Resource Oriented
  - Concepts
    - Resources
    - Their names (URIs)
    - Their representations
    - The links between them
  - Properties
    - Addressability
    - Statelessness
    - Connectedness
    - A uniform interface

《RESTful web services》- Leonard Richardson & Sam Ruby





# #3 JSON

---

- JSON for resource representation
  - API POST/PATCH/PUT request body
  - API response
  - > curl -H "Accept: application/json" -H "Content-type: application/json" -d '{"subject": "Engineering Practices in Tantan using Go", "speaker": "henry", "location": "shanghai"}' <https://example.com/v1/talks>
  - < 201 Created
  - < {"code": 201, "error": "", "message": "Created"}





# #4 Building RESTful API using Go

---

- **net/http** package
- Popular Go frameworks for building RESTful APIs
  - beego
  - gin
  - etc.





# Architecture Evolvement





# #1 Background

---

- Service Oriented Architecture
- Monolithic RESTful API Service





# #2 Challenges

---

- Growing Backend Teams
- Team Collaboration
- Loose Coupling
- Delivery Efficiency







# #3 Looking forward

---

- Writing testable code **using Go**
- Refactoring architecture to Microservices **using Go** (working on)
  - Domain Driven Design
  - Clear Service Definition and Boundaries
  - Service Communication, Discovery & Registry etc.
- Continuous integration/deployment **using Go**
  - Create tests with reasonable test coverage
  - Run tests fast and periodically
  - Automation





# Conclusion





# It's Fun in Coding Go

---

- "Go is powerful enough to make a lot happen in a few lines." — Effective Go
- It was a wise decision to choose Go 4 years ago at Tantan.
  - Move fast while keeping software quality
  - Happy hacking in coding Go and PostgreSQL
- Use Go for your next project.





# Thanks!

**Henry Ren**

henry@tantanapp.com



**GopherChina2018**

