Inception

Go programs that generate Go code

a few words about Go

Go is very simple

Go is very powerful

Go has great tools

Go stdlib is great

Go is awesome

@ernesto_jimenez

Go is what C++ should have been

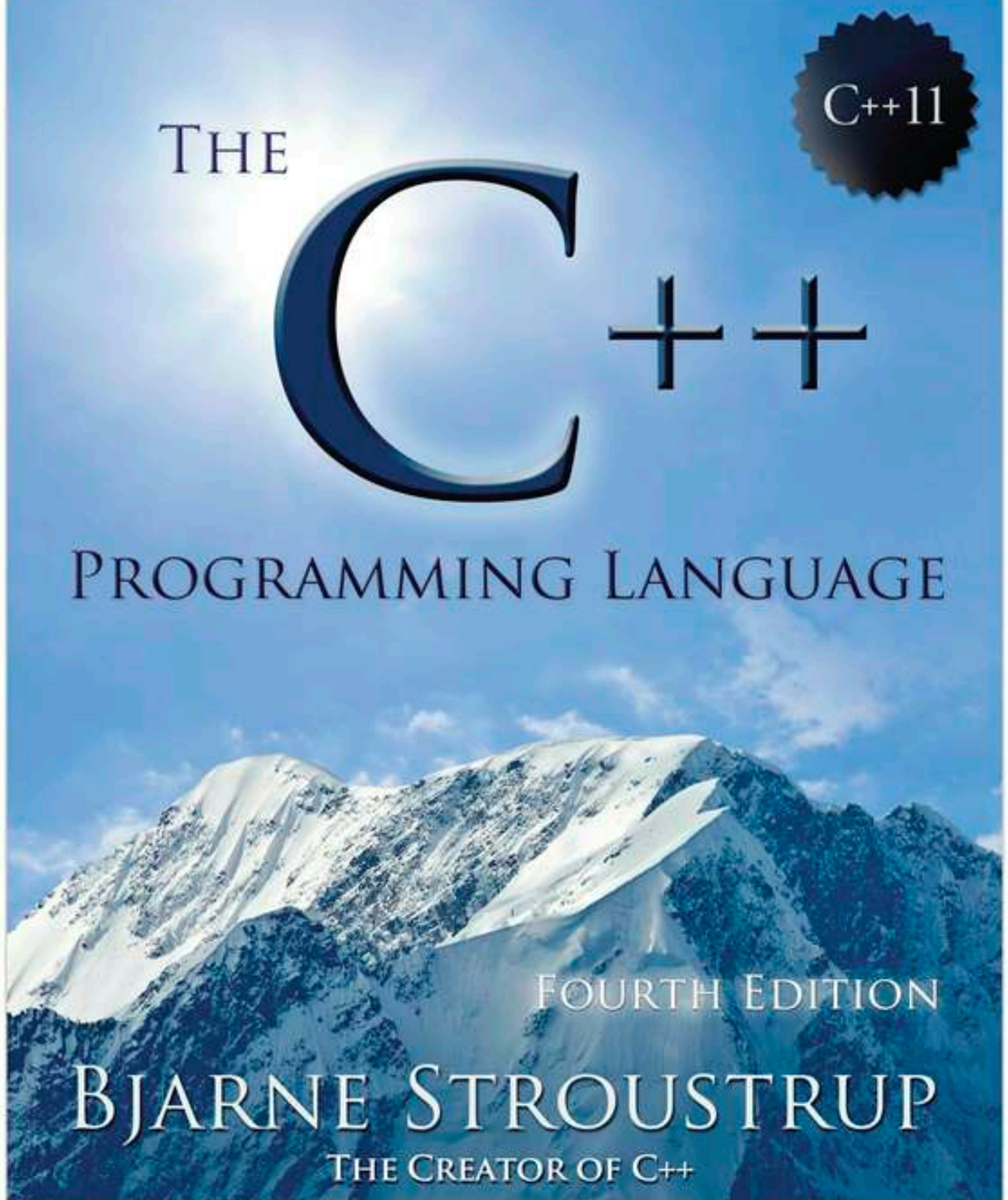
@ernesto_jimenez's joke

SECOND EDITION

THE C PROGRAMMING LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

272 pages



1.368 pages

just C++?

The Go Programming Language

Alan A. A. Donovan
Brian W. Kernighan

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

400 pages

SECOND EDITION

THE C PROGRAMMING LANGUAGE

ANSI C

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

The Go Programming Language

Alan A. A. Donovan
Brian W. Kernighan

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

I really like using it

Slackline

Pricing Community channels Open slackline

COLLABORATE SEAMLESSLY WITH OTHER TEAMS AND COMPANIES

Connect different Slack teams together. Save time and energy when working with other companies.

Your email Get started

SOME OF OUR CUSTOMERS

Bloomberg BETA DOW JONES HARVARD T.H. CHAN SCHOOL OF PUBLIC HEALTH UNIVERSAL UNIVERSAL MUSIC

ernesto-jimenez fix #201 - Remove broken link from README 9d7bb92 7 days ago

14 contributors

333 lines (230 sloc) 10.8 KB Raw Blame History

Testify - Thou Shalt Write Tests

build passing

Go code (golang) set of packages that provide many tools for testifying that your code will behave as you intend.

Features include:

- Easy assertions
- Mocking
- HTTP response trapping
- Testing suite interfaces and functions

Get started:

- Install testify with one line of code, or update it with another
- For an introduction to writing test code in Go, see http://golang.org/doc/code.html#Testing
- Check out the API Documentation http://godoc.org/github.com/stretchr/testify

no silver bullet

lack of package versioning

boilerplate

this talk

reducing boilerplate

Go programs that:

1. Parse data to generate Go code

2. Modify Go code

3. Parse Go code to generate Go code

1. Parsing data to generate Go code

API Discovery Service

Programmatically read metadata about Google APIs.

What is the Google API Discovery Service?

Build tools to work with APIs

You can use the Google API Discovery Service to build client libraries, IDE plugins, and other tools that interact with Google APIs. It provides a lightweight, JSON-based API that exposes machine-readable metadata about Google APIs, including:

- A directory of supported APIs.
- A machine-readable "Discovery document" for each of the supported APIs that includes:
  - A list of API resource schemas based on JSON Schema.
  - A list of API methods and available parameters for each method.
  - A list of available OAuth 2.0 scopes.
- Inline documentation of methods, parameters, and available parameter values.

GET STARTED REFERENCE

google / google-api-go-client

Watch 52 Star 448 Fork 83

Branch: master google-api-go-client / README.md

gmlewis google-api-go-client: add Application Default Credential example 8cbf97d on Sep 11

2 contributors

93 lines (64 sloc) 2.81 KB Raw Blame History

Google APIs Client Library for Go

Status

build passing

These are auto-generated Go libraries from the Google Discovery Service's JSON description files of the available "new style" Google APIs.

Due to the auto-generated nature of this collection of libraries, complete APIs or specific versions can appear or go away without notice. As a result, you should always locally vendor any API(s) that your code relies upon.

Announcement email:

- http://groups.google.com/group/golang-nuts/browse_thread/thread/6c7281450be9a21e

Getting started documentation:

```
469
470         p, pn := a.p, a.pn
471         reslist := a.Resources(a.m, "")
472
473         p("// Package %s provides access to the %s.\n", pkg, jstr(m, "title"))
474         docsLink = jstr(m, "documentationLink")
475         if docsLink != "" {
476                 p("//\n")
477                 p("// See %s\n", docsLink)
478         }
479         p("//\n// Usage example:\n")
480         p("//\n")
481         p("//   import %q\n", a.Target())
482         p("//   ...\n")
483         p("//   %sService, err := %s.New(oauthHttpClient)\n", pkg, pkg)
484
485         p("package %s // import %q\n", pkg, a.Target())
486         p("\n")
487         p("import (\n")
488         for _, pkg := range []string{
489                 "bytes",
490                 "encoding/json",
491                 "errors",
492                 "fmt",
493                 "io",
494                 "net/http",
495                 "net/url",
496                 "strconv",
497                 "strings",
498                 *contextHTTPPkg,
499                 *contextPkg,
500                 *googleapiPkg,
501                 *internalPkg,
502         } {
503                 p("\t%q\n", pkg)
504         }
505         p(")\n\n")
506         pn("// Always reference these packages, just in case the auto-generated code")
507         pn("// below doesn't.")
508         pn("var _ = bytes.NewBuffer")
509         pn("var _ = strconv.Itoa")
510         pn("var _ = fmt.Sprintf")
511         pn("var _ = json.NewDecoder")
```

The Heroku HTTP API Toolchain

Posted about a year ago by Mark McGranaghan

Today we're open sourcing the toolchain Heroku uses to design, document, and consume our HTTP APIs. We hope this shows how Heroku thinks about APIs and gives you new tools to create your own.

This toolchain includes:

- An HTTP API design guide, describing how we structure both internal and public-facing APIs and document them using the JSON Schema standard.
- A tool for working with JSON schemas and using them to generate API documentation.
- Ruby and Go client code generators for APIs with JSON schemas.

Here's some more information about these things, how we use them at Heroku, and an explanation of how you can try them yourself.

JSON Schema Foundation

We've developed the toolchain around the the JSON Schema standard for describing HTTP+JSON APIs. Having a consistent way to describe APIs gives us a powerful starting point for the toolchain described below.

HTTP API Design Guide

json-schema.org

The home of JSON Schema

What does it do?

JSON Schema describes your JSON data format

JSON Hyper-Schema turns your JSON data into hyper-text

Advantages

JSON Schema

- describes your existing data format
- clear, human- and machine-readable documentation
- complete structural validation, useful for
  - automated testing
  - validating client-submitted data

JSON Hyper-Schema

- describes your existing API - no new structures required
- links (including URI Templates for target URIs)
- forms - specify a JSON Schema for the desired data

More

Interested? Check out:

- the specification
- some examples
- this excellent guide for schema authors, from the Space Telescope Science Institute
- the growing list of JSON (Hyper-)Schema software

Questions? Feeling helpful? Get involved on:

- the GitHub repo
- the Google Group

cyberdelia Update README with instructions for go generate 3f28c7a on Nov 28, 2014

3 contributors

156 lines (117 sloc) 3.71 KB Raw Blame History

Schematic

Generate Go client code for HTTP APIs described by JSON Hyper-Schemas.

Installation

Download and install:

```
$ go get -u github.com/interagent/schematic/cmd/schematic
```

Warning: schematic requires Go >= 1.2.

Client Generation

Run it against your schema:

```
$ schematic platform-api.json > heroku/heroku.go
```

```
import "text/template"
```

interagent / schematic

Branch: master **schematic / templates / funcs.tmpl**

ernesto-jimenez Do not use rel=instances to infer array values 2b9479c on Nov 11, 2014

2 contributors

23 lines (19 sloc) | 815 Bytes Raw Blame History

```
1  {{$Name := .Name}}
2  {{$Def := .Definition}}
3  {{range .Definition.Links}}
4    {{if .AcceptsCustomType}}
5     type {{paramType $Name .}} {{linkGoType .}}
6    {{end}}
7
8    {{if (defineCustomType $Def .)}}
9     type {{returnType $Name $Def .}} {{$Def.ReturnedGoType .}}
10   {{end}}
11
12   {{asComment .Description}}
13   func (s *Service) {{printf "%s-%s" $Name .Title | initialCap}}({{params $Name .}}) ({{values $Name $Def .}}) {
14     {{if ($Def.EmptyResult .)}}
15       return s.{{methodCap .Method}}(nil, fmt.Sprintf("{{.HRef}}", {{args .HRef}}){{requestParams .}})
16     {{else}}
17       {{$Var := initialLow $Name}}var {{$Var}} {{returnType $Name $Def .}}
18       return {{if ($Def.ReturnsCustomType .)}}&{{end}}{{$Var}}, s.{{methodCap .Method}}(&{{$Var}}, fmt.Sprintf("{{.HRef}}", {{args
19     {{end}}
20   }
21 {{end}}
22
```

ChimeraCoder / gojson

Watch 33 Star 918 Fork 46

Branch: master gojson / README.md

ChimeraCoder Update gofmt arguments 776aa5e on May 29

2 contributors

158 lines (132 sloc) 5.15 KB Raw Blame History

gojson

gojson attempts to generate go struct definitions from json documents

Example

```
$ curl -s https://api.github.com/repos/chimeracoder/gojson | gojson -name=Repository

package main

type Repository struct {
    ArchiveURL       string      `json:"archive_url"`
    AssigneesURL     string      `json:"assignees_url"`
    BlobsURL         string      `json:"blobs_url"`
    BranchesURL      string      `json:"branches_url"`
    CloneURL         string      `json:"clone_url"`
    CollaboratorsURL string      `json:"collaborators_url"`
    CommentsURL      string      `json:"comments_url"`
```

2. Modifying Go code

a big part of Go's awesome tools

```
$ go fmt
```

limit arguments about code style

```
package main
import
"fmt"
    func      main(  )    {
fmt.  Println  (      "Hello world")}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello world")
}
```

auto run on save

golang.org/x/tools/cmd/goimports

```
package main

import "os"

func main() {
    fmt.Println("Hello world")
}
```

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello world")
}
```

golang.org/x/tools/cmd/gorename

github.com/sqs/goreturns

...

powered by the stdlib

```
import "go/build"
// gathers information about Go packages
```

```
import "go/scanner"
import "go/parser"
// read and parse parse Go source code
```

```
import "go/ast"
// declares types used to represent syntax trees
```

```
import "go/format"
// "go fmt" go code, including AST
```

```
import "go/printer"
// implements printing of AST nodes
```

Read code > Generate AST > **Modify AST** > Write code

also used for static analysis

3. Parse Go code to generate Go code

josharian / impl

Branch: master **impl / README.md**

josharian Generate stubs that compile 2d76c51 on Jul 26, 2014

2 contributors

22 lines (15 sloc) | 432 Bytes Raw Blame History

`impl` generates method stubs for implementing an interface.

Sample usage:

```
$ impl 'f *File' io.ReadWriteCloser
func (f *File) Read(p []byte) (n int, err error) {
    panic("not implemented")
}

func (f *File) Write(p []byte) (n int, err error) {
    panic("not implemented")
}

func (f *File) Close() error {
    panic("not implemented")
}
```

You can use `impl` from Vim with vim-go-impl

package mock

```
import "github.com/stretchr/testify/mock"
```

Provides a system by which it is possible to mock your objects and verify calls are happening as expected.

Example Usage

The mock package provides an object, Mock, that tracks activity on another object. It is usually embedded into a test object as shown below:

```
type MyTestObject struct {
  // add a Mock object instance
  mock.Mock

  // other fields go here as normal
}
```

When implementing the methods of an interface, you wire your functions up to call the Mock.Called(args...) method, and return the appropriate values.

For example, to mock a method that saves the name and age of a person and returns the year of their birth or an error, you might write this:

```
func (o *MyTestObject) SavePersonDetails(firstname, lastname string, age int) (int, erro
  args := o.Called(firstname, lastname, age)
  return args.Int(0), args.Error(1)
}
```

The Int, Error and Bool methods are examples of strongly typed getters that take the argument index position. Given this argument list:

```
(12, true, "Something")
```

You could read them out strongly typed like this:

```go
package main

import (
    "testing"

    "github.com/stretchr/testify/mock"
)

type downcaser interface {
    Downcase(string) (string, error)
}

func TestMock(t *testing.T) {
    m := &mockDowncaser{}
    m.On("Downcase", "FOO").Return("foo", nil)
    m.Downcase("FOO")
    m.AssertNumberOfCalls(t, "Downcase", 1)
}
```

```
type mockDowncaser struct {
    mock.Mock
}

func (m *mockDowncaser) Downcase(a0 string) (string, error) {
    ret := m.Called(a0)
    return ret.Get(0).(string), ret.Error(1)
}
```

Branch: master mockery / README.md

kasey Update README installation instructions 6a6c60c on Aug 4

6 contributors

120 lines (79 sloc) 3.1 KB Raw Blame History

mockery

mockery provides the ability to easily generate mocks for golang interfaces. It removes the boilerplate coding required to use mocks.

Installation

`go get github.com/vektra/mockery/.../` , then `$GOPATH/bin/mockery`

Example

Given this is in `string.go`

```
package test

type Stringer interface {
  String() string
```

```
$ mockery -inpkg -testonly -name=downcaser
Generating mock for: downcaser
```

mock_downcaser_test.go

```
package main

import "github.com/stretchr/testify/mock"

type mockDowncaser struct {
    mock.Mock
}

func (_m *mockDowncaser) Downcase(_a0 string) (string, error) {
    ret := _m.Called(_a0)

    var r0 string
    if rf, ok := ret.Get(0).(func(string) string); ok {
        r0 = rf(_a0)
    } else {
        r0 = ret.Get(0).(string)
    }

    var r1 error
    if rf, ok := ret.Get(1).(func(string) error); ok {
        r1 = rf(_a0)
    } else {
        r1 = ret.Error(1)
    }

    return r0, r1
}
```

how do you document the steps to generate that code?

```
$ go generate
```

```go
package main

import (
    "testing"
)

type downcaser interface {
    Downcase(string) (string, error)
}

//go:generate mockery -inpkg -testonly -name=downcaser

func TestMock(t *testing.T) {
    m := &mockDowncaser{}
    m.On("Downcase", "FOO").Return("foo", nil)
    m.Downcase("FOO")
    m.AssertNumberOfCalls(t, "Downcase", 1)
}
```

```
$ go test
# github.com/ernesto-jimenez/test
./main_test.go:14: undefined: mockDowncaser
FAIL    github.com/ernesto-jimenez/test [build failed]

$ go generate
Generating mock for: downcaser

$ go test
PASS
ok      github.com/ernesto-jimenez/test 0.011s
```

vektra / mockery

Watch 3 Star 63 Fork 18

mocks for interfaces with composition are missing methods #18

Edit New Issue

Open **ernesto-jimenez** opened this issue on Apr 8 · 8 comments

ernesto-jimenez commented on Apr 8

Given the following interface:

```
package test

import "io"

type ReadCloser interface {
        io.Reader
        Close() error
}
```

Expected:

```
type ReadCloser struct {
        mock.Mock
}

func (m *ReadCloser) Read(p []byte) (int, error) {
        ret := m.Called(p)
```

Labels

None yet

Milestone

No milestone

Assignee

No one assigned

Notifications

Unsubscribe

You're receiving notifications because you authored the thread.

6 participants

Yeah, **this is a known issue**. I'm not sure how deal with it because **it requires being able to parse entire packages to find the other interfaces**.

`go/ast`

provides very raw data

github.com/ernesto-jimenez/gogen/cmd/goautomock

import "go/types"

// types and algorithms for type-checking of Go packages

less code *and* less issues

package assert

```
import "github.com/stretchr/testify/assert"
```

Package assert provides a set of comprehensive testing tools for use with the normal Go testing system.

Example Usage

The following is a complete example using assert in a standard test function:

```
import (
  "testing"
  "github.com/stretchr/testify/assert"
)

func TestSomething(t *testing.T) {

  var a string = "Hello"
  var b string = "Hello"

  assert.Equal(t, a, b, "The two words should be the same.")

}
```

if you assert many times, use the format below:

```
import (
  "testing"
  "github.com/stretchr/testify/assert"
)

func TestSomething(t *testing.T) {
  assert := assert.New(t)

  var a string = "Hello"
```

Index

Variables
func CallerInfo() []string
func Condition(t TestingT, comp Comparison, msgAndArgs ...interface{}) bool
func Contains(t TestingT, s, contains interface{}, msgAndArgs ...interface{}) bool
func Empty(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func Equal(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func EqualError(t TestingT, theError error, errString string, msgAndArgs ...interface{}) bool
func EqualValues(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func Error(t TestingT, err error, msgAndArgs ...interface{}) bool
func Exactly(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func Fail(t TestingT, failureMessage string, msgAndArgs ...interface{}) bool
func False(t TestingT, value bool, msgAndArgs ...interface{}) bool
func HTTPBody(handler http.HandlerFunc, method, url string, values url.Values) string
func HTTPBodyContains(t TestingT, handler http.HandlerFunc, method, url string, values url.Values, str
interface{}) bool
func HTTPBodyNotContains(t TestingT, handler http.HandlerFunc, method, url string, values url.Values, str
interface{}) bool

tens of assertion helpers

func HTTPSuccess(t TestingT, handler http.HandlerFunc, method, url string, values url.Values) bool
func Implements(t TestingT, interfaceObject interface{}, object interface{}, msgAndArgs ...interface{}) bool
func InDelta(t TestingT, expected, actual interface{}, delta float64, msgAndArgs ...interface{}) bool
func InDeltaSlice(t TestingT, expected, actual interface{}, delta float64, msgAndArgs ...interface{}) bool
func InEpsilon(t TestingT, expected, actual interface{}, epsilon float64, msgAndArgs ...interface{}) bool
func InEpsilonSlice(t TestingT, expected, actual interface{}, delta float64, msgAndArgs ...interface{}) bool
func IsType(t TestingT, expectedType interface{}, object interface{}, msgAndArgs ...interface{}) bool
func JSONEq(t TestingT, expected string, actual string, msgAndArgs ...interface{}) bool
func Len(t TestingT, object interface{}, length int, msgAndArgs ...interface{}) bool
func Nil(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func NoError(t TestingT, err error, msgAndArgs ...interface{}) bool
func NotContains(t TestingT, s, contains interface{}, msgAndArgs ...interface{}) bool
func NotEmpty(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func NotEqual(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) bool
func NotNil(t TestingT, object interface{}, msgAndArgs ...interface{}) bool
func NotPanics(t TestingT, f PanicTestFunc, msgAndArgs ...interface{}) bool
func NotRegexp(t TestingT, rx interface{}, str interface{}, msgAndArgs ...interface{}) bool
func NotZero(t TestingT, i interface{}, msgAndArgs ...interface{}) bool

two functions per assertion

```
func Equal(t TestingT, expected, actual interface{}, msgAndArgs …interface{}) bool {

        if !ObjectsAreEqual(expected, actual) {
                diff := diff(expected, actual)
                return Fail(t, fmt.Sprintf("Not equal: %#v (expected)\n"+
                        "        != %#v (actual)%s", expected, actual, diff), msgAndArgs…)
        }

        return true

}

func (a *Assertions) Equal(expected, actual interface{}, msgAndArgs …interface{}) bool {
        return Equal(a.t, expected, actual, msgAndArgs…)
}
```

package require

```
import "github.com/stretchr/testify/require"
```

Alternative testing tools which stop test execution if test failed.

Example Usage

The following is a complete example using require in a standard test function:

```
import (
  "testing"
  "github.com/stretchr/testify/require"
)

func TestSomething(t *testing.T) {

  var a string = "Hello"
  var b string = "Hello"

  require.Equal(t, a, b, "The two words should be the same.")

}
```

Assertions

The `require` package have same global functions as in the `assert` package, but instead of returning a boolean result they call `t.FailNow()`.

Every assertion function also takes an optional string message as the final argument, allowing custom error messages to be appended to the message the assertion method outputs.

Index

two extra functions per assertion

```
func Equal(t TestingT, expected, actual interface{}, msgAndArgs ...interface{}) {
        if !assert.Equal(t, expected, actual, msgAndArgs...) {
                t.FailNow()
        }
}

func (a *Assertions) Equal(expected, actual interface{}, msgAndArgs ...interface{}) {
        Equal(a.t, expected, actual, msgAndArgs...)
}
```

3 boilerplate functions per each assertion

stretchr / testify

Unwatch 41 Star 1,410 Fork 174

Autogenerate code for require and forwarded assertions #241

Edit

Open **ernesto-jimenez** wants to merge 2 commits into `stretchr:master` from `ernesto-jimenez:codegen`

Conversation 0 Commits 2 Files changed 11 +1,475 −839

ernesto-jimenez commented an hour ago Collaborator

Right now each assertion requires to define 4 methods: the actual assertion and three methods wrapping the assertion.

This PR introduces a Go program to automatically generate the wrappers.

This is still work in progress, since it needs a refactor.

ernesto-jimenez added some commits 3 days ago

- First version of code generation `efd1b85`
- Save file once everything succeeds ✓ `7cb9aa9`

Add more commits by pushing to the **codegen** branch on **ernesto-jimenez/testify**.

All checks have passed
1 successful check

Show all checks

This branch is up-to-date with the base branch

Labels
None yet

Milestone
No milestone

Assignee
No one—assign yourself

Notifications
Unsubscribe
You're receiving notifications because you authored the thread.

1 participant

Lock conversation

```
//go:generate go run ../_codegen/main.go -output-package=assert -template=assertion_forward.go.tmpl
//go:generate go run ../_codegen/main.go -output-package=require -template=require_forward.go.tmpl
//go:generate go run ../_codegen/main.go -output-package=require -template=require.go.tmpl

{{.Comment}}
func (a *Assertions) {{.DocInfo.Name}}({{.Params}}) bool {
        return {{.DocInfo.Name}}(a.t, {{.ForwardedParams}})
}
```

go/doc

go/types

codegen: 239 lines of code

removed: 839 lines of code

lastly: a tiny experiment

```go
package main

import (
    "container/list"
    "fmt"
)

func main() {
    l := list.New()
    l.PushBack(1)
    l.PushBack(2)
    l.PushBack(3)
    for e := l.Front(); e != nil; e = e.Next() {
        if e.Value > 2 {
            fmt.Println(e.Value)
        }
    }
}
```

```
$ go run main.go
# command-line-arguments
./main.go:14: invalid operation: e.Value > 2 (operator > not defined on interface)
```

```
type Element struct {
    // The value stored with this element.
    Value interface{}
}
```

```go
// Must add some type casting
v := e.Value.(int)
if v > 2 {
  fmt.Println(v)
}
```

```
// No type safety on build
l.PushBack("oops")

for e := l.Front(); e != nil; e = e.Next() {
  v := e.Value.(int) // Panic!
  if v > 2 {
    fmt.Println(v)
  }
}
```

could this be keeping us from writing certain reusable packages?

ernesto-jimenez / gogen

Unwatch 1 Star 0 Fork 0

Branch: master gogen / cmd / gospecific / +

ernesto-jimenez Update README Latest commit 2a4144b 12 days ago

..

README.md	Update README	12 days ago
main.go	Fix README, add more tests and specific types	12 days ago

README.md

gospecific

Avoid using generic packages with `interface{}` by generating specific packages that can be used with safe types.

Usage

Install gospecific

```
go get github.com/ernesto-jimenez/gogen/cmd/gospecific
```

Add a go generate comment to generate a package

```
//go:generate gospecific -pkg=container/list -specific-type=string
```

```go
package main

// Copy container/list source and replace interface{} with int
//go:generate gospecific -pkg=container/list -specific-type=int

import (
    “fmt”
    “github.com/ernesto-jimenez/test/list”
)

func main() {
    l := list.New()
    l.PushBack(1)
    l.PushBack(“oops”) // Fails on build time
    l.PushBack(3)
    for e := l.Front(); e != nil; e = e.Next() {
        if e.Value > 2 {
            fmt.Println(e.Value)
        }
    }
}
```

next: writting some packages with reusable concurrency patterns

Wrapping up

you can reduce boilerplate

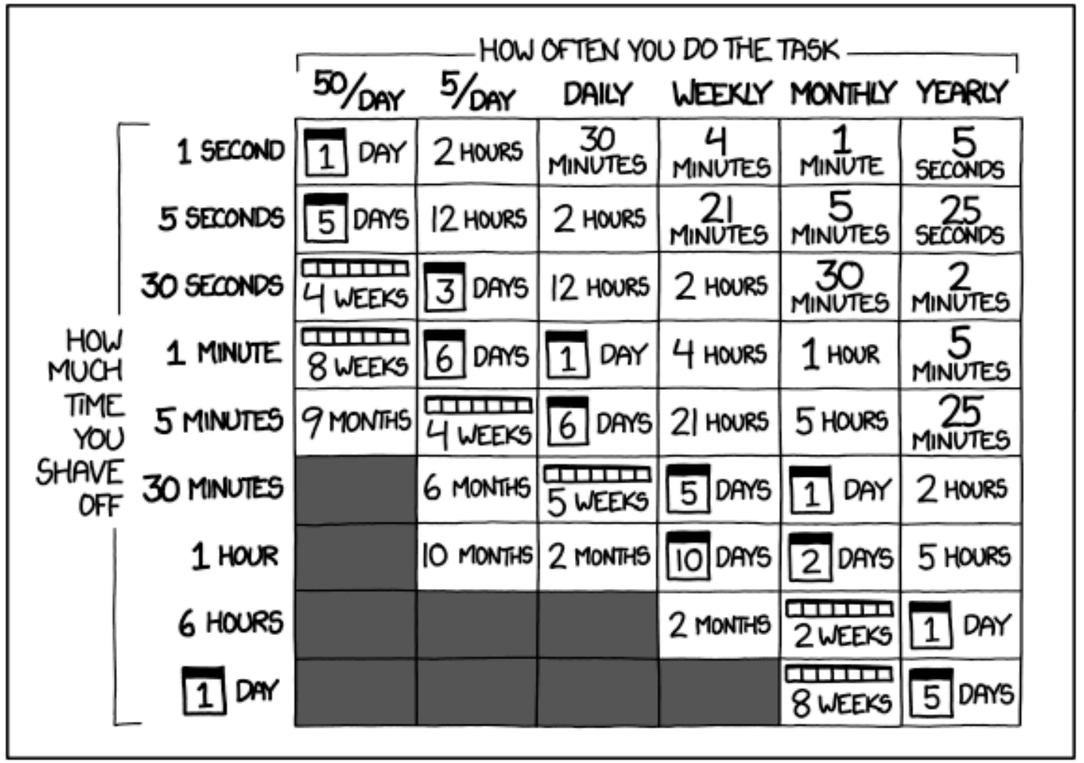
tools already available to reduce boilerplate

stdlib has great packages to build your own tools

many open source projects to learn from

Just remember...

Is It Worth the Time?



Permanent link to this comic: http://xkcd.com/1205/
Image URL (for hotlinking/embedding): http://imgs.xkcd.com/comics/is_it_worth_the_time.png

Questions?

@ernesto_jimenez

ernesto@slakline.io

https://slackline.io/shared_channels/golang