golang测试参考

一. 基础测试

golang测试默认规则：

golang自带了轻量级的测试框架-testing包，一般遵循以下规则

- 规则1：test文件以file_test.go命令
- 规则2：函数名以TestXXX命令，使用(t *testing.T)参数
- 规则3：testdata目录放置test使用的数据
- 规则4：测试框架执行测试函数时调用t.Error、t.Fail等函数认为测试失败。
- 规则5：灵活使用testing内置函数

1.1 入门

```go
// hello.go
package main

import "fmt"

const englishHelloPrefix = "Hello, "

func Hello(name string) string {
    if name == "" {
        name = "World"
    }
    return englishHelloPrefix + name
}

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

对Hello函数进行test，创建test文件

```go
// hello_test.go
package main

import "testing"

func TestHello(t *testing.T) {
    assertCorrectMessage := func(t *testing.T, got, want string) {
        t.Helper()
        if got != want {
            t.Errorf("got '%q' want '%q'", got, want)
        }
    }

    t.Run("saying hello to people", func(t *testing.T) {
        got := Hello("Chris")
        want := "Hello, Chris"
        assertCorrectMessage(t, got, want)
    })

    t.Run("empty string defaults to 'world'", func(t *testing.T) {
        got := Hello("")
        want := "Hello, World"
        assertCorrectMessage(t, got, want)
    })
}
```

1.2 整数

测试整数加法例子

```go
package integers

func Add(x, y int) int {
    return x + y
}
```

```go
package integers

import "testing"

func TestAdder(t *testing.T) {
    sum := Add(2, 2)
    expected := 4

    if sum != expected {
        t.Errorf("expected '%d' but got '%d'", expected, sum)
    }
}
```

执行测试

```
go test -v
```

1.3 循环

测试循环例子

```go
package iteration

const repeatCount = 5

func Repeat(character string) string {
    var repeated string
    for i := 0; i < repeatCount; i++ {
        repeated += character
    }
    return repeated
}
```

```go
package iteration

import "testing"

func TestRepeat(t *testing.T) {
    repeated := Repeat("a")
    expected := "aaaaa"

    if repeated != expected {
        t.Errorf("expected '%q' but got '%q'", expected, repeated)
    }
}

func BenchmarkRepeat(b *testing.B) {
    for i := 0; i < b.N; i++ {
        Repeat("a")
    }
}
```

执行测试

```
go test -v
```

1.4 数组与切片

```go
package main

func Sum(numbers [5]int) int {
    sum := 0
    for _, number := range numbers {
        sum += number
    }
    return sum
}

func SumAll(numbersToSum ...[]int) (sums []int) {
    lengthOfNumbers := len(numbersToSum)
    sums = make([]int, lengthOfNumbers)

    for i, numbers := range numbersToSum {
        sums[i] = Sum(numbers)
    }

    return
}
```

```go
package main

import "testing"

func TestSum(t *testing.T) {

    t.Run("collection of 5 numbers", func(t *testing.T) {
        numbers := []int{1, 2, 3, 4, 5}

        got := Sum(numbers)
        want := 15

        if got != want {
            t.Errorf("got %d want %d given, %v", got, want, numbers)
        }
    })

    t.Run("collection of any size", func(t *testing.T) {
        numbers := []int{1, 2, 3}

        got := Sum(numbers)
        want := 6
        if got != want {
            t.Errorf("got %d want %d given, %v", got, want, numbers)
        }
    })
}

func TestSumAll(t *testing.T) {

    got := SumAll([]int{1,2}, []int{0,9})
    want := []int{3, 9}

    if !reflect.DeepEqual(got, want) {
        t.Errorf("got %v want %v", got, want)
    }
}
```

执行测试

```
go test -v
```

1.5 结构体、方法和接口

```
package main

import "math"

type Rectangle struct {
    Width  float64
    Height float64
}

type Shape interface {
    Area() float64
}

func Perimeter(rectangle *Rectangle) float64 {
    return 2 * (rectangle.Width + rectangle.Height)
}

func (r *Rectangle) Area() float64 {
    return r.Width * r.Height
}

type Triangle struct {
    Base   float64
    Height float64
```

```go
}

func (c *Triangle) Area() float64 {
    return (c.Base * c.Height) * 0.5
}

type Circle struct {
    Radius float64
}

func (c *Circle) Area() float64 {
    return math.Pi * c.Radius * c.Radius
}
```

```go
package main

import "testing"

func TestPerimeter(t *testing.T) {
    rectangle := &Rectangle{10.0, 10.0}
    got := Perimeter(rectangle)
    want := 40.0

    if got != want {
        t.Errorf("got %.2f want %.2f", got, want)
    }
}

func TestArea(t *testing.T) {

    areaTests := []struct {
        name    string
        shape   Shape
        hasArea float64
    }{
        {name: "Rectangle", shape: Rectangle{Width: 12, Height: 6}, hasArea: 72.0},
        {name: "Circle", shape: Circle{Radius: 10}, hasArea: 314.1592653589793},
        {name: "Triangle", shape: Triangle{Base: 12, Height: 6}, hasArea: 36.0},
    }

    for _, tt := range areaTests {
        // using tt.name from the case to use it as the `t.Run` test name
        t.Run(tt.name, func(t *testing.T) {
            got := tt.shape.Area()
            if got != tt.hasArea {
                t.Errorf("%#v got %.2f want %.2f", tt.shape, got, tt.hasArea)
            }
        })
    }
}
```

执行测试

```
go test -v
```

1.6 指针

测试例子

```
package main

import "math"

type Bitcoin int

type Wallet struct {
    balance Bitcoin
}

func (w *Wallet) Deposit(amount Bitcoin) {
    w.balance += amount
}

func (w *Wallet) Balance() Bitcoin {
    return w.balance
}

var InsufficientFundsError = errors.New("cannot withdraw, insufficient funds")

func (w *Wallet) Withdraw(amount Bitcoin) error {

    if amount > w.balance {
        return InsufficientFundsError
    }

    w.balance -= amount
    return nil
}
```

```
package main
```

```go
import "testing"

func TestWallet(t *testing.T) {

    t.Run("Deposit", func(t *testing.T) {
        wallet := &Wallet{}
        wallet.Deposit(Bitcoin(10))

        assertBalance(t, wallet, Bitcoin(10))
    })

    t.Run("Withdraw with funds", func(t *testing.T) {
        wallet := &Wallet{Bitcoin(20)}
        err := wallet.Withdraw(Bitcoin(10))

        assertBalance(t, wallet, Bitcoin(10))
        assertNoError(t, err)
    })

    t.Run("Withdraw insufficient funds", func(t *testing.T) {
        wallet := &Wallet{Bitcoin(20)}
        err := wallet.Withdraw(Bitcoin(100))

        assertBalance(t, wallet, Bitcoin(20))
        assertError(t, err, InsufficientFundsError)
    })
}

func assertBalance(t *testing.T, wallet *Wallet, want Bitcoin) {
    got := wallet.Balance()

    if got != want {
        t.Errorf("got %s want %s", got, want)
    }
}

func assertNoError(t *testing.T, got error) {
    if got != nil {
        t.Fatal("got an error but didnt want one")
    }
}

func assertError(t *testing.T, got error, want error) {
    if got == nil {
        t.Fatal("didn't get an error but wanted one")
    }

    if got != want {
        t.Errorf("got %s, want %s", got, want)
    }
}
```

执行测试

```
go test -v
```

1.7 Maps

```
package main

const (
  // ErrNotFound means the definition could not be found for the given word
  ErrNotFound = DictionaryErr("could not find the word you were looking for")

  // ErrWordExists means you are trying to add a word that is already known
  ErrWordExists = DictionaryErr("cannot add word because it already exists")

  // ErrWordDoesNotExist occurs when trying to update a word not in the dictionary
  ErrWordDoesNotExist = DictionaryErr("cannot update word because it does not exist")
)

// DictionaryErr are errors that can happen when interacting with the dictionary.
type DictionaryErr string

func (e DictionaryErr) Error() string {
  return string(e)
}

// Dictionary store definitions to words.
type Dictionary map[string]string

// Search find a word in the dictionary.
func (d Dictionary) Search(word string) (string, error) {
  definition, ok := d[word]
  if !ok {
    return "", ErrNotFound
  }

  return definition, nil
}

// Add inserts a word and definition into the dictionary.
func (d Dictionary) Add(word, definition string) error {
  _, err := d.Search(word)
  switch err {
  case ErrNotFound:
```

```go
		d[word] = definition
	case nil:
		return ErrWordExists
	default:
		return err

	}

	return nil
}

// Update changes the definition of a given word.
func (d Dictionary) Update(word, definition string) error {
	_, err := d.Search(word)
	switch err {
	case ErrNotFound:
		return ErrWordDoesNotExist
	case nil:
		d[word] = definition
	default:
		return err

	}

	return nil
}

// Delete removes a word from the dictionary.
func (d Dictionary) Delete(word string) {
	delete(d, word)
}
```

```go
package main

import (
	"testing"
)

func TestSearch(t *testing.T) {
	dictionary := Dictionary{"test": "this is just a test"}

	t.Run("known word", func(t *testing.T) {
		got, _ := dictionary.Search("test")
		want := "this is just a test"

		assertStrings(t, got, want)
	})

	t.Run("unknown word", func(t *testing.T) {
		_, got := dictionary.Search("unknown")

		assertError(t, got, ErrNotFound)
	})
}

func TestAdd(t *testing.T) {
	t.Run("new word", func(t *testing.T) {
		dictionary := Dictionary{}
		word := "test"
		definition := "this is just a test"

		err := dictionary.Add(word, definition)

		assertError(t, err, nil)
		assertDefinition(t, dictionary, word, definition)
	})

	t.Run("existing word", func(t *testing.T) {
		word := "test"
		definition := "this is just a test"
		dictionary := Dictionary{word: definition}
		err := dictionary.Add(word, "new test")

		assertError(t, err, ErrWordExists)
		assertDefinition(t, dictionary, word, definition)
	})
}

func TestUpdate(t *testing.T) {
	t.Run("existing word", func(t *testing.T) {
		word := "test"
		definition := "this is just a test"
		newDefinition := "new definition"
		dictionary := Dictionary{word: definition}
		err := dictionary.Update(word, newDefinition)

		assertError(t, err, nil)
		assertDefinition(t, dictionary, word, newDefinition)
	})

	t.Run("new word", func(t *testing.T) {
		word := "test"
		definition := "this is just a test"
		dictionary := Dictionary{}
```

```go
    err := dictionary.Update(word, definition)

    assertError(t, err, ErrWordDoesNotExist)
  })
}

func TestDelete(t *testing.T) {
  word := "test"
  dictionary := Dictionary{word: "test definition"}

  dictionary.Delete(word)

  _, err := dictionary.Search(word)
  if err != ErrNotFound {
    t.Errorf("Expected %q to be deleted", word)
  }
}

func assertStrings(t testing.TB, got, want string) {
  t.Helper()

  if got != want {
    t.Errorf("got %q want %q", got, want)
  }
}

func assertError(t testing.TB, got, want error) {
  t.Helper()

  if got != want {
    t.Errorf("got error %q want %q", got, want)
  }
}

func assertDefinition(t testing.TB, dictionary Dictionary, word, definition string) {
  t.Helper()

  got, err := dictionary.Search(word)
  if err != nil {
    t.Fatal("should find added word:", err)
  }

  if definition != got {
    t.Errorf("got %q want %q", got, definition)
  }
}
```

执行测试

```
go test -v
```

1.8 依赖注入

测试例子

```go
package main

import (
  "fmt"
  "io"
  "log"
  "net/http"
)

// Greet sends a personalised greeting to writer.
func Greet(writer io.Writer, name string) {
  fmt.Fprintf(writer, "Hello, %s", name)
}

// MyGreeterHandler says Hello, world over HTTP.
func MyGreeterHandler(w http.ResponseWriter, r *http.Request) {
  Greet(w, "world")
}

func main() {
  log.Fatal(http.ListenAndServe(":5000", http.HandlerFunc(MyGreeterHandler)))
}
```

```go
package main

import "testing"

func TestWallet(t *testing.T) {

    t.Run("Deposit", func(t *testing.T) {
        wallet := Wallet{}
        wallet.Deposit(Bitcoin(10))

        assertBalance(t, wallet, Bitcoin(10))
    })
    t.Run("Withdraw with funds", func(t *testing.T) {
        wallet := Wallet{Bitcoin(20)}
        err := wallet.Withdraw(Bitcoin(10))

        assertBalance(t, wallet, Bitcoin(10))
        assertNoError(t, err)
    })

    t.Run("Withdraw insufficient funds", func(t *testing.T) {
        wallet := Wallet{Bitcoin(20)}
        err := wallet.Withdraw(Bitcoin(100))

        assertBalance(t, wallet, Bitcoin(20))
        assertError(t, err, InsufficientFundsError)
    })
}

func assertBalance(t *testing.T, wallet Wallet, want Bitcoin) {
    got := wallet.Balance()

    if got != want {
        t.Errorf("got %s want %s", got, want)
    }
}

func assertNoError(t *testing.T, got error) {
    if got != nil {
        t.Fatal("got an error but didnt want one")
    }
}

func assertError(t *testing.T, got error, want error) {
    if got == nil {
        t.Fatal("didn't get an error but wanted one")
    }

    if got != want {
        t.Errorf("got %s, want %s", got, want)
    }
}
```

执行测试

```
go test -v
```

1.9 并发

测试例子

```go
package concurrency

import "net/http"

// CheckWebsite returns true if the URL returns a 200 status code, false otherwise.
func CheckWebsite(url string) bool {
  response, err := http.Head(url)
  if err != nil {
    return false
  }

  if response.StatusCode != http.StatusOK {
    return false
  }

  return true
}


// WebsiteChecker checks a url, returning a bool.
type WebsiteChecker func(string) bool
type result struct {
  string
  bool
}

// CheckWebsites takes a WebsiteChecker and a slice of urls and returns  a map.
// of urls to the result of checking each url with the WebsiteChecker function.
func CheckWebsites(wc WebsiteChecker, urls []string) map[string]bool {
  results := make(map[string]bool)
  resultChannel := make(chan result)

  for _, url := range urls {
    go func(u string) {
      resultChannel <- result{u, wc(u)}
    }(url)
  }

  for i := 0; i < len(urls); i++ {
    r := <-resultChannel
    results[r.string] = r.bool
  }
  return results
}
```

```go
package concurrency

import (
  "reflect"
  "testing"
)

func mockWebsiteChecker(url string) bool {
  if url == "waat://furhurterwe.geds" {
    return false
  }
  return true
}

func TestCheckWebsites(t *testing.T) {
  websites := []string{
    "http://google.com",
    "http://blog.gypsydave5.com",
    "waat://furhurterwe.geds",
  }

  want := map[string]bool{
    "http://google.com":          true,
    "http://blog.gypsydave5.com": true,
    "waat://furhurterwe.geds":    false,
  }

  got := CheckWebsites(mockWebsiteChecker, websites)

  if !reflect.DeepEqual(want, got) {
    t.Fatalf("Wanted %v, got %v", want, got)
  }
}
```

执行测试

```
go test -v
```

```go
package concurrency

import (
	"testing"
	"time"
)

func slowStubWebsiteChecker(_ string) bool {
	time.Sleep(20 * time.Millisecond)
	return true
}

func BenchmarkCheckWebsites(b *testing.B) {
	urls := make([]string, 100)
	for i := 0; i < len(urls); i++ {
		urls[i] = "a url"
	}

	for i := 0; i < b.N; i++ {
		CheckWebsites(slowStubWebsiteChecker, urls)
	}
}
```

```
go test -bench=.
```

1.10 Select

测试例子

```
package racer

import (
	"fmt"
	"net/http"
	"time"
)

var tenSecondTimeout = 10 * time.Second

// Racer compares the response times of a and b, returning the fastest one, timing out after 10s.
func Racer(a, b string) (winner string, error error) {
	return ConfigurableRacer(a, b, tenSecondTimeout)
}

// ConfigurableRacer compares the response times of a and b, returning the fastest one.
```

```go
func ConfigurableRacer(a, b string, timeout time.Duration) (winner string, error error)
{
  select {
  case <-ping(a):
    return a, nil
  case <-ping(b):
    return b, nil
  case <-time.After(timeout):
    return "", fmt.Errorf("timed out waiting for %s and %s", a, b)
  }
}

func ping(url string) chan struct{} {
  ch := make(chan struct{})
  go func() {
    http.Get(url)
    close(ch)
  }()
  return ch
}
```

```go
package racer

import (
  "net/http"
  "net/http/httptest"
  "testing"
  "time"
)

func TestRacer(t *testing.T) {

  t.Run("compares speeds of servers, returning the url of the fastest one", func(t
*testing.T) {
    slowServer := makeDelayedServer(20 * time.Millisecond)
    fastServer := makeDelayedServer(0 * time.Millisecond)

    defer slowServer.Close()
    defer fastServer.Close()

    slowURL := slowServer.URL
    fastURL := fastServer.URL

    want := fastURL
    got, err := Racer(slowURL, fastURL)
    if err != nil {
      t.Fatalf("did not expect an error but got one %v", err)
    }

    if got != want {
      t.Errorf("got %q, want %q", got, want)
    }
  })

  t.Run("returns an error if a server doesn't respond within 10s", func(t *testing.T) {
    server := makeDelayedServer(25 * time.Millisecond)

    defer server.Close()

    _, err := ConfigurableRacer(server.URL, server.URL, 20*time.Millisecond)

    if err == nil {
      t.Error("expected an error but didn't get one")
    }
  })
}

func makeDelayedServer(delay time.Duration) *httptest.Server {
  return httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r
*http.Request) {
    time.Sleep(delay)
    w.WriteHeader(http.StatusOK)
  }))
}
```

执行测试

```
go test -v
```

1.11 反射

测试例子

```
package main

import (
  "reflect"
)

func walk(x interface{}, fn func(input string)) {
  val := getValue(x)
  walkValue := func(value reflect.Value) {
    walk(value.Interface(), fn)
  }

  switch val.Kind() {
  case reflect.String:
    fn(val.String())
  case reflect.Struct:
    for i := 0; i < val.NumField(); i++ {
      walkValue(val.Field(i))
    }
  case reflect.Slice, reflect.Array:
    for i := 0; i < val.Len(); i++ {
      walkValue(val.Index(i))
    }
  case reflect.Map:
    for _, key := range val.MapKeys() {
      walkValue(val.MapIndex(key))
    }
  case reflect.Chan:
    for v, ok := val.Recv(); ok; v, ok = val.Recv() {
      walkValue(v)
    }
  case reflect.Func:
    valFnResult := val.Call(nil)
    for _, res := range valFnResult {
      walkValue(res)
    }
  }
}

func getValue(x interface{}) reflect.Value {
  val := reflect.ValueOf(x)

  if val.Kind() == reflect.Ptr {
    val = val.Elem()
  }

  return val
}
```

```
package main

import (
  "reflect"
```

```go
  "testing"
)

func TestWalk(t *testing.T) {

  cases := []struct {
    Name          string
    Input         interface{}
    ExpectedCalls []string
  }{
    {
      "Struct with one string field",
      struct{ Name string }{"Chris"},
      []string{"Chris"},
    },
    {
      "Struct with two string fields",
      struct {
        Name string
        City string
      }{"Chris", "London"},
      []string{"Chris", "London"},
    },
    {
      "Struct with non string field",
      struct {
        Name string
        Age  int
      }{"Chris", 33},
      []string{"Chris"},
    },
    {
      "Nested fields",
      Person{
        "Chris",
        Profile{33, "London"},
      },
      []string{"Chris", "London"},
    },
    {
      "Pointers to things",
      &Person{
        "Chris",
        Profile{33, "London"},
      },
      []string{"Chris", "London"},
    },
    {
      "Slices",
      []Profile{
        {33, "London"},
        {34, "Reykjavík"},
      },
      []string{"London", "Reykjavík"},
    },
    {
      "Arrays",
      [2]Profile{
        {33, "London"},
        {34, "Reykjavík"},
      },
      []string{"London", "Reykjavík"},
    },
  }

  for _, test := range cases {
    t.Run(test.Name, func(t *testing.T) {
      var got []string
      walk(test.Input, func(input string) {
        got = append(got, input)
      })

      if !reflect.DeepEqual(got, test.ExpectedCalls) {
        t.Errorf("got %v, want %v", got, test.ExpectedCalls)
      }
    })
  }

  t.Run("with maps", func(t *testing.T) {
    aMap := map[string]string{
      "Foo": "Bar",
      "Baz": "Boz",
    }

    var got []string
    walk(aMap, func(input string) {
      got = append(got, input)
    })

    assertContains(t, got, "Bar")
    assertContains(t, got, "Boz")
  })

  t.Run("with channels", func(t *testing.T) {
    aChannel := make(chan Profile)

    go func() {
      aChannel <- Profile{33, "Berlin"}
      aChannel <- Profile{34, "Katowice"}
      close(aChannel)
    }()

    var got []string
    want := []string{"Berlin", "Katowice"}

    walk(aChannel, func(input string) {
      got = append(got, input)
    })

    if !reflect.DeepEqual(got, want) {
      t.Errorf("got %v, want %v", got, want)
    }
  })

  t.Run("with function", func(t *testing.T) {
    aFunction := func() (Profile, Profile) {
      return Profile{33, "Berlin"}, Profile{34, "Katowice"}
    }

    var got []string
    want := []string{"Berlin", "Katowice"}

    walk(aFunction, func(input string) {
      got = append(got, input)
    })

    if !reflect.DeepEqual(got, want) {
      t.Errorf("got %v, want %v", got, want)
    }
  })
}

type Person struct {
  Name    string
  Profile Profile
}

type Profile struct {
  Age  int
  City string
}

func assertContains(t testing.TB, haystack []string, needle string) {
  t.Helper()
  contains := false
  for _, x := range haystack {
    if x == needle {
      contains = true
    }
  }
  if !contains {
    t.Errorf("expected %+v to contain %q but it didn't", haystack, needle)
  }
}
```

执行测试

```
go test -v
```

1.12 同步

测试例子

```
package v1

import "sync"

// Counter will increment a number.
type Counter struct {
  mu    sync.Mutex
  value int
}

// NewCounter returns a new Counter.
func NewCounter() *Counter {
  return &Counter{}
}

// Inc the count.
func (c *Counter) Inc() {
  c.mu.Lock()
  defer c.mu.Unlock()
  c.value++
}

// Value returns the current count.
func (c *Counter) Value() int {
  return c.value
}
```

```go
package v1

import (
  "sync"
  "testing"
)

func TestCounter(t *testing.T) {

  t.Run("incrementing the counter 3 times leaves it at 3", func(t *testing.T) {
    counter := NewCounter()
    counter.Inc()
    counter.Inc()
    counter.Inc()

    assertCounter(t, counter, 3)
  })

  t.Run("it runs safely concurrently", func(t *testing.T) {
    wantedCount := 1000
    counter := NewCounter()

    var wg sync.WaitGroup
    wg.Add(wantedCount)

    for i := 0; i < wantedCount; i++ {
      go func() {
        counter.Inc()
        wg.Done()
      }()
    }
    wg.Wait()

    assertCounter(t, counter, wantedCount)
  })

}

func assertCounter(t testing.TB, got *Counter, want int) {
  t.Helper()
  if got.Value() != want {
    t.Errorf("got %d, want %d", got.Value(), want)
  }
}
```

执行测试

```
go test -v
```

1.13 Context

测试例子

```go
package context3

import (
  "context"
  "fmt"
  "net/http"
)

// Store fetches data.
type Store interface {
  Fetch(ctx context.Context) (string, error)
}

// Server returns a handler for calling Store.
func Server(store Store) http.HandlerFunc {
  return func(w http.ResponseWriter, r *http.Request) {
    data, err := store.Fetch(r.Context())

    if err != nil {
      return // todo: log error however you like
    }

    fmt.Fprint(w, data)
  }
}
```

```go
package context3

import (
  "context"
  "net/http"
  "net/http/httptest"
  "testing"
  "time"
)

func TestServer(t *testing.T) {
  data := "hello, world"

  t.Run("returns data from store", func(t *testing.T) {
    store := &SpyStore{response: data, t: t}
    svr := Server(store)

    request := httptest.NewRequest(http.MethodGet, "/", nil)
    response := httptest.NewRecorder()

    svr.ServeHTTP(response, request)

    if response.Body.String() != data {
      t.Errorf(`got "%s", want "%s"`, response.Body.String(), data)
    }
  })

  t.Run("tells store to cancel work if request is cancelled", func(t *testing.T) {
    store := &SpyStore{response: data, t: t}
    svr := Server(store)

    request := httptest.NewRequest(http.MethodGet, "/", nil)

    cancellingCtx, cancel := context.WithCancel(request.Context())
    time.AfterFunc(5*time.Millisecond, cancel)
    request = request.WithContext(cancellingCtx)

    response := &SpyResponseWriter{}

    svr.ServeHTTP(response, request)

    if response.written {
      t.Error("a response should not have been written")
    }
  })
}
```

执行测试

```
go test -v
```

1.15 io操作

测试例子

```
package main
```

```go
import (
  "os"
)

type tape struct {
  file *os.File
}

func (t *tape) Write(p []byte) (n int, err error) {
  t.file.Truncate(0)
  t.file.Seek(0, 0)
  return t.file.Write(p)
}
```

```go
package main

import (
  "io/ioutil"
  "testing"
  "os"
)

func createTempFile(t testing.TB, initialData string) (*os.File, func()) {
  t.Helper()

  tmpfile, err := ioutil.TempFile("", "db")

  if err != nil {
    t.Fatalf("could not create temp file %v", err)
  }

  tmpfile.Write([]byte(initialData))

  removeFile := func() {
    tmpfile.Close()
    os.Remove(tmpfile.Name())
  }

  return tmpfile, removeFile
}


func TestTape_Write(t *testing.T) {
  file, clean := createTempFile(t, "12345")
  defer clean()
  tape := &tape{file}

  tape.Write([]byte("abc"))

  file.Seek(0, 0)
  newFileContents, _ := ioutil.ReadAll(file)

  got := string(newFileContents)
  want := "abc"

  if got != want {
    t.Errorf("got %q want %q", got, want)
  }
}
```

执行测试

```
go test -v
```

二、常用测试方法和工具

实际项目中，经常使用testify包帮助快速写test文件

如使用：github.com/stretchr/testify/assert 官方例子：

```
package yours

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

func TestSomething(t *testing.T) {

  // assert equality
  assert.Equal(t, 123, 123, "they should be equal")

  // assert inequality
  assert.NotEqual(t, 123, 456, "they should not be equal")

  // assert for nil (good for errors)
  assert.Nil(t, object)

  // assert for not nil (good when you expect something)
  if assert.NotNil(t, object) {

    // now we know that object isn't nil, we are safe to make
    // further assertions without causing any errors
    assert.Equal(t, "Something", object.Value)

  }
}
```

##

类似的库还有：gocheck，参考doc地址： https://godoc.org/gopkg.in/check.v1

2.1 使用table-driven方式测试

golang测试经常使用table-driven test，详细信息可参考：https://mj-go.in/golang/table-driven-tests-in-go。

以 $GOPATH/src/github.com/user/stringutil/reverse_test.go 为例说明：

```
package stringutil

import "testing"

func TestReverse(t *testing.T) {
  cases := []struct {
    in, want string
  }{
    {"Hello, world", "dlrow ,olleH"},
    {"Hello, 世界", "界世 ,olleH"},
  }
  for _, c := range cases {
    got := Reverse(c.in)
    if got != c.want {
      t.Errorf("Reverse(%q) == %q, want %q", c.in, got, c.want)
    }
  }
}
```

执行测试：

```
$ go test github.com/user/stringutil
ok    github.com/user/stringutil 0.165s
```

也可以在package目录执行：

```
$ go test
ok    github.com/user/stringutil 0.165s
```

2.2 使用Parallel并行测试

在某些场景下，需要多个协程并行执行才能做好测试。

例子1：对map加锁以实现安全读和写

```
package mymap

import "sync"

var (
    data   = make(map[string]string)
    locker sync.RWMutex
)

func WriteToMap(k, v string) {
    locker.Lock()
    defer locker.Unlock()
    data[k] = v
}

func ReadFromMap(k string) string {
    locker.RLock()
    defer locker.RUnlock()
    return data[k]
}
```

```
package mymap

import "testing"

var kvs = []struct {
    k string
    v string
}{
    {"aa1", "889893"},
    {"aa2", "Go8888"},
    {"aa3", "hello"},
    {"aa4", "world"},
    {"aa5", "new"},
    {"aa6", "one"},
}

// 注意 TestWriteToMap 需要在 TestReadFromMap 之前
func TestWriteToMap(t *testing.T) {
    t.Parallel()
    for _, tt := range kvs {
        WriteToMap(tt.k, tt.v)
    }
}

func TestReadFromMap(t *testing.T) {
    t.Parallel()
    for _, tt := range kvs {
        actual := ReadFromMap(tt.k)
        if actual != tt.v {
            t.Errorf("the value of key(%s) is %s, expected: %s", tt.k, actual, tt.v)
        }
    }
}
```

执行步骤：

1. 注释掉 WriteToMap 和 ReadFromMap 中 locker 保护的代码，同时注释掉测试代码中的 t.Parallel，执行测试，测试通过，即使加上 `-race`，测试依然通过；
2. 只注释掉 WriteToMap 和 ReadFromMap 中 locker 保护的代码，执行测试，测试失败（如果未失败，加上 `-race` 一定会失败）；

```
 go test -race -v
=== RUN   TestWriteToMap
=== PAUSE TestWriteToMap
=== RUN   TestReadFromMap
=== PAUSE TestReadFromMap
=== CONT  TestWriteToMap
=== CONT  TestReadFromMap
--- PASS: TestWriteToMap (0.00s)
--- PASS: TestReadFromMap (0.00s)
PASS
ok    github.com/smartystreets/goconvey/examples/bench/mymap  0.378s
```

2.3 使用HTTPTest测试HTTP(S)

Go 的标准库为我们提供了一个 httptest 的库，通过它就能够轻松的完成 HTTP 的测试。

```
package main

import (
    "fmt"
    "io"
    "io/ioutil"
    "net/http"
    "net/http/httptest"
)

var HandleHelloWorld = func(w http.ResponseWriter, r *http.Request) {
    io.WriteString(w, "<html><body>Hello World!</body></html>")
}

func main() {
    req := httptest.NewRequest("GET", "http://example.com/foo", nil)
    w := httptest.NewRecorder()
    HandleHelloWorld(w, req)

    resp := w.Result()
    body, _ := ioutil.ReadAll(resp.Body)

    fmt.Println(resp.StatusCode)
    fmt.Println(resp.Header.Get("Content-Type"))
    fmt.Println(string(body))
}
```

它也支持https:

```
package main

import (
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "net/http/httptest"
)

func main() {
    ts := httptest.NewTLSServer(http.HandlerFunc(func(w http.ResponseWriter, r
*http.Request) {
        fmt.Fprintln(w, "Hello, client")
    }))
    defer ts.Close()

    client := ts.Client()
    res, err := client.Get(ts.URL)
    if err != nil {
        log.Fatal(err)
    }

    greeting, err := ioutil.ReadAll(res.Body)
    res.Body.Close()
    if err != nil {
        log.Fatal(err)
    }

    fmt.Printf("%s", greeting)
}
```

2.4 使用gomock

gomock 是官方提供的 mock 框架，同时还提供了 mockgen 工具用来辅助生成测试代码。

官方sample：https://github.com/golang/mock/tree/master/sample

使用如下命令即可安装：

```
go get -u github.com/golang/mock/gomock
go get -u github.com/golang/mock/mockgen
```

2.4.1 使用gomock的例子

例子1

```
package dbo

// db.go

type DB interface {
  Get(key string) (int, error)
}

func GetFromDB(db DB, key string) int {
  if value, err := db.Get(key); err == nil {
    return value
  }

  return -1
}
```

如果此时需要测试 `GetFromDB` 这个函数内部的逻辑，就需要 mock 接口 `DB`。步骤如下：

第一步：使用 `mockgen` 生成 `db_mock.go`。一般传递三个参数。包含需要被mock的接口得到源文件`source`，生成的目标文件`destination`，包名`package`。

```
$ mockgen -source=db.go -destination=db_mock.go -package=main
```

第二步：新建 `db_test.go`，写测试用例。

```go
package dbo

import (
  "errors"
  "testing"

  "github.com/golang/mock/gomock"
)

func TestGetFromDB(t *testing.T) {
  ctrl := gomock.NewController(t)
  defer ctrl.Finish() // 断言 DB.Get() 方法是否被调用

  m := NewMockDB(ctrl)
  m.EXPECT().Get(gomock.Eq("Tom")).Return(100, errors.New("not exist"))

  if v := GetFromDB(m, "Tom"); v != -1 {
    t.Fatal("expected -1, but got", v)
  }
}
```

- 这个测试用例有2个目的，一是使用 `ctrl.Finish()` 断言 `DB.Get()` 被是否被调用，如果没有被调用，后续的 mock 就失去了意义；
- 二是测试方法 `GetFromDB()` 的逻辑是否正确(如果 `DB.Get()` 返回 error，那么 `GetFromDB()` 返回 -1)。
- `NewMockDB()` 的定义在 `db_mock.go` 中，由 mockgen 自动生成。

最终的代码结构如下：

```
dbo/
    |--db.go
    |--db_mock.go // generated by mockgen
    |--db_test.go
```

第三步： 执行测试：

```
$ go test . -cover -v
=== RUN   TestGetFromDB
--- PASS: TestGetFromDB (0.00s)
PASS
coverage: 81.2% of statements
ok      example 0.008s  coverage: 81.2% of statements
```

例子2

```go
// person/male.go
package person

type Male interface {
    Get(id int64) error
}
```

```go
// user/user.go
package user

import "github.com/EDDYCJY/mockd/person"

type User struct {
    Person person.Male
}

func NewUser(p person.Male) *User {
    return &User{Person: p}
}

func (u *User) GetUserInfo(id int64) error {
    return u.Person.Get(id)
}
```

生成 mock 文件

回到 `mockd/` 的根目录下，执行以下命令

```
$ mockgen -source=./person/male.go -destination=./mock/male_mock.go -package=mock
```

在执行完毕后，可以发现 `mock/` 目录下多出了 male_mock.go 文件，这就是 mock 文件。那么命令中的指令又分别有什么用呢？如下：

- -source：设置需要模拟（mock）的接口文件
- -destination：设置 mock 文件输出的地方，若不设置则打印到标准输出中
- -package：设置 mock 文件的包名，若不设置则为 `mock_` 前缀加上文件名（如本文的包名会为 mock_person）

```go
// Code generated by MockGen. DO NOT EDIT.
// Source: ./person/male.go

// Package mock is a generated GoMock package.
package mock

```

```go
import (
	gomock "github.com/golang/mock/gomock"
	reflect "reflect"
)

// MockMale is a mock of Male interface
type MockMale struct {
	ctrl     *gomock.Controller
	recorder *MockMaleMockRecorder
}

// MockMaleMockRecorder is the mock recorder for MockMale
type MockMaleMockRecorder struct {
	mock *MockMale
}

// NewMockMale creates a new mock instance
func NewMockMale(ctrl *gomock.Controller) *MockMale {
	mock := &MockMale{ctrl: ctrl}
	mock.recorder = &MockMaleMockRecorder{mock}
	return mock
}

// EXPECT returns an object that allows the caller to indicate expected use
func (m *MockMale) EXPECT() *MockMaleMockRecorder {
	return m.recorder
}

// Get mocks base method
func (m *MockMale) Get(id int64) error {
	ret := m.ctrl.Call(m, "Get", id)
	ret0, _ := ret[0].(error)
	return ret0
}

// Get indicates an expected call of Get
func (mr *MockMaleMockRecorder) Get(id interface{}) *gomock.Call {
	return mr.mock.ctrl.RecordCallWithMethodType(mr.mock, "Get",
reflect.TypeOf((*MockMale)(nil).Get), id)
}
```

编写test

```go
// user/user_test.go
package user

import (
	"testing"
```

```
    "github.com/EDDYCJY/mockd/mock"

    "github.com/golang/mock/gomock"
)

func TestUser_GetUserInfo(t *testing.T) {
    ctl := gomock.NewController(t)
    defer ctl.Finish()

    var id int64 = 1
    mockMale := mock.NewMockMale(ctl)
    gomock.InOrder(
        mockMale.EXPECT().Get(id).Return(nil),
    )

    user := NewUser(mockMale)
    err := user.GetUserInfo(id)
    if err != nil {
        t.Errorf("user.GetUserInfo err: %v", err)
    }
}
```

1. gomock.NewController：返回 `gomock.Controller`，它代表 mock 生态系统中的顶级控件。定义了 mock 对象的范围、生命周期和期待值。另外它在多个 goroutine 中是安全的
2. mock.NewMockMale：创建一个新的 mock 实例
3. gomock.InOrder：声明给定的调用应按顺序进行（是对 gomock.After 的二次封装）
4. mockMale.EXPECT().Get(id).Return(nil)：这里有三个步骤，`EXPECT()`返回一个允许调用者设置**期望**和**返回值**的对象。`Get(id)` 是设置入参并调用 mock 实例中的方法。`Return(nil)` 是设置先前调用的方法出参。简单来说，就是设置入参并调用，最后设置返回值
5. NewUser(mockMale)：创建 User 实例，值得注意的是，在这里**注入了 mock 对象**，因此实际在随后的 `user.GetUserInfo(id)` 调用（入参：id 为 1）中。它调用的是我们事先模拟好的 mock 方法
6. ctl.Finish()：进行 mock 用例的期望值断言，一般会使用 `defer` 延迟执行，以防止我们忘记这一操作

回到 `mockd/` 的根目录下，执行测试：

```
$ go test ./user
ok      github.com/EDDYCJY/mockd/user
```

2.4.2 gomock常用说明

gomock调用方法

- Call.Do()：声明在匹配时要运行的操作
- Call.DoAndReturn()：声明在匹配调用时要运行的操作，并且模拟返回该函数的返回值
- Call.MaxTimes()：设置最大的调用次数为 n 次

- Call.MinTimes()：设置最小的调用次数为 n 次
- Call.AnyTimes()：允许调用次数为 0 次或更多次
- Call.Times()：设置调用次数为 n 次

gomock参数匹配

- gomock.Any()：匹配任意值
- gomock.Eq()：通过反射匹配到指定的类型值，而不需要手动设置
- gomock.Nil()：返回 nil

建议更多的方法可参见 官方文档

2.5 使用GoConvey

GoConvey是一款针对Golang的测试框架，可以管理和运行测试用例，同时提供了丰富的断言函数，并支持多种Web界面特性。
官方地址：
https://github.com/smartystreets/goconvey
安装：
`go get github.com/smartystreets/goconvey`

GoConvey的特点

GoConvey支持 go test，可直接在终端窗口和浏览器上使用。GoConvey特点如下：
A、直接集成go test
B、巨大的回归测试套件
C、可读性强的色彩控制台输出
D、完全自动化的Web UI
E、测试代码生成器

2.5.1 GoConvey标准断言

官方文档：
https://github.com/smartystreets/goconvey/wiki

GoConvey自带大量的标准断言函数，可以通过So()使用。
（1）通用相等比较
So(thing1, ShouldEqual, thing2)
So(thing1, ShouldNotEqual, thing2)
So(thing1, ShouldResemble, thing2)
用于数组、切片、map和结构体的深度比较
So(thing1, ShouldNotResemble, thing2)
So(thing1, ShouldPointTo, thing2)
So(thing1, ShouldNotPointTo, thing2)
So(thing1, ShouldBeNil)
So(thing1, ShouldNotBeNil)

So(thing1, ShouldBeTrue)
So(thing1, ShouldBeFalse)
So(thing1, ShouldBeZeroValue)
（2）数值比较
So(1, ShouldBeGreaterThan, 0)
So(1, ShouldBeGreaterThanOrEqualTo, 0)
So(1, ShouldBeLessThan, 2)
So(1, ShouldBeLessThanOrEqualTo, 2)
So(1.1, ShouldBeBetween, .8, 1.2)
So(1.1, ShouldNotBeBetween, 2, 3)
So(1.1, ShouldBeBetweenOrEqual, .9, 1.1)
So(1.1, ShouldNotBeBetweenOrEqual, 1000, 2000)
So(1.0, ShouldAlmostEqual, 0.99999999, .0001)
带容差比较，默认容差为0.0000000001
So(1.0, ShouldNotAlmostEqual, 0.9, .0001)
（3）数据集合比较
So([]int{2, 4, 6}, ShouldContain, 4)
So([]int{2, 4, 6}, ShouldNotContain, 5)
So(4, ShouldBeIn, ...[]int{2, 4, 6})
So(4, ShouldNotBeIn, ...[]int{1, 3, 5})
So([]int{}, ShouldBeEmpty)
So([]int{1}, ShouldNotBeEmpty)
So(map[string]string{“a”: “b”}, ShouldContainKey, “a”)
So(map[string]string{“a”: “b”}, ShouldNotContainKey, “b”)
So(map[string]string{“a”: “b”}, ShouldNotBeEmpty)
So(map[string]string{}, ShouldBeEmpty)
So(map[string]string{“a”: “b”}, ShouldHaveLength, 1)
支持map、切片、通道、字符串
（4）字符串比较
So(“asdf”, ShouldStartWith, “as”)
So(“asdf”, ShouldNotStartWith, “df”)
So(“asdf”, ShouldEndWith, “df”)
So(“asdf”, ShouldNotEndWith, “df”)
So(“asdf”, ShouldContainSubstring, “sd”)
So(“asdf”, ShouldNotContainSubstring, “er”)
So(“adsf”, ShouldBeBlank)
So(“asdf”, ShouldNotBeBlank)
（5）异常比较
So(func(), ShouldPanic)
So(func(), ShouldNotPanic)
So(func(), ShouldPanicWith, “”) // or errors.New(“something”)
So(func(), ShouldNotPanicWith, “”) // or errors.New(“something”)
（6）类型检查
So(1, ShouldHaveSameTypeAs, 0)
So(1, ShouldNotHaveSameTypeAs, “asdf”)
（7）时间比较
So(time.Now(), ShouldHappenBefore, time.Now())

So(time.Now(), ShouldHappenOnOrBefore, time.Now())
So(time.Now(), ShouldHappenAfter, time.Now())
So(time.Now(), ShouldHappenOnOrAfter, time.Now())
So(time.Now(), ShouldHappenBetween, time.Now(), time.Now())
So(time.Now(), ShouldHappenOnOrBetween, time.Now(), time.Now())
So(time.Now(), ShouldNotHappenOnOrBetween, time.Now(), time.Now())
So(time.Now(), ShouldHappenWithin, duration, time.Now())
So(time.Now(), ShouldNotHappenWithin, duration, time.Now())

2.5.2 使用GoConvery的例子

代码如下：

```
package examples

// Game contains the state of a bowling game.
type Game struct {
  rolls   []int
  current int
}

// NewGame allocates and starts a new game of bowling.
func NewGame() *Game {
  game := new(Game)
  game.rolls = make([]int, maxThrowsPerGame)
  return game
}

// Roll rolls the ball and knocks down the number of pins specified by pins.
func (self *Game) Roll(pins int) {
  self.rolls[self.current] = pins
  self.current++
}

// Score calculates and returns the player's current score.
func (self *Game) Score() (sum int) {
  for throw, frame := 0, 0; frame < framesPerGame; frame++ {
    if self.isStrike(throw) {
      sum += self.strikeBonusFor(throw)
      throw += 1
    } else if self.isSpare(throw) {
      sum += self.spareBonusFor(throw)
      throw += 2
    } else {
      sum += self.framePointsAt(throw)
      throw += 2
    }
  }
  return sum
}

// isStrike determines if a given throw is a strike or not. A strike is knocking
// down all pins in one throw.
func (self *Game) isStrike(throw int) bool {
  return self.rolls[throw] == allPins
}

// strikeBonusFor calculates and returns the strike bonus for a throw.
func (self *Game) strikeBonusFor(throw int) int {
  return allPins + self.framePointsAt(throw+1)
}

// isSpare determines if a given frame is a spare or not. A spare is knocking
// down all pins in one frame with two throws.
func (self *Game) isSpare(throw int) bool {
  return self.framePointsAt(throw) == allPins
}

// spareBonusFor calculates and returns the spare bonus for a throw.
func (self *Game) spareBonusFor(throw int) int {
  return allPins + self.rolls[throw+2]
}

// framePointsAt computes and returns the score in a frame specified by throw.
func (self *Game) framePointsAt(throw int) int {
  return self.rolls[throw] + self.rolls[throw+1]
}

const (
  // allPins is the number of pins allocated per fresh throw.
  allPins          = 10

  // framesPerGame is the number of frames per bowling game.
  framesPerGame    = 10

  // maxThrowsPerGame is the maximum number of throws possible in a single game.
  maxThrowsPerGame = 21
)
```

```
/*
Reference: http://butunclebob.com/ArticleS.UncleBob.TheBowlingGameKata
See the very first link (which happens to be the very first word of
the first paragraph) on the page for a tutorial.
*/
```

```go
package examples

import (
  "testing"

  . "github.com/smartystreets/goconvey/convey"
)

func TestBowlingGameScoring(t *testing.T) {
  Convey("Given a fresh score card", t, func() {
    game := NewGame()

    Convey("When all gutter balls are thrown", func() {
      game.rollMany(20, 0)

      Convey("The score should be zero", func() {
        So(game.Score(), ShouldEqual, 0)
      })
    })

    Convey("When all throws knock down only one pin", func() {
      game.rollMany(20, 1)

      Convey("The score should be 20", func() {
        So(game.Score(), ShouldEqual, 20)
      })
    })

    Convey("When a spare is thrown", func() {
      game.rollSpare()
      game.Roll(3)
      game.rollMany(17, 0)

      Convey("The score should include a spare bonus.", func() {
        So(game.Score(), ShouldEqual, 16)
      })
    })

    Convey("When a strike is thrown", func() {
      game.rollStrike()
      game.Roll(3)
      game.Roll(4)
      game.rollMany(16, 0)

      Convey("The score should include a strike bonus.", func() {
        So(game.Score(), ShouldEqual, 24)
      })
    })
    Convey("When all strikes are thrown", func() {
      game.rollMany(21, 10)

      Convey("The score should be 300.", func() {
        So(game.Score(), ShouldEqual, 300)
      })
    })
  })
}

func (self *Game) rollMany(times, pins int) {
  for x := 0; x < times; x++ {
    self.Roll(pins)
  }
}
func (self *Game) rollSpare() {
  self.Roll(5)
  self.Roll(5)
}
func (self *Game) rollStrike() {
  self.Roll(10)
}
```

执行测试

GoConvey兼容Go原生的单元测试，可直接使用Go命令来执行测试。
在测试代码目录下运行go test命令：
`go test -v`
测试执行结果如下：

```
=== RUN   TestBowlingGameScoring

  Given a fresh score card
    When all gutter balls are thrown
      The score should be zero ✔
    When all throws knock down only one pin
      The score should be 20 ✔
    When a spare is thrown
      The score should include a spare bonus. ✔
    When a strike is thrown
      The score should include a strike bonus. ✔
    When all strikes are thrown
      The score should be 300. ✔


5 total assertions

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

```
ok    github.com/smartystreets/goconvey/examples/sub  0.364s
```

2.5.3 使用web界面查看

查看goconvey用法

```
goconvey -h
-cover: 开启覆盖率统计功能
-depth int: 扫描目录的深度，-1: 扫描无穷深度，0: 扫描当前目录
-excludedDirs string: 将某些目录排除在扫描外，多个目录使用逗号分割
-host string: 指定开启HTTP服务的主机
-launchBrowser: 触发自动开启浏览器，默认为true
-port int: 指定HTTP服务的端口
-workDir string: 指定工作目录，默认为当前目录
```

在测试用例源码目录下运行goconvey：

```
goconvey -port 8081
```

在浏览器打开：
http://localhost:8081

三. 其他测试

3.1 Benchmark测试

go-mock是专门为go语言开发的mock库，该库使用方式简单，

基准测试函数和普通测试函数写法类似，但是以Benchmark为前缀名，并且带有一个*testing.B类型的参数；* testing.B参数除了提供和*testing.T类似的方法，还有额外一些和性能测量相关的方法。

它还提供了一个整数N，用于指定操作执行的循环次数。以下面例子说明：

```
package mymap

import "sync"

var (
    data   = make(map[string]string)
    locker sync.RWMutex
)

func WriteToMap(k, v string) {
    locker.Lock()
    defer locker.Unlock()
    data[k] = v
}
```

```
func ReadFromMap(k string) string {
    locker.RLock()
    defer locker.RUnlock()
    return data[k]
}
```

```
package mymap

import (
    "testing"
)

func BenchmarkWriteToMap(b *testing.B) {
    key := "mytest"
    value := "800"

    for i := 0; i < b.N; i++ { //use b.N for looping
        WriteToMap(key, value)
    }
}

func BenchmarkReadFromMap(b *testing.B) {
    b.StopTimer() //调用该函数停止压力测试的时间计数

    //做一些初始化的工作,例如读取文件数据,数据库连接之类的,
    key := "mytest"
    value := "800"
    WriteToMap(key, value)

    b.StartTimer() //重新开始时间
    for i := 0; i < b.N; i++ {
        ReadFromMap(key)
    }
}
```

默认情况下，每个基准测试最少运行 1 秒。如果基准测试函数返回时，还不到 1 秒钟，`b.N` 的值会按照序列 1,2,5,10,20,50,... 增加，同时再次运行基准测测试函数。执行测试，

```
go test -bench=.
goos: darwin
goarch: amd64
pkg: github.com/smartystreets/goconvey/examples/bench/mymap
cpu: Intel(R) Core(TM) i5-1038NG7 CPU @ 2.00GHz
BenchmarkWriteToMap-8     29821398          39.66 ns/op
BenchmarkReadFromMap-8    43221535          27.26 ns/op
PASS
```

```
ok      github.com/smartystreets/goconvey/examples/bench/mymap  3.407s

go test -bench=BenchmarkWriteToMap  -benchtime=5s
goos: darwin
goarch: amd64
pkg: github.com/smartystreets/goconvey/examples/bench/mymap
cpu: Intel(R) Core(TM) i5-1038NG7 CPU @ 2.00GHz
BenchmarkWriteToMap-8     156710461         40.09 ns/op
PASS
ok      github.com/smartystreets/goconvey/examples/bench/mymap  55.938s
```

B类型

B 是传递给基准测试函数的一种类型，它用于管理基准测试的计时行为，并指示应该迭代地运行测试多少次。

当基准测试函数返回时，或者当基准测试函数调用 `FailNow`、`Fatal`、`Fatalf`、`SkipNow`、`Skip`、`Skipf` 中的任意一个方法时，则宣告测试函数结束。至于其他报告方法，比如 `Log` 和 `Error` 的变种，则可以在其他 goroutine 中同时进行调用。

跟单元测试一样，基准测试会在执行的过程中积累日志，并在测试完毕时将日志转储到标准错误。但跟单元测试不一样的是，为了避免基准测试的结果受到日志打印操作的影响，基准测试总是会把日志打印出来。

B 类型中的报告方法使用方式和 T 类型是一样的，一般来说，基准测试中也不需要使用，毕竟主要是测性能。这里我们对 B 类型中其他的一些方法进行讲解。

3.1.1 计时方法

有三个方法用于计时：

1. StartTimer：开始对测试进行计时。该方法会在基准测试开始时自动被调用，我们也可以在调用 StopTimer 之后恢复计时；
2. StopTimer：停止对测试进行计时。当你需要执行一些复杂的初始化操作，并且你不想对这些操作进行测量时，就可以使用这个方法来暂时地停止计时；
3. ResetTimer：对已经逝去的基准测试时间以及内存分配计数器进行清零。对于正在运行中的计时器，这个方法不会产生任何效果。本节开头有使用示例。

3.1.2 并行执行

通过 `RunParallel` 方法能够并行地执行给定的基准测试。`RunParallel`会创建出多个 goroutine，并将 b.N 分配给这些 goroutine 执行，其中 goroutine 数量的默认值为 GOMAXPROCS。用户如果想要增加非 CPU 受限（non-CPU-bound）基准测试的并行性，那么可以在 `RunParallel` 之前调用 `SetParallelism` （如 `SetParallelism(2)`，则 goroutine 数量为 2*GOMAXPROCS） 。`RunParallel` 通常会与 `-cpu` 标志一同使用。

`body` 函数将在每个 goroutine 中执行，这个函数需要设置所有 goroutine 本地的状态，并迭代直到 `pb.Next` 返回 false 值为止。因为 `StartTimer`、`StopTime` 和 `ResetTimer` 这三个方法都带有全局作用，所以 `body` 函数不应该调用这些方法； 除此之外，`body` 函数也不应该调用 `Run` 方法。

3.1.3 内存统计

`ReportAllocs` 方法用于打开当前基准测试的内存统计功能， 与 `go test` 使用 `-benchmem` 标志类似，但 `ReportAllocs` 只影响那些调用了该函数的基准测试。

测试示例：

```
func BenchmarkTmplExucte(b *testing.B) {
    b.ReportAllocs()
    templ := template.Must(template.New("test").Parse("Hello, {{.}}!"))
    b.RunParallel(func(pb *testing.PB) {
        // Each goroutine has its own bytes.Buffer.
        var buf bytes.Buffer
        for pb.Next() {
            // The loop body is executed b.N times total across all goroutines.
            buf.Reset()
            templ.Execute(&buf, "World")
        }
    })
}
```

测试结果类似这样：

```
BenchmarkTmplExucte-4          2000000              898 ns/op             368 B/op               9
allocs/
```

对上述结果中的每一项，你是否都清楚是什么意思呢？

- `2000000` ： 基准测试的迭代总次数 b.N
- `898 ns/op`： 平均每次迭代所消耗的纳秒数
- `368 B/op`： 平均每次迭代内存所分配的字节数
- `9 allocs/op`： 平均每次迭代的内存分配次数

`testing` 包中的 `BenchmarkResult` 类型能为你提供帮助，它保存了基准测试的结果，定义如下：

```
type BenchmarkResult struct {
    N         int           // The number of iterations. 基准测试的迭代总次数，即 b.N
    T         time.Duration // The total time taken. 基准测试的总耗时
    Bytes     int64         // Bytes processed in one iteration. 一次迭代处理的字节数，通过
b.SetBytes 设置
    MemAllocs uint64        // The total number of memory allocations. 内存分配的总次数
    MemBytes  uint64        // The total number of bytes allocated. 内存分配的总字节数
}
```

3.2 Example测试

`testing` 包除了测试，还提供了运行并验证Example的功能。

一个示例的例子如下：

```
func ExampleHello() {
    fmt.Println("Hello")
    // Output: Hello
}
```

如果 `Output: Hello` 改为：`Output: hello`，运行测试会失败，提示：

```
got:
Hello
want:
hello
```

一个示例函数以 Example 开头，如果示例函数包含以 "Output:" 开头的行注释，在运行测试时，go 会将示例函数的输出和 "Output:" 注释中的值做比较，就如上面的例子。

有时候，输出顺序可能不确定，比如循环输出 map 的值，那么可以使用 "Unordered output:" 开头的注释。

如果Example函数没有上述输出注释，该函数只会被编译而不会被运行。

3.2.1 命名约定

Go 语言通过大量的命名约定来简化工具的复杂度，规范代码的风格。对示例函数的命名有如下约定：

- 包级别的示例函数，直接命名为 `func Example() { ... }`
- 函数 F 的示例，命名为 `func ExampleF() { ... }`
- 类型 T 的示例，命名为 `func ExampleT() { ... }`
- 类型 T 上的 方法 M 的示例，命名为 `func ExampleT_M() { ... }`

有时，我们想要给 包 / 类型 / 函数 / 方法 提供多个示例，可以通过在示例函数名称后附加一个不同的后缀来实现，但这种后缀必须以小写字母开头，如：

```
func Example_suffix() { ... }
func ExampleF_suffix() { ... }
func ExampleT_suffix() { ... }
func ExampleT_M_suffix() { ... }
```

通常，示例代码会放在单独的示例文件中，命名为 `example_test.go`。可以查看 `io` 包中的 `example_test.go` 了解示例的编写。

3.2.2 测试例子

本节开头提到了示例的两个作用，它们分别是由 `godoc` 和 `go test` 这两个命令实现的。

在执行 `go test` 时，会运行示例。具体的实现原理，可以通过阅读 `go test` 命令源码和 `testing` 包中 `example.go` 文件了解。

```go
// Copyright 2017 The Go Authors. All rights reserved.
// Use of this source code is governed by a BSD-style
// license that can be found in the LICENSE file.

package fmt_test

import (
  "fmt"
  "io"
  "math"
  "os"
  "strings"
  "time"
)

// The Errorf function lets us use formatting features
// to create descriptive error messages.
func ExampleErrorf() {
  const name, id = "bueller", 17
  err := fmt.Errorf("user %q (id %d) not found", name, id)
  fmt.Println(err.Error())

  // Output: user "bueller" (id 17) not found
}

func ExampleFscanf() {
  var (
    i int
    b bool
    s string
  )
  r := strings.NewReader("5 true gophers")
  n, err := fmt.Fscanf(r, "%d %t %s", &i, &b, &s)
  if err != nil {
    fmt.Fprintf(os.Stderr, "Fscanf: %v\n", err)
  }
  fmt.Println(i, b, s)
  fmt.Println(n)
  // Output:
  // 5 true gophers
  // 3
}
```

```go
func ExampleFscanln() {
  s := `dmr 1771 1.61803398875
  ken 271828 3.14159`
  r := strings.NewReader(s)
  var a string
  var b int
  var c float64
  for {
    n, err := fmt.Fscanln(r, &a, &b, &c)
    if err == io.EOF {
      break
    }
    if err != nil {
      panic(err)
    }
    fmt.Printf("%d: %s, %d, %f\n", n, a, b, c)
  }
  // Output:
  // 3: dmr, 1771, 1.618034
  // 3: ken, 271828, 3.141590
}

func ExampleSscanf() {
  var name string
  var age int
  n, err := fmt.Sscanf("Kim is 22 years old", "%s is %d years old", &name, &age)
  if err != nil {
    panic(err)
  }
  fmt.Printf("%d: %s, %d\n", n, name, age)

  // Output:
  // 2: Kim, 22
}

func ExamplePrint() {
  const name, age = "Kim", 22
  fmt.Print(name, " is ", age, " years old.\n")

  // It is conventional not to worry about any
  // error returned by Print.

  // Output:
  // Kim is 22 years old.
}

func ExamplePrintln() {
  const name, age = "Kim", 22
  fmt.Println(name, "is", age, "years old.")
```

```go
	// It is conventional not to worry about any
	// error returned by Println.

	// Output:
	// Kim is 22 years old.
}

func ExamplePrintf() {
	const name, age = "Kim", 22
	fmt.Printf("%s is %d years old.\n", name, age)

	// It is conventional not to worry about any
	// error returned by Printf.

	// Output:
	// Kim is 22 years old.
}

func ExampleSprint() {
	const name, age = "Kim", 22
	s := fmt.Sprint(name, " is ", age, " years old.\n")

	io.WriteString(os.Stdout, s) // Ignoring error for simplicity.

	// Output:
	// Kim is 22 years old.
}

func ExampleSprintln() {
	const name, age = "Kim", 22
	s := fmt.Sprintln(name, "is", age, "years old.")

	io.WriteString(os.Stdout, s) // Ignoring error for simplicity.

	// Output:
	// Kim is 22 years old.
}

func ExampleSprintf() {
	const name, age = "Kim", 22
	s := fmt.Sprintf("%s is %d years old.\n", name, age)

	io.WriteString(os.Stdout, s) // Ignoring error for simplicity.

	// Output:
	// Kim is 22 years old.
}
```

```go
func ExampleFprint() {
  const name, age = "Kim", 22
  n, err := fmt.Fprint(os.Stdout, name, " is ", age, " years old.\n")

  // The n and err return values from Fprint are
  // those returned by the underlying io.Writer.
  if err != nil {
    fmt.Fprintf(os.Stderr, "Fprint: %v\n", err)
  }
  fmt.Print(n, " bytes written.\n")

  // Output:
  // Kim is 22 years old.
  // 21 bytes written.
}

func ExampleFprintln() {
  const name, age = "Kim", 22
  n, err := fmt.Fprintln(os.Stdout, name, "is", age, "years old.")

  // The n and err return values from Fprintln are
  // those returned by the underlying io.Writer.
  if err != nil {
    fmt.Fprintf(os.Stderr, "Fprintln: %v\n", err)
  }
  fmt.Println(n, "bytes written.")

  // Output:
  // Kim is 22 years old.
  // 21 bytes written.
}

func ExampleFprintf() {
  const name, age = "Kim", 22
  n, err := fmt.Fprintf(os.Stdout, "%s is %d years old.\n", name, age)

  // The n and err return values from Fprintf are
  // those returned by the underlying io.Writer.
  if err != nil {
    fmt.Fprintf(os.Stderr, "Fprintf: %v\n", err)
  }
  fmt.Printf("%d bytes written.\n", n)

  // Output:
  // Kim is 22 years old.
  // 21 bytes written.
}

// Print, Println, and Printf lay out their arguments differently. In this example
// we can compare their behaviors. Println always adds blanks between the items it
// prints, while Print adds blanks only between non-string arguments and Printf
// does exactly what it is told.
// Sprint, Sprintln, Sprintf, Fprint, Fprintln, and Fprintf behave the same as
// their corresponding Print, Println, and Printf functions shown here.
func Example_printers() {
  a, b := 3.0, 4.0
  h := math.Hypot(a, b)

  // Print inserts blanks between arguments when neither is a string.
  // It does not add a newline to the output, so we add one explicitly.
  fmt.Print("The vector (", a, b, ") has length ", h, ".\n")

  // Println always inserts spaces between its arguments,
  // so it cannot be used to produce the same output as Print in this case;
  // its output has extra spaces.
  // Also, Println always adds a newline to the output.
  fmt.Println("The vector (", a, b, ") has length", h, ".")

  // Printf provides complete control but is more complex to use.
  // It does not add a newline to the output, so we add one explicitly
  // at the end of the format specifier string.
  fmt.Printf("The vector (%g %g) has length %g.\n", a, b, h)

  // Output:
  // The vector (3 4) has length 5.
  // The vector ( 3 4 ) has length 5 .
  // The vector (3 4) has length 5.
}

// These examples demonstrate the basics of printing using a format string. Printf,
// Sprintf, and Fprintf all take a format string that specifies how to format the
// subsequent arguments. For example, %d (we call that a 'verb') says to print the
// corresponding argument, which must be an integer (or something containing an
// integer, such as a slice of ints) in decimal. The verb %v ('v' for 'value')
// always formats the argument in its default form, just how Print or Println would
// show it. The special verb %T ('T' for 'Type') prints the type of the argument
// rather than its value. The examples are not exhaustive; see the package comment
// for all the details.
func Example_formats() {
  // A basic set of examples showing that %v is the default format, in this
  // case decimal for integers, which can be explicitly requested with %d;
  // the output is just what Println generates.
  integer := 23
  // Each of these prints "23" (without the quotes).
  fmt.Println(integer)
  fmt.Printf("%v\n", integer)
  fmt.Printf("%d\n", integer)
```

```go
// The special verb %T shows the type of an item rather than its value.
fmt.Printf("%T %T\n", integer, &integer)
// Result: int *int

// Println(x) is the same as Printf("%v\n", x) so we will use only Printf
// in the following examples. Each one demonstrates how to format values of
// a particular type, such as integers or strings. We start each format
// string with %v to show the default output and follow that with one or
// more custom formats.

// Booleans print as "true" or "false" with %v or %t.
truth := true
fmt.Printf("%v %t\n", truth, truth)
// Result: true true

// Integers print as decimals with %v and %d,
// or in hex with %x, octal with %o, or binary with %b.
answer := 42
fmt.Printf("%v %d %x %o %b\n", answer, answer, answer, answer, answer)
// Result: 42 42 2a 52 101010

// Floats have multiple formats: %v and %g print a compact representation,
// while %f prints a decimal point and %e uses exponential notation. The
// format %6.2f used here shows how to set the width and precision to
// control the appearance of a floating-point value. In this instance, 6 is
// the total width of the printed text for the value (note the extra spaces
// in the output) and 2 is the number of decimal places to show.
pi := math.Pi
fmt.Printf("%v %g %.2f (%6.2f) %e\n", pi, pi, pi, pi, pi)
// Result: 3.141592653589793 3.141592653589793 3.14 (  3.14) 3.141593e+00

// Complex numbers format as parenthesized pairs of floats, with an 'i'
// after the imaginary part.
point := 110.7 + 22.5i
fmt.Printf("%v %g %.2f %.2e\n", point, point, point, point)
// Result: (110.7+22.5i) (110.7+22.5i) (110.70+22.50i) (1.11e+02+2.25e+01i)

// Runes are integers but when printed with %c show the character with that
// Unicode value. The %q verb shows them as quoted characters, %U as a
// hex Unicode code point, and %#U as both a code point and a quoted
// printable form if the rune is printable.
smile := '😀'
fmt.Printf("%v %d %c %q %U %#U\n", smile, smile, smile, smile, smile, smile)
// Result: 128512 128512 😀 '😀' U+1F600 U+1F600 '😀'

// Strings are formatted with %v and %s as-is, with %q as quoted strings,
// and %#q as backquoted strings.
placeholders := `foo "bar"`
fmt.Printf("%v %s %q %#q\n", placeholders, placeholders, placeholders, placeholders)
```

```
  // Result: foo "bar" foo "bar" "foo \"bar\"" `foo "bar"`

  // Maps formatted with %v show keys and values in their default formats.
  // The %#v form (the # is called a "flag" in this context) shows the map in
  // the Go source format. Maps are printed in a consistent order, sorted
  // by the values of the keys.
  isLegume := map[string]bool{
    "peanut":    true,
    "dachshund": false,
  }
  fmt.Printf("%v %#v\n", isLegume, isLegume)
  // Result: map[dachshund:false peanut:true] map[string]bool{"dachshund":false,
"peanut":true}

  // Structs formatted with %v show field values in their default formats.
  // The %+v form shows the fields by name, while %#v formats the struct in
  // Go source format.
  person := struct {
    Name string
    Age  int
  }{"Kim", 22}
  fmt.Printf("%v %+v %#v\n", person, person, person)
  // Result: {Kim 22} {Name:Kim Age:22} struct { Name string; Age int }{Name:"Kim",
Age:22}

  // The default format for a pointer shows the underlying value preceded by
  // an ampersand. The %p verb prints the pointer value in hex. We use a
  // typed nil for the argument to %p here because the value of any non-nil
  // pointer would change from run to run; run the commented-out Printf
  // call yourself to see.
  pointer := &person
  fmt.Printf("%v %p\n", pointer, (*int)(nil))
  // Result: &{Kim 22} 0x0
  // fmt.Printf("%v %p\n", pointer, pointer)
  // Result: &{Kim 22} 0x010203 // See comment above.

  // Arrays and slices are formatted by applying the format to each element.
  greats := [5]string{"Kitano", "Kobayashi", "Kurosawa", "Miyazaki", "Ozu"}
  fmt.Printf("%v %q\n", greats, greats)
  // Result: [Kitano Kobayashi Kurosawa Miyazaki Ozu] ["Kitano" "Kobayashi" "Kurosawa"
"Miyazaki" "Ozu"]

  kGreats := greats[:3]
  fmt.Printf("%v %q %#v\n", kGreats, kGreats, kGreats)
  // Result: [Kitano Kobayashi Kurosawa] ["Kitano" "Kobayashi" "Kurosawa"]
[]string{"Kitano", "Kobayashi", "Kurosawa"}

  // Byte slices are special. Integer verbs like %d print the elements in
  // that format. The %s and %q forms treat the slice like a string. The %x
```

```
	// verb has a special form with the space flag that puts a space between
	// the bytes.
	cmd := []byte("a⌘")
	fmt.Printf("%v %d %s %q %x % x\n", cmd, cmd, cmd, cmd, cmd, cmd)
	// Result: [97 226 140 152] [97 226 140 152] a⌘ "a⌘" 61e28c98 61 e2 8c 98

	// Types that implement Stringer are printed the same as strings. Because
	// Stringers return a string, we can print them using a string-specific
	// verb such as %q.
	now := time.Unix(123456789, 0).UTC() // time.Time implements fmt.Stringer.
	fmt.Printf("%v %q\n", now, now)
	// Result: 1973-11-29 21:33:09 +0000 UTC "1973-11-29 21:33:09 +0000 UTC"

	// Output:
	// 23
	// 23
	// 23
	// int *int
	// true true
	// 42 42 2a 52 101010
	// 3.141592653589793 3.141592653589793 3.14 (  3.14) 3.141593e+00
	// (110.7+22.5i) (110.7+22.5i) (110.70+22.50i) (1.11e+02+2.25e+01i)
	// 128512 128512 😀 '😀' U+1F600 U+1F600 '😀'
	// foo "bar" foo "bar" "foo \"bar\"" `foo "bar"`
	// map[dachshund:false peanut:true] map[string]bool{"dachshund":false, "peanut":true}
	// {Kim 22} {Name:Kim Age:22} struct { Name string; Age int }{Name:"Kim", Age:22}
	// &{Kim 22} 0x0
	// [Kitano Kobayashi Kurosawa Miyazaki Ozu] ["Kitano" "Kobayashi" "Kurosawa"
"Miyazaki" "Ozu"]
	// [Kitano Kobayashi Kurosawa] ["Kitano" "Kobayashi" "Kurosawa"] []string{"Kitano",
"Kobayashi", "Kurosawa"}
	// [97 226 140 152] [97 226 140 152] a⌘ "a⌘" 61e28c98 61 e2 8c 98
	// 1973-11-29 21:33:09 +0000 UTC "1973-11-29 21:33:09 +0000 UTC"
}
```

3.3 TestMain测试

在写测试时，有时需要在测试之前或之后进行额外的设置（setup）或拆卸（teardown）；有时，测试还需要控制在主线程上运行的代码。为了支持这些需求， `testing` 包提供了 `TestMain` 函数：

```
func TestMain(m *testing.M)
```

如果测试文件中包含该函数，那么生成的测试将调用 `TestMain(m)`，而不是直接运行测试。`TestMain` 运行在主 goroutine 中，可以在调用 `m.Run` 前后做任何设置和拆卸。注意，在 `TestMain` 函数的最后，应该使用 `m.Run` 的返回值作为参数去调用 `os.Exit`。

另外，在调用 `TestMain` 时，`flag.Parse` 并没有被调用。所以，如果 `TestMain` 依赖于 command-line 标志（包括 `testing` 包的标志），则应该显式地调用 `flag.Parse`。注意，这里的依赖是指，若 `TestMain` 函数内需要用到 command-line 标志，则必须显式地调用 `flag.Parse`，否则不需要，因为 `m.Run` 中调用 `flag.Parse`。

一个包含 `TestMain` 的例子如下：

```go
package mytestmain

import (
    "flag"
    "fmt"
    "os"
    "testing"
)

var db struct {
    Dns string
}

func TestMain(m *testing.M) {
    db.Dns = os.Getenv("DATABASE_DNS")
    if db.Dns == "" {
        db.Dns = "root:123456@tcp(localhost:3306)/?
charset=utf8&parseTime=True&loc=Local"
    }

    flag.Parse()
    exitCode := m.Run()

    db.Dns = ""

    // 退出
    os.Exit(exitCode)
}

func TestDatabase(t *testing.T) {
    fmt.Println(db.Dns)
}
```

四. 参考

learn-go-with-tests

gomock

goconvey

测试

Unit-Testing-http-client-in-Go

GoConvey框架使用指南

GoMock框架使用指南