

Go Mock (gomock)简明教程

Go 简明教程系列文章链接:

- [Go 语言简明教程](#) (Aug 6, 2019)
- [Go Gin 简明教程](#) (Aug 7, 2019)
- [Go2 新特性简明教程](#) (Aug 15, 2019)
- [Go Protobuf 简明教程](#) (Jan 11, 2020)
- [Go RPC & TLS 鉴权简明教程](#) (Jan 13, 2020)
- [Go WebAssembly \(Wasm\) 简明教程](#) (Jan 23, 2020)
- [Go Test 单元测试简明教程](#) (Feb 10, 2020)
- [Go Mock \(gomock\)简明教程](#) (Feb 14, 2020)
- [Go Mmap 文件内存映射简明教程](#) (Apr 20, 2020)
- [Go Context 并发编程简明教程](#) (Apr 20, 2020)



1 gomock 简介

上一篇文章 [Go Test 单元测试简明教程](#) 介绍了 Go 语言中单元测试的常用方法，包括子测试(subtests)、表格驱动测试(table-driven tests)、帮助函数(helpers)、网络测试和基准测试(Benchmark)等。这篇文章介绍一种新的测试方法，mock/stub 测试，当待测试的函数/对象的依赖关系很复杂，并且有些依赖不能直接创建，例如数据库连接、文件I/O等。这种场景就非常适合使用 mock/stub 测试。简单来说，就是用 mock 对象模拟依赖项的行为。

GoMock is a mocking framework for the Go programming language. It integrates well with Go's built-in testing package, but can be used in other contexts too.

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。

使用如下命令即可安装：

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

2 一个简单的 Demo

```
1 // db.go
2 type DB interface {
3     Get(key string) (int, error)
4 }
5
6 func GetFromDB(db DB, key string) int {
7     if value, err := db.Get(key); err == nil {
8         return value
9     }
10
11     return -1
12 }
```

假设 **DB** 是代码中负责与数据库交互的部分(在这里用 **map** 模拟)，测试用例中不能创建真实的数据库连接。这个时候，如果我们需要测试 **GetFromDB** 这个函数内部的逻辑，就需要 **mock** 接口 **DB**。

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**。

```
1 $ mockgen -source=db.go -destination=db_mock.go -package=main
```

第二步：新建 **db_test.go**，写测试用例。

```
1 func TestGetFromDB(t *testing.T) {
2     ctrl := gomock.NewController(t)
3     defer ctrl.Finish() // 断言 DB.Get() 方法是否被调用
4
5     m := NewMockDB(ctrl)
6     m.EXPECT().Get(gomock.Eq("Tom")).Return(100, errors.New("not exist"))
7
8     if v := GetFromDB(m, "Tom"); v != -1 {
9         t.Fatal("expected -1, but got", v)
10     }
11 }
```

```

10     }
11 }

```

- 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：

```

1  project/
2      |--db.go
3      |--db_mock.go // generated by mockgen
4      |--db_test.go

```

执行测试：

```

1  $ go test . -cover -v
2  === RUN   TestGetFromDB
3  --- PASS: TestGetFromDB (0.00s)
4  PASS
5  coverage: 81.2% of statements
6  ok      example 0.008s coverage: 81.2% of statements

```

3 打桩(stubs)

在上面的例子中，当 `Get()` 的参数为 `Tom`，则返回 `error`，这称之为 打桩(stub)，有明确的参数和返回值是最简单打桩方式。除此之外，检测调用次数、调用顺序，动态设置返回值等方式也经常使用。

3.1 参数(Eq, Any, Not, Nil)

```

1  m.EXPECT().Get(gomock.Eq("Tom")).Return(0, errors.New("not exist"))
2  m.EXPECT().Get(gomock.Any()).Return(630, nil)
3  m.EXPECT().Get(gomock.Not("Sam")).Return(0, nil)
4  m.EXPECT().Get(gomock.Nil()).Return(0, errors.New("nil"))

```

- `Eq(value)` 表示与 `value` 等价的值。
- `Any()` 可以用来表示任意的入参。
- `Not(value)` 用来表示非 `value` 以外的值。
- `Nil()` 表示 `None` 值

3.2 返回值(Return, DoAndReturn)

```
1  m.EXPECT().Get(gomock.Not("Sam")).Return(0, nil)
2  m.EXPECT().Get(gomock.Any()).Do(func(key string) {
3      t.Log(key)
4  })
5  m.EXPECT().Get(gomock.Any()).DoAndReturn(func(key string) (int, error) {
6      if key == "Sam" {
7          return 630, nil
8      }
9      return 0, errors.New("not exist")
10 })
```

- Return 返回确定的值
- Do Mock 方法被调用时，要执行的操作吗，忽略返回值。
- DoAndReturn 可以动态地控制返回值。

3.3 调用次数(Times)

```
1  func TestGetFromDB(t *testing.T) {
2      ctrl := gomock.NewController(t)
3      defer ctrl.Finish()
4
5      m := NewMockDB(ctrl)
6      m.EXPECT().Get(gomock.Not("Sam")).Return(0, nil).Times(2)
7      GetFromDB(m, "ABC")
8      GetFromDB(m, "DEF")
9  }
```

- Times() 断言 Mock 方法被调用的次数。
- MaxTimes() 最大次数。
- MinTimes() 最小次数。
- AnyTimes() 任意次数 (包括 0 次) 。

3.4 调用顺序(InOrder)

```
1  func TestGetFromDB(t *testing.T) {
2      ctrl := gomock.NewController(t)
3      defer ctrl.Finish() // 断言 DB.Get() 方法是否被调用
4
5      m := NewMockDB(ctrl)
6      o1 := m.EXPECT().Get(gomock.Eq("Tom")).Return(0, errors.New("not exist"))
7      o2 := m.EXPECT().Get(gomock.Eq("Sam")).Return(630, nil)
8      gomock.InOrder(o1, o2)
9      GetFromDB(m, "Tom")
10 }
```

```
10     GetFromDB(m, "Sam")
11 }
```

4 如何编写可 mock 的代码

写可测试的代码与写好测试用例是同等重要的，如何写可 mock 的代码呢？

- mock 作用的是接口，因此将依赖抽象为接口，而不是直接依赖具体的类。
- 不直接依赖的实例，而是使用依赖注入降低耦合性。

在软件工程中，依赖注入的意思为，给予调用方它所需要的事物。“依赖”是指可被方法调用的事物。依赖注入形式下，调用方不再直接指使用“依赖”，取而代之是“注入”。“注入”是指将“依赖”传递给调用方的过程。在“注入”之后，调用方才会调用该“依赖”。传递依赖给调用方，而不是让调用方直接获得依赖，这个是该设计的根本需求。

– 依赖注入 - Wikipedia

如果 `GetFromDB()` 方法长这个样子

```
1 func GetFromDB(key string) int {
2     db := NewDB()
3     if value, err := db.Get(key); err == nil {
4         return value
5     }
6
7     return -1
8 }
```

对 DB 接口的 mock 并不能作用于 `GetFromDB()` 内部，这样写是没办法进行测试的。那如果将接口 `db DB` 通过参数传递到 `GetFromDB()`，那么就可以轻而易举地传入 Mock 对象了。

专题: [Go 简明教程](#)

本文发表于 2020-02-14，最后修改于 2021-02-06。

本站永久域名「geektutu.com」，也可搜索「极客兔兔」找到我。

[上一篇](#) « 动手写分布式缓存 - GeeCache第三天 HTTP 服务端

[下一篇](#) » 动手写分布式缓存 - GeeCache第四天 一致性哈希(hash)

赞赏支持



推荐阅读

动手写ORM框架 - GeeORM第三天 记录新增和查询

发表于2020-03-08, 阅读约33分钟

Go WebAssembly (Wasm) 简明教程

发表于2020-01-23, 阅读约30分钟

Go Protobuf 简明教程

发表于2020-01-11, 阅读约26分钟


#关于我 (9) #Go (48) #百宝箱 (2) #Cheat Sheet (1) #Go语言高性能编程 (20) #友链 (1) #Pandas (3)
#机器学习 (9) #TensorFlow (9) #mnist (5) #Python (10) #强化学习 (3) #OpenAI gym (4) #DQN (1)
#Q-Learning (1) #CNN (1) #TensorFlow 2 (10) #官方文档 (10) #Rust (1)



Gitalk 加载中 ...

如何退出协程 goroutine (超时场景)

2 评论 • 7天前

 **zeromake** —— 还有一个技巧对于chan只需要做一次信号传递, 且没有数据传输可以用 ``make(chan struct{})`` 传递时可以用

Go语言动手写Web框架 - Gee第四天 分组控制Group

24 评论 • 3天前

feixintianxia —— @geektutu @sunanzhi 这个就看如何去设计了, 前缀区分一般是比较好的方


动手写分布式缓存 - GeeCache第二天 单机并发缓存

32 评论 • 12小时前

feixintianxia —— @geektutu
@yangzhuangqiu @walkmiao 保存时复制一

动手写分布式缓存 - GeeCache第五天 分布式节点


30 评论 • 15天前


 **FinaLone** —— @wanboyan 我怎么感觉这个代码有点问题, 因为节点之间可能会循环请

Gitalk Plus



© 2021 - 极客兔兔 - 沪ICP备18001798号-1

Powered by [Hexo](#) | Theme [Geektutu](#)  [Star](#)

 996641  10917