testing 提供对 Go 包的自动化测试的支持，包括单元测试（testing.T）和基准测试（testing.B）,本文主要探讨单元测试 。通过 `go test` 命令，能够自动执行如下形式的任何函数：

```
func TestXxx(*testing.T)
```

其中 Xxx 可以是任何字母数字字符串（但第一个字母不能是 [a-z]），用于识别测试例程。要编写一个新的测试套件，需要创建一个名称以 _test.go 结尾的文件，该文件包含 `TestXxx` 函数，如上所述。 将该文件放在与被测试的包相同的包中。该文件将被排除在正常的程序包之外，但在运行 “go test” 命令时将被包含。

进行测试之前需要初始化操作(例如打开连接)，测试结束后，需要做清理工作(例如关闭连接)等等。这个时候就可以使用TestMain()。如下列TestMain用于初始化变量servers 和pools

```
var servers []*tempredis.Server
var pools []redsync.Pool
func TestMain(m *testing.M) {
    for i := 0; i < 4; i++ {
        server, err := tempredis.Start(tempredis.Config{})
        if err != nil {
            panic(err)
        }
        defer server.Term()
        servers = append(servers, server)
    }
    pools = makeTestPools()
    result := m.Run()
    for _, server := range servers {
        server.Term()
    }
    os.Exit(result)
}
```

安装：

gomock主要包含两个部分：" gomock库"和 “ 辅助代码生成工具mockgen”

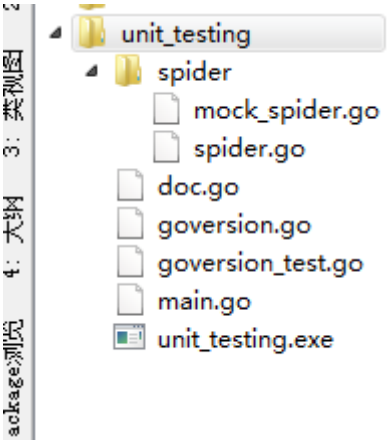
他们都可以通过`go get`来获取：

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

在mockgen文件夹下执行 go build 后在当前目录下生成了一个可执行程序mockgen。

将mockgen程序移动到$GOPATH/bin目录下就可以在命令行运行mockgen

示例：



示例代码目录如上图，这里spider.go作为接口文件，定义了spider包的接口：

```
package spider

type Spider interface {
    GetBody() string
}
```

这里假设接口`GetBody`直接可以抓取"https://golang.org"首页的 “Build version” 字段来得到当前Golang发布出来的版本。

这里在goversion.go中对这个接口进行使用：

```
// goversion
package main

import (
    "unit_testing/spider"
)
```

```
func GetGoVersion(s spider.Spider) string {
    body := s.GetBody()
    return body
}
```

现在要针对goversion文件中的GetGoVersion方法进行单元测试。
首先定义一个goversion_test.go 文件，文件名必须是该格式。然后在文件中添加一个TestGetGoVersion方法：

```
func TestGetGoVersion(t *testing.T) {
    v := GetGoVersion(spider.CreateGoVersionSpider())
    if v != "go1.8.3" {
        t.Error("Get wrong version %s", v)
    }
}
```

这里spider.CreateGoVersionSpider()返回一个实现了Spider接口的用来获得Go版本号的爬虫。
这个单元测试其实既测试了函数GetGoVersion也测试了spider.CreateGoVersionSpider返回的对象。
而有时候，我们可能仅仅想测试下GetGoVersion函数，或者我们的spider.CreateGoVersionSpider爬虫实现还没有写好，那该如何是好呢？
此时Mock工具就显的尤为重要了。

这里首先用gomock提供的mockgen工具生成要mock的接口（即Spider接口）的实现：

```
mockgen -destination spider/mock_spider.go -package spider -source spider/spider.go
```

如果有接口文件，则可以通过：

- -source： 指定接口文件
- -destination: 生成的文件名
- -package:生成文件的包名

- -imports: 依赖的需要import的包
- -aux_files:接口文件不止一个文件时附加文件
- -build_flags: 传递给build工具的参数

```
mockgen -destination spider/mock_spider.go -package spider unit_testing/spider
Spider
```

在我们的上面的例子中，并没有使用"-source",那是如何实现接口的呢？
mockgen还支持通过反射的方式来找到对应的接口。只要在所有选项的最后增加一个包名和里面对应的类型就可以了。其他参数和上面的公用。

通过注释指定mockgen
如果有多个文件，并且分散在不同的位置，那么我们要生成mock文件的时候，需要对每个文件执行多次mockgen命令（假设包名不相同）。这样在真正操作起来的时候非常繁琐，mockgen还提供了一种通过注释生成mock文件的方式，此时需要借助go的"go generate "工具。
在接口文件的注释里面增加如下：

```
//go:generate mockgen -destination mock_spider.go -package spider
unit_testing/spider Spider
```

这样，只要在spider目录下执行

```
go generate
```

命令就可以自动生成mock文件了。

在生成了mock实现代码之后，我们就可以进行正常使用了。这里假设结合testing进行使用（当然你也可考虑使用GoConvey）。我们就可以
在单元测试代码里面首先创建一个mock控制器：
mockCtl := gomock.NewController(t)
将* testing.T传递给gomock生成一个"Controller"对象，该对象控制了整个Mock的过程。在操作完后还需要进行回收，所以一般会在New后面defer一个Finish
defer mockCtl.Finish()
然后就是调用mock生成代码里面为我们实现的接口对象：

mockSpider := spider.NewMockSpider(mockCtl)

有了实现对象，我们就可以调用其断言方法了:EXPECT()
这里gomock非常牛的采用了链式调用法，通过"."连接函数调用，可以像链条一样连接下去。
mockSpider.EXPECT().GetBody().Return("go1.8.3")
这里的每个"."调用都得到一个"Call"对象，该对象有如下方法：
func (c *Call) After(preReq *Call) *Call
func (c *Call) AnyTimes() *Call
func (c *Call) Do(f interface{}) *Call
func (c *Call) MaxTimes(n int) *Call
func (c *Call) MinTimes(n int) *Call
func (c *Call) Return(rets ...interface{}) *Call
func (c *Call) SetArg(n int, value interface{}) *Call
func (c *Call) String() string
func (c *Call) Times(n int) *Call
这里EXPECT()得到实现的对象，然后调用实现对象的接口方法，接口方法返回第一个"Call"对象，然后对其进行条件约束。
上面约束都可以在文档中或者根据字面意思进行理解，这里列举几个例子：
指定返回值
如我们的例子，调用Call的`Return`函数，可以指定接口的返回值：

```
mockSpider.EXPECT().GetBody().Return("go1.8.3")
```

指定执行次数
有时候我们需要指定函数执行多次，比如接受网络请求的函数，计算其执行了多少次。

```
mockSpider.EXPECT().Recv().Return(nil).Times(3)
```

执行三次Recv函数，这里还可以有另外几种限制：

- AnyTimes() ： 0到多次
- MaxTimes(n int) ：最多执行n次，如果没有设置
- MinTimes(n int) ：最少执行n次，如果没有设置

指定执行顺序

有时候我们还要指定执行顺序，比如要先执行Init操作，然后才能执行Recv操作。

```
initCall := mockSpider.EXPECT().Init()
mockSpider.EXPECT().Recv().After(initCall)
```

单元测试代码如下：

```
// goversion_test
package main

import (
    "testing"
    "unit_testing/spider"

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

//mockgen工具是gomock提供的用来为要mock的接口生成实现的
//-source： 指定接口文件
//-destination: 生成的文件名
//-package:生成文件的包名
//-imports: 依赖的需要import的包
//-aux_files:接口文件不止一个文件时附加文件
//-build_flags: 传递给build工具的参数

//mockgen -destination spider/mock_spider.go -package spider -source
spider/spider.go
func TestGetGoVersion(t *testing.T) {
    //这里spider.CreateGoVersionSpider()返回一个实现了Spider接口的用来获得Go版本
号的爬虫。
    //这个单元测试其实既测试了函数GetGoVersion也测试了
    //spider.CreateGoVersionSpider返回的对象。
```

```
    //而有时候，我们可能仅仅想测试下GetGoVersion函数，
    //或者我们的spider.CreateGoVersionSpider爬虫实现还没有写好，那该如何是好呢？
    //    此时Mock工具就显的尤为重要了。

    //    v := GetGoVersion(spider.CreateGoVersionSpider())

    //****************这里是分割下***************
    //    t.Skip("跳过基准测试后下面的测试将不再执行")
    //    fmt.Println("跳过基准测试")

    mockCtl := gomock.NewController(t)
    defer mockCtl.Finish()
    //NewMockSpider返回一个接口的mock实例(该方法是mockgen工具自动生成的)
    mockSpider := spider.NewMockSpider(mockCtl)
    //    这里EXPECT()得到实现的对象，然后调用实现对象的接口方法
    mockSpider.EXPECT().GetBody().Return("go1.8.3")
    v := GetGoVersion(mockSpider)

    if v != "go1.8.3" {
        t.Error("Get wrong version %s", v)
    }
}
```

最新通过go test 执行单元测试