http mocking: Small is beautiful

Piero de Salvia
github.com/pierods
twitter.com/dspiero

What am I interested into when mocking?

request	response
httpmock, httptest.NewRecorder, gock,...	Baloo, hoverfly, prism...

We will focus today on the request.

Plenty of packages, but...small is beautiful

And homemade is even more beautiful. Let’s take a look at http.Client:

```
type Client struct {

    Transport RoundTripper

…
```

What is a RoundTripper?

```
type RoundTripper interface {

    RoundTrip(*Request) (*Response, error)

}
```

RoundTripper does the actual network call

And since it is an interface, we can implement it, and replace the exported field in http.Client:

```
httpClient = &http.Client{}

httpClient.Transport = MyRoundTripper
```

This is what gock, httpmock and others do*. How difficult it is to create a RoundTripper? Let’s see an example.

*almost - they replace the global Transport

The first RoundTripper you will probably create is...

```
type FixedResponseRoundTripper struct {

    Resp     *http.Response

    RespBytes []byte

}

func (f FixedResponseRoundTripper) RoundTrip(*http.Request) (*http.Response,
error) {

    byteReader := ioutil.NopCloser(bytes.NewReader(c.RespBytes))

    f.Resp.Body = byteReader

    f.Resp.StatusCode = 200

    return f.Resp, nil

}
```

And the second is...

```
type BadRequestRoundTripper struct {...

func (b BadRequestRoundTripper) RoundTrip(*http.Request) (*http.Response,
error) {

    byteReader := ioutil.NopCloser(bytes.NewReader(b.RespBytes))

    b.Resp.Body = byteReader

    b.resp.StatusCode = 400

    return b.resp, nil

}
```

A likely third...

A “memento” round tripper (equivalent to http.ResponseRecorder):

```
type MementoRoundTripper struct {
    Request   *http.Request
    …
```

Which will give back the received request:

```
func (m *MementoRoundTripper) Request() *http.Request {
    return m.request
}
```

(notice the pointer receiver)

An interesting fourth: WaitgroupRoundTripper

```
type WaitGroupRoundTripper struct {

    …

    wg       *sync.WaitGroup

}

func (c WaitGroupRoundTripper) RoundTrip(*http.Request) (*http.Response,
error) {

…

defer c.wg.Done()

…

}
```

Quite useful for testing asynchronous calls

Sometimes http requests are launched asynchronously, making it hard for testing code to verify results:

…

```
go httpClient.Get(...)
```

…

In this case, by using a WaitgroupRoundTripper, I can test the above code.

```
func TestSomethingAsync(t *testing.T) {

    …

    var wg sync.WaitGroup

    wg.Add(1)

    …

    rt := roundtrippers.NewWaitgroupRoundTripper(&wg)

    httpClient.Transport = rt

    go whateverFunc()

    wg.Wait()

    assert.Equal(...)

}
```

Advantages and disadvantages

Advantages:

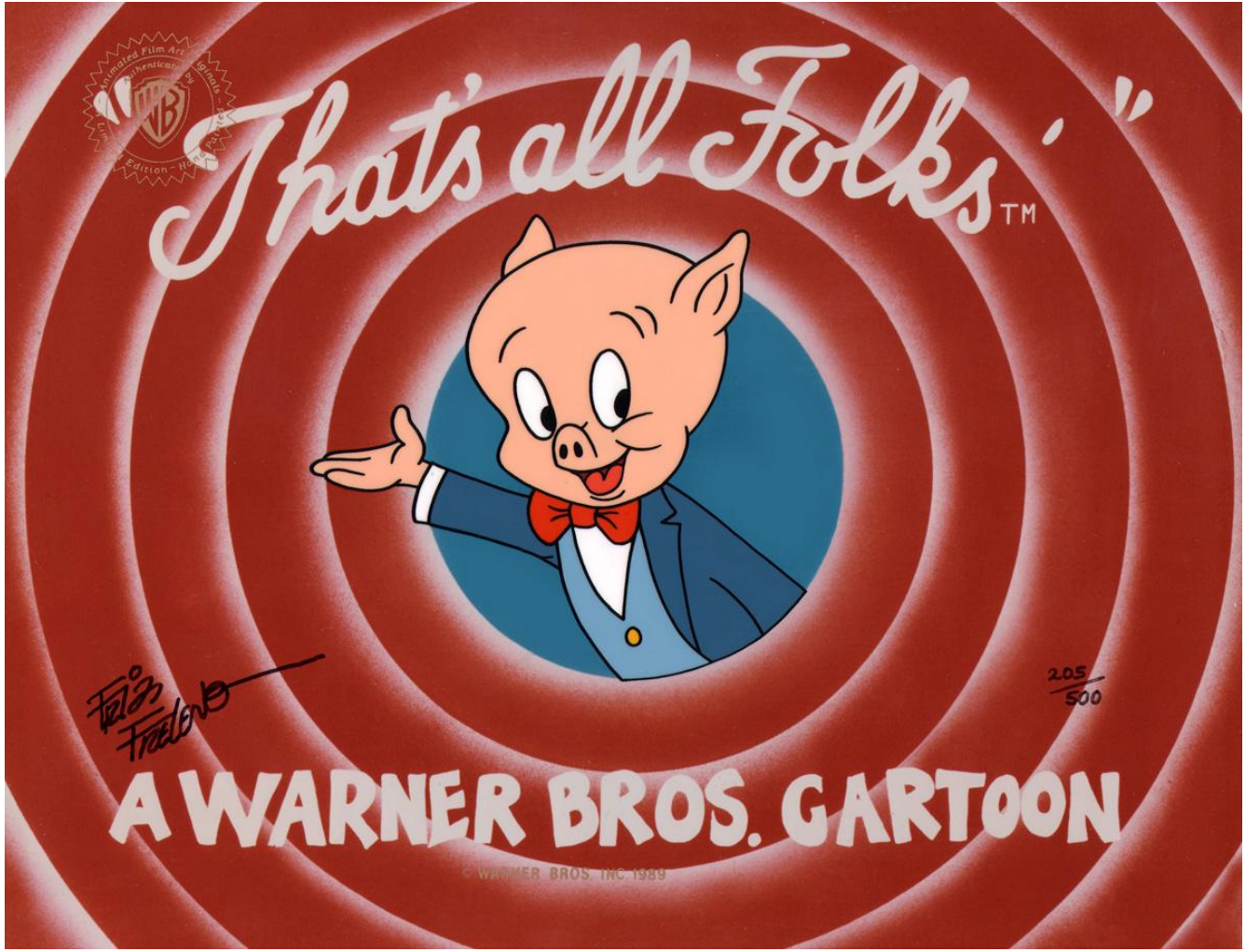
- No importing third party packages
- No learning APIs (however small)
- Mocked client will do exactly what I want since I will code it (using my types, being in the package I want etc).

Advantages and disadvantages

Disadvantages:

- If the http.Client I need to mock is in another package, or however hidden, this method will not work*. Most third party packages replace the global Transport so they don’t have this problem.
- Buy vs build: some people will prefer to use third-party packages anyway.

*Since http.Client is supposed to be reused, it is usually stored in a struct field or in a package variable, so this is not really a problem in practice.


That's all Folks'."
TM
Felix Freleng
205/500
A WARNER BROS. CARTOON
© WARNER BROS. INC. 1989