Uncle John's Bathroom Reader

RSpec Basics 3

by Speedy Spec

A few weeks ago we were discussing RSpec's **mocking** functionality, and the concept of **message expectations** were mentioned. In this issue of **Uncle John's Bathroom Reader**, let's dive deeper expectations and how you can set them in RSpec.

Message expectations are complementary to the concept of assertions and are used to make sure that certain methods are invoked in certain ways within your tests. A simple example might be to make sure the method **`valid_session?`** is being called in the following spec:

```
describe TweetController do
  context 'when logged in' do
    it 'should allow posting' do
      controller.should_receive(:valid_session?)
      controller.post(:tweet => '@horse_ebooks lol')
    end
  end
end
```

In the above example, if the **`controller`** object does not receive the message (i.e. the method is not called), then RSpec will raise an expectation error indicating a failure. The **`should_receive`** syntax is very similar to **`stub`** which we learned about previously. Like **`stub`** it will overwrite the specified method and can be used to return an alternate response by chaining **`and_return`** on the declaration:

```
it 'should not allow posting' do
  # Ensure valid_session? is called, but is falsey
  controller.should_receive(:valid_session?).and_return(false)
  controller.post(:tweet => 'im #poopin')
end
```

Unlike **stub** however, you can also allow **should_receive** to delegate to the original implementation for the result.

```
it 'should allow posting' do
  controller.should_receive(:valid_session?).and_call_original
  controller.post(:tweet => 'it stinks')
end
```

Message expectations are an easy way to do *integration* testing between different components with RSpec. The ease of use is a double-edged sword, as message expectations also allow you to tightly couple to method internals, leading to brittle tests. Like most things in RSpec, message expectations work best when used in moderation!

That's it for this issue of **Uncle John's Bathroom Reader**, now go wash your hands.

Different expectations

Expecting arguments:

Instead of expecting a simple method call, RSpec also allows you to specify an expectation that a method will be invoked in a particular way, with specific arguments, e.g.:

```
# Expect no arguments
o.should_receive(:post).with(no_args)

# Expect any kind of arguments
o.should_receive(:get).with(any_args)

# Expect to be called with specific
# arguments
o.should_receive(:put).with(12, true)
```

Setting Responses:

Message expectations use the same API for setting responses that basic RSpec mocks do:

```
should_receive(:get).and_raise(err)
should_receive(:put).and_throw(ball)
should_receive(:post).and_yield(mock)

should_receive(:delete) do |arg|
  # generate the expected response
end
```

Counting Calls:

Setting up message expectations to receive certain messages a few times per example is fairly straight-forward:

```
should_receive(:delete).once
should_receive(:patch).twice
should_receive(:xhr).exactly(3).times
should_receive(:post).at_most(:once)
should_receive(:get).at_least(:twice)
should_receive(:put).at_most(5).times
```

RSpec message expectation docs: https://github.com/rspec/rspec-mocks

Collection of tips and tricks for writing better specs: http://betterspecs.org/

Send comments and suggestions for future issues to: **ujbr@lookout.com**