

Rspec expect document:

<https://github.com/rspec/rspec-expectations#typesclasses>

Rspec Should be kind of <https://www.rubydoc.info/gems/rspec-expectations/2.0.1/RSpec/Matchers>

Rspec 教程: <https://ihower.tw/rails/testing-cn.html>

Rspec should eq!:

<https://relishapp.com/rspec/rspec-expectations/v/3-9/docs/built-in-matchers>

Rspec <https://semaphoreci.com/community/tutorials/getting-started-with-rspec>

Rspec: <https://relishapp.com/rspec/>

Ruby metaprogramming : <https://www.toptal.com/ruby/ruby-metaprogramming-cooler-than-it-sounds>

Scrum 介绍: <https://www.youtube.com/watch?v=XU0lRltyFM>

设计模式: (这份文件里的UML图没法显示。。可能还是找别的参考一下比较好。)

<https://gof.quanke.name/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E8%AE%BE%E8%AE%A1%E5%8E%9F%E5%88%99.html>

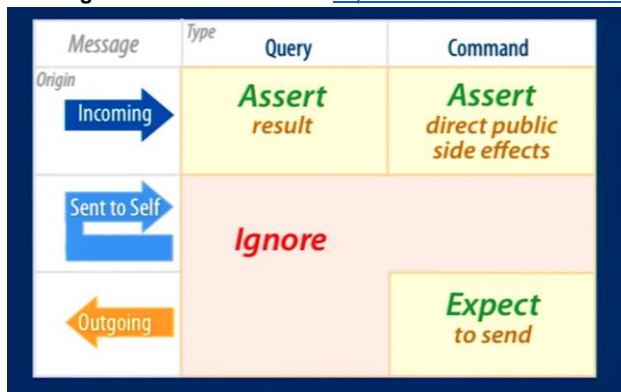
有图的设计模式:

<https://juejin.im/post/5bc96aff265da0aa94a4493#heading-10>

Command pattern

<https://www.cnblogs.com/java-my-life/archive/2012/06/01/2526972.html>

The Magic Tricks of Unit Test <https://medium.com/@smeriwether/the-magic-tricks-of-unit-testing-28ce0b300cee>



Override and Overload

16 Overloading and overriding are complementary things, overloading means the same method name but different parameters, and overriding means the same method name in a subclass with the same parameters. So its not possible for overloading and overriding to happen at the same time because overloading implies different parameters.

Examples:



```
class A {  
    public void doSth() { /// }  
}  
  
class B extends A {  
    public void doSth() { /* method overridden */ }  
    public void doSth(String b) { /* method overloaded */ }  
}
```

Focus on messages

Incoming:

the object under test receives messages from others
Not let the outside see

Outgoing:

Sends messages
Can not let the inside see out

Sent to itself:

calling a private method

These three can be queries or commands

Query:

Return something/change nothing

Command:

Return nothing/change something

Smart User Story

<https://www.visual-paradigm.com/scrum/write-user-story-smart-goals/>

Rails原生test , self.should(base):

<https://stackoverflow.com/questions/5160780/what-does-self-includedbase-do-in-ruby-on-rails-restful-authentication>

Cucumber/user story/feature/steps

What is Scrum?

- Scrum is an “agile” process that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.

5 Step Keywords

1. **Given** steps represent state of world before event: preconditions
2. **When** steps represent event
 - e.g., simulate user pushing a button
3. **Then** steps represent expected postconditions; check if true
4. / 5. **And** & **But** extend previous step

How to test functionality?

How do we decide

- ✦ ... where to test functionality?
- ✦ *Rule #1: Unit tests should be fast.*
 - Tests that access a db are not fast.
 - Therefore, anything that accesses the db in a unit test should be mocked and/or stubbed.
- ✦ *Rule #2: Test functionality in only one place.*
 - Functionality that is tested in the model via unit tests should not also be tested by functional tests.

How do we decide (cont.)

- ✦ *Rule #3: Don't test Rails*
 - It's not your responsibility to make sure that Rails is behaving according to its specifications.
- Thus, don't test actions like ...
 - ✦ making sure that an object that has just been saved can be accessed;
 - ✦ making sure that creating an object causes the object count to rise by 1.

Controller Specs and Refactoring



Should & Should-not

- Matcher applies test to receiver of *should*

<code>count.should == 5</code>	Syntactic sugar for <code>count.should == (5)</code>
<code>5.should(be.<(7))</code>	<code>be</code> creates a lambda that tests the predicate expression
<code>5.should be < 7</code>	Syntactic sugar allowed
<code>5.should be_odd</code>	Use <code>method_missing</code> to call <code>odd?</code> on 5
<code>result.should include(elt)</code>	calls <code>Enumerable#include?</code>
<code>result.should match(/regex/)</code>	
<u><code>should_not</code> also available</u>	
<code>result.should render_template('search_tmdb')</code>	



Two new seam concepts

- `stub`
 - similar to `should_receive`, but not expectation
 - `and_return` optionally controls return value
- `mock`: create dumb “stunt double” object
 - stub individual methods on it:


```
m = mock('movie1')
m.stub(:title).and_return('Rambo')
```
 - shortcut: `m=mock('movie1',:title=>'Rambo')`

each seam enables just enough functionality
for some *specific* behavior under test



Test Cookery #1

- Each spec should test *just one behavior*
- Use seams as needed to isolate that behavior
- Determine which expectation you'll use to check the behavior
- Write the test and make sure it fails for the right reason
- Add code until test is green
- Look for opportunities to refactor/beautify



Test techniques we know

```
obj.should_receive(a).with(b).and_return(c)
obj.stub(a).and_return(b)
```

Optional!

```
d = mock('impostor')
```

```
obj.should match-condition
```

Rails-specific extensions to RSpec:

```
assigns(:instance_var)
response()
render_template()
```