Jasmine

Testing JavaScript

Jasmine is a browser based testing framework for JavaScript, similar to RSpec for Ruby, but runs via a HTML file in the browser.

The Jasmine program is initialized simply by adding its source files to the project directory (lib and src), and then including the scrpt and and spec files in the SpecRunner.html file.

To test open the html file in the browser (cmd + O)

The Test Suite

A test suite begins with a call to the global Jasmine function ‘describe’, passing through a string (the name of the test) and a function (the test itself). The function contains various sections for the suite, and in its most basic form it will contain one ‘spec’ by calling ‘it’, also passing through a naming string and function. The ‘it’ call function will then have an expectation which will return true if the test passes. The scoping rules of JavaScript apply, so any variable declared in the ‘describe’ function will be available in the ‘it’ function.

*describe("A suite", function() {*

*it("contains spec with an expectation", function() {*

*expect(true).toBe(true);*

*});*

*});*

Expectations

Matchers

There a many included matchers in Jasmine, however it is possible to make custom matchers. Commonly used matchers:

* expect(a).toBe(b)
* expect(a).not.toBe(c)
* expect(message).toMatch(/regex/)
* expect(message).not.toMatch(/regex/)
* expect(a).toBeDefined()
* expect(null).toBeNull()
* expect(a).toBeTruthy()
* expect(b).toBeFasly()

Mocking

Since JavaScript is run in the browser it is often important to mock and stub variables allowing them to be available and return a specific value during tests. Mocks in JavaScript can be created in two ways, either the creation of a mock object which has specific return values as required and is injected, or using Jasmine test spies to change the output of methods on an object.

If the test is simple, like checking that an object returns 1 which called with .isStormy(), it is possible to simply create an object with one method returning 1.

Test Doubles - Spies

Spies are used to sub a method but still track calls to it and all its arguments.

When not wanting to actually call dependent functions of a method being tested (such as if the method performed AJAX calls), the spyOn() function can be used.

spyOn takes two parameters, the name of the object in question and the name of the method to be spied upon. The spyOn() function will then replace the spied method with a stub, which means if the spy stub method is called the calls to the methods can still be tracked, but the original dependent functions will not be called.

**.and.callThrough()**

If calls to some of original dependent functions are actually required, the .callThrough() function can be chained to the spyOn() function aswell.

**.and.ReturnValue()**

If a specific value is required to be returned from the stubbed method, the .andReturnValue() can be used. This allows for any value to be returned from the stub and passed through to the program, mocking behaviour.

**.and.callFake()**

.callFake() allows for the stubbed method to be replaced with the one defined in the parameters. Arguments can be passed through to the fake function from the Jasmine test scope.

**and.throwError()**

To throw an error chain the .throwError method and include the error message in the parameters.

Async in Jasmine

Async can be controlled in Jasmine using timeouts and done() functions, to check how long to wait before running and failing the test respectiviely. Putting the argument done into a before, it, or after will not perform the function until

*beforeEach(function(done) {*

*setTimeout(function() {*

*value = 0;*

*done();*

*}, 1);*

*});*

This spec will not start until the done function is called in the call to beforeEach above. And this spec will not complete until its done is called.

*it("should support async execution of test preparation and expectations", function(done) {*

*value++;*

*expect(value).toBeGreaterThan(0);*

*done();*

*});*