AngularJS, Jasmine Testing, and Mocking Global Objects

Posted At : August 26, 2015 3:54 PM | Posted By : Cutter
Related Categories: Jasmine, AngularJS, JavaScript

So, you've made the jump to writing tests for your code. You've discovered the value of TDD, and even integrated it into your CI process. Awesome! Well, until your tests fail. Then you bang your head against your desk, trying to figure out how to test "this" scenario, and get totally frustrated by the lack of information. Makes you want to quit writing code and start driving the little cart around the driving range that picks up the golf balls.

We've all been there. This morning in fact. Don't give up, you'll figure it out soon enough. For example, you're trying to test a new method in your code. Within this new method you're referencing an object that's on the global scope. But, of course, you get a 'fail' every time, because your test doesn't know what that object is.

The answer is to mock the object. But how do you mock an object on the global scope? Really, it's easier than you might think. Just decorate your window.

Within your root 'describe' block, you probably have a test global 'beforeEach()' where you're mocking your module. Here, you can create a 'decorator' for Angular's '$window', and attach a mock of your object.

```
beforeEach(angular.mock.module('myModule', function ($provide) {
  $provide.decorator('$window', function ($delegate) {
   $delegate.foo = {
    bar: function () {
     // just for test
    }
   };
   return $delegate;
  });
 }));
```

Then later, when your tests hit code that references that global 'foo' object, your tests won't bomb. Further, you can attach spies to see that object methods are being called in your test

```
it ('myMethod', function () {
```

```
  spyOn(window.foo, 'bar').andReturn(true);

  ctrl.myMethod();
  $scope.$digest();

  expect(window.foo.bar).toHaveBeenCalled();
});
```

When the test is run, and the 'myMethod()' function is called on your controller, which internally calls 'foo.bar()', the test will see that and resolve it with the 'andReturn()' value you supplied.

This is invaluable when you're writing tests for those areas of your app where you're calling the Google API's, or some script that dynamically embeds a video player or something. Your test isn't worried about their code (that's something else all together). Your test is about knowing that your code does what it needs to do.

As they say, "there's usually more than one way to skin a cat" (I'm really going to have to find out where that coloquial nugget came from), so if you know of another way to perform the same test scenario, please share.