FREE eBook

LEARNING jasmine

Free unaffiliated eBook created from
Stack Overflow contributors.

#jasmine

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: jasmine

It is an unofficial and free jasmine ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official jasmine.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with jasmine

Remarks

Sometime testing our JavaScript code becomes a tough task. Jasmine is a behavior-driven development framework for testing our JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean syntax which makes you easily write the tests. You can find the Jasmine documentation here and the project in GitHub.

Versions

Version	Release Date
1.0.0	2010-09-14
1.3.0	2012-11-27
2.0.0	2013-12-16
2.1.0	2014-11-14
2.2.0	2015-02-02
2.3.0	2015-04-28
2.4.0	2015-12-02
2.5.0	2016-08-30

Examples

Installation or Setup

Installing Jasmine standalone

Download the latest Jasmine release from the Jasmine release page:

Running Jasmine locally

1. Run Jasmine in the browser by downloading the zip file, extracting it, the referencing the files as follows:

```
<link rel="shortcut icon" type="image/png" href="jasmine/lib/jasmine-
2.0.0/jasmine_favicon.png">
<link rel="stylesheet" type="text/css" href="jasmine/lib/jasmine-2.0.0/jasmine.css">

<script type="text/javascript" src="jasmine/lib/jasmine-2.0.0/jasmine.js"></script>
<script type="text/javascript" src="jasmine/lib/jasmine-2.0.0/jasmine-html.js"></script>
<script type="text/javascript" src="jasmine/lib/jasmine-2.0.0/boot.js"></script>
```

Installing Jasmine using npm (Node Package Manager)

1. Set up project directory for Jasmine

Create a folder and run `npm init` this will create an empty `package.json` file and will ask some questions about your project to fill project `json` file.

Add 2 directories `app` - for the Server and `spec` - for tests

2. Get Jasmine

   From root project directory run

   ```
   npm install jasmine-node --save

   npm install request --save

   npm install express --save
   ```

   this will get you the packages

   `./node_packages/.bin/jasmine-node spec` will run jasmine binary

   After this your `package.json` should look similar to this

   package.json file, after which that file should look like this:

```
{
    "name": "Jasmine",
    "version": "0.0.1",
    "description": "Jasmine",
    "main": "index.js",
    "scripts": {
      "test": "./node_modules/.bin/jasmine-node spec"
    },
    "author": "Me",
    "license": "ISC"
  }
```

Install with npm

```
npm install -g jasmine
```

If being used with karma, install `karma-jasmine`

```
npm install --save-dev karma-jasmine
```

Hello World

To create a most basic test with Jasmine go to your `spec` (tests) folder and add file named `testSpec.js`.

In that file add following:

```
var request = require("request");

describe("Hello World Test", function() {
```

```
  // This is your test bundle

  describe("GET SO", function() {
    //This is testing that http GET works

    it("Checks if SO is online", function() {
      // This is description of your test - this is what you get when it fails

      request.get("http://stackoverflow.com/", function(error, response, body) {
        // this is your test body

        expect(response.statusCode).toBe(200);
        // this is your test assertion - it expects status code to be '200'
      });
    });
  });
});
```

Read Getting started with jasmine online: https://riptutorial.com/jasmine/topic/1302/getting-started-with-jasmine

Chapter 2: Attributes

Remarks

There are some terms you must be aware of before going to write the Jasmine test cases.

1. Suites

A suit is the starting point of a Jasmine test cases, it actually calls the global jasmine function describe. It can have two parameters, a string value which describes the suit, and a function which implements the suit.

2. Spec

Like suites, a spec starts with a string which can be the title of the suit and a function where we write the tests. A spec can contain one or more expectation that test the state of our code.

3. Expectation

Value of an expectation is either true or false, an expectation starts with the function expect. It takes a value and call the actual one.

Examples

Suites

```
describe("Includes validations for index page", function () {

});
```

Spec

```
it("Spy call for datepicker date validation", function () {

});
```

Expectation

```
describe("Includes validations for index page", function () {
    var indexPage;

    it("Check for null values", function () {
        // We are going to pass "" (null) value to the function
        var retVal = indexPage.isNullValue("");
        expect(retVal).toBeTruthy();
    });

});
```

Read Attributes online: https://riptutorial.com/jasmine/topic/7980/attributes

Chapter 3: Custom Matchers

Examples

Adding Custom Matchers

Custom matchers can be added in jasmine using the syntax:

```
jasmine.addMatchers([
    toMatch: function () {
    return {
        compare: function (actual, expected) {
            return {
                pass: actual===expected,
                message: "Expected actual to match expected
            }
        }
    }
}
]);
```

This matcher can now be called with:

```
expected(actual).toMatch(expected);
```

Negative Matchers

Custom matchers will have their pass value negated when using 'not'. Custom matchers can have a negative compare attribute to explicitly handle cases where their negation is desired:

```
toMatch: function () {
        return {
            compare: function (actual, expected) {
                return {
                    pass: actual===expected,
                    message: "Expected actual to match expected"
                }
            },
            negativeCompare: function(actual, expected){
                return {
                    pass: actual!==expected,
                    message: "Expected actual not to match expected"
                }
            }
        }
    }
```

Read Custom Matchers online: https://riptutorial.com/jasmine/topic/6945/custom-matchers

Chapter 4: Spies

Remarks

A spy is defined as a test specific function which intercepts calls to an underlying function in the application code and dispatches its own implementation when the underlying function is called to test the interface rather than the implementation.

Examples

Spying on an existing function

Jasmine can spy on an existing function using the `spyOn` function.

```
let calculator = {
  multiply: function(a, b) {
     return a * b;
  },

  square: function(a) {
    return this.multiply(a, a);
  }
}


describe('calculator', function() {
  it('squares numbers by multiplying them by themselves', function() {
    let num = 2;
    spyOn(calculator, 'multiply');
    calculator.square(NUM);
    expect(calculator.multiply).toHaveBeenCalledWith(NUM, NUM);
  })
});
```

After the function has been spied on it is replaced with a spy, that can be queried for information about how and when it has been called.

Creating a new spy

We can use `jasmine.createSpy()` to create a standalone spy. This is often useful if we need to pass a function as a callback to another function and want to test how it is used.

```
// source code
function each(arr, fn) {
    arr.forEach(fn);
}

// test code
describe('each', function() {
  let mockFn = jasmine.createSpy();
```

```
  it('calls a function for each item in the array ', function() {
    let arr = [1,2,3,4,5]
    each(arr, mockFn);
    expect(mockFn.calls.count()).toBe(arr.length);
  })
});
```

Spying on an angular service

In this example we have a service, let's call it search service that has a method called search() which will initiate a get request to a back end API.

```
function SearchService($http) {
    const service = {};

    service.search = function() {
        return $http({method: 'GET', url: `/api/search`})
    }

    return service;
}
angular.module('app').factory('searchService', SearchService);
```

Testing

```
describe('search service', function() {
    var $httpBackend;
    var searchService;
    beforeEach(angular.mock.module('app'));

    beforeEach(inject(function(_$httpBackend_, _searchService_) {
        $httpBackend = _$httpBackend_;
        searchService = _searchService_;
    }));

    it('should perform http call to the search api', function(){
        searchService.search();
        $httpBackend.expectGET('/api/search');
    });

})
```

Spying on an angular service that doesn't call back end service

```
function calculatorService() {
    const service = {};
    service.add = function(a,b) {
        return a + b
    }

    return service;
}

angular.module('app').factory('calculatorService', calculatorService);
```

Testing

```
describe('calculator service', function() {
    var calculatorService;
    beforeEach(angular.mock.module('app'));

    beforeEach(inject(function(_calculatorService_) {
         calculatorService = _calculatorService_;
    }));

    it('should should add two numbers', function(){
        var actual = calculatorService.add(1,2);
        expect(actual).toBe(3);
    });
})
```

Spying on a property

```
const foop = {
    get value() {},
    set value(v) {}
};

it('can spy on getter', () => {
    spyOnProperty(foop, 'value', 'get').and.returnValue(1);
    expect(foop.value).toBe(1);
});

it('and on setters', () => {
    const spiez = spyOnProperty(foop, 'value', 'set');
    foop.value = true;
    expect(spiez).toHaveBeenCalled();
});
```

Read Spies online: https://riptutorial.com/jasmine/topic/1979/spies

Credits

S. No	Chapters	Contributors
1	Getting started with jasmine	Akxe, cezar, Community, dev ツ, Matas Vaitkevicius, Paul Sweatte, Sibeesh Venu, tkwargs
2	Attributes	Sibeesh Venu
3	Custom Matchers	SilentLupin
4	Spies	Ahmed Ahmed, Ben McCormick, gmuraleekrishna, hansmaad, Laoujin, Paul Sweatte