عنوان: آماده سازی Jasmine برای پروژه های Asp.Net MVC
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۷/۰۸ ۹:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, test, TDD, JavaScriptFrameWork, Jasmine

با گسترش روز افزون برنامه‌های تحت وب، نیاز به یک سری ابزار برای تست و اطمینان از نحوه عملکرد صحیح کدهای نوشته شده احساس می‌شود. Jasmine یکی از این ابزارهای قدرتمند برای تست کدهای JavaScript است.
چندی پیش در سایت جاری چند مقاله خوب توسط یکی از دوستان درباره Qunit منتشر شد. Qunit یک ابزار قدرتمند و مناسب برای تست کدهای جاوااسکریپت است و در اثبات صحت این گفته همین کافیست که بدانیم برای تست کدهای نوشته شده در پروژه‌های متن بازی هم چون Backbone.Js و JQuery از این فریم ورک استفاده شده است. اما به احتمال قوی در ذهن شما این سوال مطرح شده است که خب! در صورت آشنایی با Qunit چه نیاز به یادگیری Jasmine یا خدای نکرده Mocha و FuncUnit است؟ هدف صرفا معرفی یک ابزار غیر برای تست کد است نه مقایسه و نتیجه گیری برای تعیین میزان برتری این ابزارها. اصولا مهم‌ترین دلیل برای انتخاب، علاوه بر امکانات و انعطاف پذیری، فاکتور راحتی و آسان بودن در هنگام استفاده است که به صورت مستقیم به شما و تیم توسعه نرم افزار بستگی دارد.

اما به عنوان توسعه دهنده نرم افزار که قرار است از این ابزار استفاده کنیم بهتر است با تفاوت‌ها و شباهت‌های مهم این دو فریم ورک آشنا باشیم:

»Jasmine یک فریم ورک تست کدهای جاوا اسکریپ بر مبنای Behavior-Driven Development است در حالی که Qunit بر مبنای Test-Driven Development است و همین مسئله مهم‌ترین تفاوت بین این دو فریم ورک می‌باشد.
»اگر قصد دارید که از Qunit نیز به روش BDD استفاده نمایید باید از ترکیب Pavlov به همراه Qunit استفاده کنید.
»Jasmine از مباحث مربوط به Spies و Mocking به خوبی پشتیبانی می‌کند ولی این امکان به صورت توکار در Qunit فراهم نیست. برای اینکه بتوانیم این مفاهیم را در Qunit پیاده سازی کنیم باید از فریم ورک‌های دیگر نظیر SinonJS به همراه Qunit استفاده کنیم.
»هر دو فریم ورک بالا به سادگی و راحتی کار معروف هستند
»تمام موارد مربوط به الگوهای Matching در هر دو فریم ورک به خوبی تعبیه شده است
» هر دو فریم ورک بالا از مباحث مربوط به Asynchronous Testing برای تست کدهای Ajax ای به خوبی پشتیبانی می‌کنند.

بررسی چند مفهوم
قبل از شروع، بهتر است که با چند مفهوم کلی و در عین حال مهم این فریم ورک آشنا شویم

```
describe('JavaScript addition operator', function () {
 it('adds two numbers together', function () {
  expect(1 + 2).toEqual(3);
 });
});
```

در کد بالا یک نمونه از تست نوشته شده با استفاده از Jasmine را مشاهده می‌کنید. دستور describe برای تعریف یک تابع تست مورد استفاده قرار می‌گیرد که دارای دو پارامتر ورودی است. ابتدا یک نام را به این تست اختصاص دهید(بهتر است که این عنوان به صورت یک جمله قابل فهم باشد). سپس یک تابع به عنوان بدنه تست نوشته می‌شود. به این تابع Spec گفته می‌شود.
در تابع it کد بالا شما می‌توانید کدهای مربوط بدنه توابع تست خود را بنویسید. برای پیاده سازی Assert در توابع تست مفهوم expectationها وجود دارد. در واقع expect برای بررسی مقادیر حقیقی با مقادیر مورد انتظار مورد استفاده قرار می‌گیرد و شامل مقادیر true یا false خواهد بود.
برای Setup و Teardown توابع تست خود باید از توابع beforeEach و afterEach که بدین منظور تعبیه شده اند استفاده کنید.

```
describe("A spec (with setup and tear-down)", function() {
```

```
  var foo;

  beforeEach(function() {
    foo = 0;
    foo += 1;
  });

  afterEach(function() {
    foo = 0;
  });

  it("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });

  it("can have more than one expectation", function() {
    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

کاملا واضح است که در تابع beforeEach مجموعه دستورالعمل‌های مربوط به setup تست وجود دارد. سپس دو تابع it برای پیاده سازی عملیات Assertion نوشته شده است. در پایان هم دستورات تابع afterEach ایجاد می‌شوند.

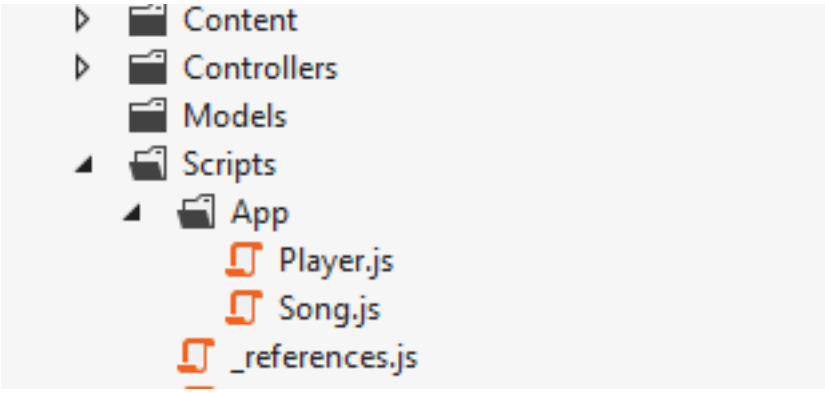
اگر در کد تست خود قصد دارید که یک تابع describe یا it را غیر فعال کنید کافیست یک x به ابتدای آن‌ها اضافه کنید و دیگر نیاز به هیچ کار اضافه دیگری برای comment کردن کد نیست.

```
xdescribe("A spec", function() {
  var foo;

  beforeEach(function() {
    foo = 0;
    foo += 1;
  });

  xit("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });
});
```

توابع describe و it بالا در هنگام تست نادیده گرفته می‌شوند و خروجی آن‌ها مشاهده نخواهد شد.

درادامه قصد پیاده سازی یک مثال را با استفاده از Jasmine و RequireJs در پروژه Asp.Net MVC دارم.
برای شروع آخرین نسخه Jasmine را از اینجا دریافت نمایید. یک پروژه Asp.Net MVC به همراه پروژه تست به صورت Empty ایجاد کنید(در هنگام ایجاد پروژه، گزینه create unit test را انتخاب نمایید). فایل دانلود شده را unzip نمایید و دو پوشه lib و spec ،به همراه فایل specRunner.html را در پروژه تست خود کپی نمایید.
فولدر lib شامل فایل‌ها کدهای Jasmine برای setup و tear down و spice و تست کدهای شما می‌باشد.
فایل specRunner.html به واقع یک فایل برای نمایش فایل‌های تست و همچنین نمایش نتیجه تست است.
فولدر spec نیز شامل کدهای Jasmine برای کمک به نوشتن تست می‌باشد.

در این مثال قصد داریم فایل‌های player.js و song.js که به عنوان نمونه به همراه این فریم ورک قرار دارد را در قالب یک پروژه MVC به همراه RequireJs، تست نماییم. در نتیجه این فایل‌ها را از فولدر src انتخاب نمایید و آن‌ها را در قسمت Scripts پروژه اصلی خود کپی کنید(ابتدا یک پوشه به نام App بسازید و فایل‌ها را در آن قرار دهید)



برای استفاده از requireJs باید دستور define را در ابتدا این فایل‌ها اضافه نماییم. در نتیجه فایل‌های Player.js و Song.js را باز کنید و تغییرات زیر را در ابتدای این فایل‌ها اعمال نمایید.

Song.js

```
define(function () {
    function Song() {
    }

    Song.prototype.persistFavoriteStatus = function (value) {
        // something complicated
        throw new Error("not yet implemented");
    };
});
```

Player.js

```
define(function () {
    function Player() {
    }
    Player.prototype.play = function (song) {
        this.currentlyPlayingSong = song;
        this.isPlaying = true;
    };

    Player.prototype.pause = function () {
        this.isPlaying = false;
    };

    Player.prototype.resume = function () {
        if (this.isPlaying) {
            throw new Error("song is already playing");
        }

        this.isPlaying = true;
    };

    Player.prototype.makeFavorite = function () {
        this.currentlyPlayingSong.persistFavoriteStatus(true);
    };
});
```

حال فایل SpecRunner.html را بازکنید و کدهای مربوط به تگ script که به مسیر اصلی فایل‌های تست اشاره می‌کند را Comment نمایید و به جای آن تگ Script مربوط به RequireJs را اضافه نمایید. برای پیکر بندی RequireJs باید از baseUrl و paths استفاده کرد.

```
<link rel="shortcut icon" type="image/png" href="lib/jasmine-1.2.0/jasmine_favicon.png">
<link rel="stylesheet" type="text/css" href="lib/jasmine-1.2.0/jasmine.css">
<script type="text/javascript" src="lib/jasmine-1.2.0/jasmine.js"></script>
<script type="text/javascript" src="lib/jasmine-1.2.0/jasmine-html.js"></script>

<script type="text/javascript" src="../RequireJsMvcStarter/Scripts/require.js"></script>

<!-- include source files here... -->
<!--<script type="text/javascript" src="spec/SpecHelper.js"></script>-->
<!--<script type="text/javascript" src="spec/PlayerSpec.js"></script>-->

<!-- include spec files here... -->
<!--<script type="text/javascript" src="src/Player.js"></script>
<script type="text/javascript" src="src/Song.js"></script>-->

<script type="text/javascript">

    require.config({
        baseUrl: '../RequireJsMvcStarter/Scripts/App',
        paths: {
            spec: '../../../RequireJsMvcStarter.Scripts.Test/spec'
        }
    });
</script>
```

baseUrl در پیکر بندی requireJs به مسیر فایل‌های پروژه که در پروژه اصلی MVC قرار دارد اشاره می‌کند. paths برای تعیین مسیر فایل‌های تست که در پوشه spec در پروژه تست قرار دارد اشاره می‌کند. اگر دقت کرده باشید به دلیل اینگه تگ‌های script مربوط به لود فایل‌های SpecHelper.js و PlayerSpec.js به صورت comment در آمده اند در نتیجه این فایل‌ها لود نخواهند شد و خروجی مورد نظر مشاهده نمی‌شود. در این جا باید از مکانیزم AMD موجود در RequireJs استفاده نماییم و فایل‌های مربوطه را لود کنیم. برای این کار نیاز به اضافه کردن دستور require در ابتدای تگ script به صورت زیر در این فایل است. در نتیجه فایل‌های PlayerSpec و SpecHelper نیز توسط RequireJs لود خواهند شد.

```
<script type="text/javascript">

    require(['spec/PlayerSpec', 'spec/SpecHelper'], function() {
        var jasmineEnv = jasmine.getEnv();
        jasmineEnv.updateInterval = 1000;

        var htmlReporter = new jasmine.HtmlReporter();

        jasmineEnv.addReporter(htmlReporter);

        jasmineEnv.specFilter = function(spec) {
            return htmlReporter.specFilter(spec);
        };

        var currentWindowOnload = window.onload;

        window.onload = function() {
            if (currentWindowOnload) {
                currentWindowOnload();
            }
            execJasmine();
        };

        function execJasmine() {
            jasmineEnv.execute();
        }
    });
</script>
```

نیاز به یک تغییر کوچک دیگر نیز وجود دارد. فایل PlayerSpec را باز نمایید و وابستگی فایل‌های آن را تعیین نمایید. از آن جا که این فایل برای تست فایل‌های Player , Song ایجاد شده است در نتیجه باید از define برای تعیین این وابستگی‌ها استفاده نماییم.

PlayerSpec.js

```
1  define(['Player', 'Song'], function(Player, Song) {
2      describe("Player", function() {
3          var player;
4          var song;
5
6          beforeEach(function() {
7              player = new Player();
8              song = new Song();
9          });
10
```

یادآوری :

»دستور describe در فایل بالا برای تعریف تابع تست است. همان طور که می‌بینید بک نام به آن داده می‌شود به همراه بدنه تابع تست.

»دستور beforeEach برای آماده سازی مواردی است که قصد داریم در تست مورد استفاده قرار گیرند. همانند متدهای Setup در UnitTest.

» دستور expect نیز معادل Assert در UnitTest است و برای بررسی صحت عملکرد تست نوشته می‌شود.

اگر فایل SpecRunner.html را دوباره در مرورگر خود باز نمایید تصویر زیر را مشاهده خواهید کرد که به عنوان موفقیت آمیز بودن پیکر بندی پروژه و تست‌های آن می‌باشد.