عنوان: **آموزش QUnit #1**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۴/۱۷ ۱۹:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, jQuery, QUnit, Unit testing

مقدمه: تست و آزمایش کد برنامه‌ها و وب سایت‌هایمان، بهترین راه کاهش خطا و مشکلات آنها بعد از انتشار است. از جمله روش‌های موجود، تست واحد است که ویژوال استادیو نیز از آن برای پروژه‌های دات نت پشتیبانی می‌کند. با افزایش روز افزون کتابخانه‌های جاوا اسکریپتی و جی کوئری، نیاز به تست کدهای جاواسکریپتی نیز بیشتر به نظر می‌رسد و بهتر است تست واحد و آزمایش شوند. اما برخلاف کدهای C# و ASP.NET تست کدهای جاوا اسکریپت، مخصوصا زمانی که به دستکاری عناصر DOM می‌پردازیم و یا رویدادهای درون صفحه وب را با استفاده از جی کوئری می‌نویسیم، حتی اگر در فایل جداگانه‌ای نوشته شود، این بدان معنی نیست که آماده تست واحد است و ممکن است امکان نوشتن تست وجود نداشته باشد.
بنابراین چه چیزی یک تست واحد است؟ در بهترین حالت توابعی که مقداری را برمی گردادنند، بهترین حالت برای تست واحد است. اما در بیشتر موارد شما نیاز دارید تا تاثیر کد را بر روی عناصر صفحه نیز مشاهد نمایید.

ساخت تست واحد

برای تست پذیری بهتر، توابع جاوا اسکریپت و هر کد دیگری، آن را می‌بایست طوری بنویسید که مقادیر تاثیر گذار در اجرای تابع به عنوان ورودی تابع در نظر گرفته شده باشند و همیشه نتیجه به عنوان خروجی تابع برگردانده شود؛ قطعه کد زیر را در نظر بگیرید:

```
function prettyDate(time){
    var date = new Date(time || ""),
      diff = (((new Date()).getTime() - date.getTime()) / 1000),
      day_diff = Math.floor(diff / 86400);

    if ( isNaN(day_diff) || day_diff < 0 || day_diff >= 31 )
      return;

    return day_diff == 0 && (
        diff < 60 && "just now" ||
        diff < 120 && "1 minute ago" ||
        diff < 3600 && Math.floor( diff / 60 ) +
          " minutes ago" ||
        diff < 7200 && "1 hour ago" ||
        diff < 86400 && Math.floor( diff / 3600 ) +
          " hours ago") ||
      day_diff == 1 && "Yesterday" ||
      day_diff < 7 && day_diff + " days ago" ||
      day_diff < 31 && Math.ceil( day_diff / 7 ) +
        " weeks ago";
  }
```

تابع pertyDate اختلاف زمان حال را نسبت به زمان ورودی، بصورت یک رشته برمی گرداند. اما در اینجا مقدار زمان حال، در خط سوم، در خود تابع ایجاد شده است و در صورتی که بخواهیم برای چندین مقدار آن را تست کنیم زمان حال متفاوتی در نظر گرفته می‌شود و حداکثر، زمان 31 روز قبل را نمایش داده و در بقیه تاریخ ها undefined را بر می‌گرداند. برای تست واحد، چند تغییر می‌دهیم.

بهینه سازی، مرحله اول:

پارامتری به عنوان مقدار زمان جاری برای تابع در نظر می‌گیریم و تابع را جدا کرده و در یک فایل جداگانه قرار می‌دهیم. فایل prettydate.js بصورت زیر خواهد شد.

```
function prettyDate(now, time){
  var date = new Date(time || ""),
    diff = (((new Date(now)).getTime() - date.getTime()) / 1000),
    day_diff = Math.floor(diff / 86400);

  if ( isNaN(day_diff) || day_diff < 0 || day_diff >= 31 )
    return;

  return day_diff == 0 && (
      diff < 60 && "just now" ||
      diff < 120 && "1 minute ago" ||
      diff < 3600 && Math.floor( diff / 60 ) +
```

```
        " minutes ago" ||
      diff < 7200 && "1 hour ago" ||
      diff < 86400 && Math.floor( diff / 3600 ) +
        " hours ago") ||
    day_diff == 1 && "Yesterday" ||
    day_diff < 7 && day_diff + " days ago" ||
    day_diff < 31 && Math.ceil( day_diff / 7 ) +
      " weeks ago";
}
```

حال یک تابع برای تست داریم، چند تست واحد واقعی می‌نویسیم

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Refactored date examples</title>
  <script src="prettydate.js"></script>
  <script>
  function test(then, expected) {
    results.total++;
    var result = prettyDate("2013/01/28 22:25:00", then);
    if (result !== expected) {
      results.bad++;
      console.log("Expected " + expected +
        ", but was " + result);
    }
  }
  var results = {
    total: 0,
    bad: 0
  };
  test("2013/01/28 22:24:30", "just now");
  test("2013/01/28 22:23:30", "1 minute ago");
  test("2013/01/28 21:23:30", "1 hour ago");
  test("2013/01/27 22:23:30", "Yesterday");
  test("2013/01/26 22:23:30", "2 days ago");
  test("2012/01/26 22:23:30", undefined);
  console.log("Of " + results.total + " tests, " +
    results.bad + " failed, " +
    (results.total - results.bad) + " passed.");
  </script>
</head>
<body>

</body>
</html>
```

در کد بالا یک تابع بدون استفاده از Qunit برای تست واحد نوشته ایم که با آن تابع prettyDate را تست می‌کند. تابع test مقدار زمان حال و رشته خروجی را گرفته و آن را با تابع اصلی تست می‌کند در آخر تعداد تست ها، تست‌های شکست خورده و تست های پاس شده گزارش داده می‌شود.
خروجی می‌تواند مانند زیر باشد:

.Of 6 tests, 0 failed, 6 passed

.Expected 2 day ago, but was 2 days ago

.f 6 tests, 1 failed, 5 passed

عنوان: **آموزش QUnit #2**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۴/۲۱ ۱۲:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, Unit testing, jQuery, QUnit

فریم ورک تست جاوا اسکریپت QUnit: انتخاب و استفاده از یک فریم ورک برای تست کدهای جاوا اسکریپت، قطعا نتیجه بهتری را به همراه خواهد داشت. من در این جا از QUnit که یکی از بهترین‌های تست واحد است، استفاده می‌کنم. برای این کار فایل‌های qunit.js و qunit.css را دانلود و مانند زیر برای تست واحد آماده کنید:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Refactored date examples</title>

  <link rel="stylesheet" href="../qunit.css">
  <script src="../qunit.js"></script>
  <script src="prettydate.js"></script>

  <script>
  test("prettydate basics", function() {
    var now = "2013/01/28 22:25:00";
    equal(prettyDate(now, "2013/01/28 22:24:30"), "just now");
    equal(prettyDate(now, "201308/01/28 22:23:30"), "1 minute ago");
    equal(prettyDate(now, "2013/01/28 21:23:30"), "1 hour ago");
    equal(prettyDate(now, "2013/01/27 22:23:30"), "Yesterday");
    equal(prettyDate(now, "2013/01/26 22:23:30"), "2 days ago");
    equal(prettyDate(now, "2012/01/26 22:23:30"), undefined);
  });
  </script>
</head>
<body>

<div id="qunit"></div>

</body>
</html>
```

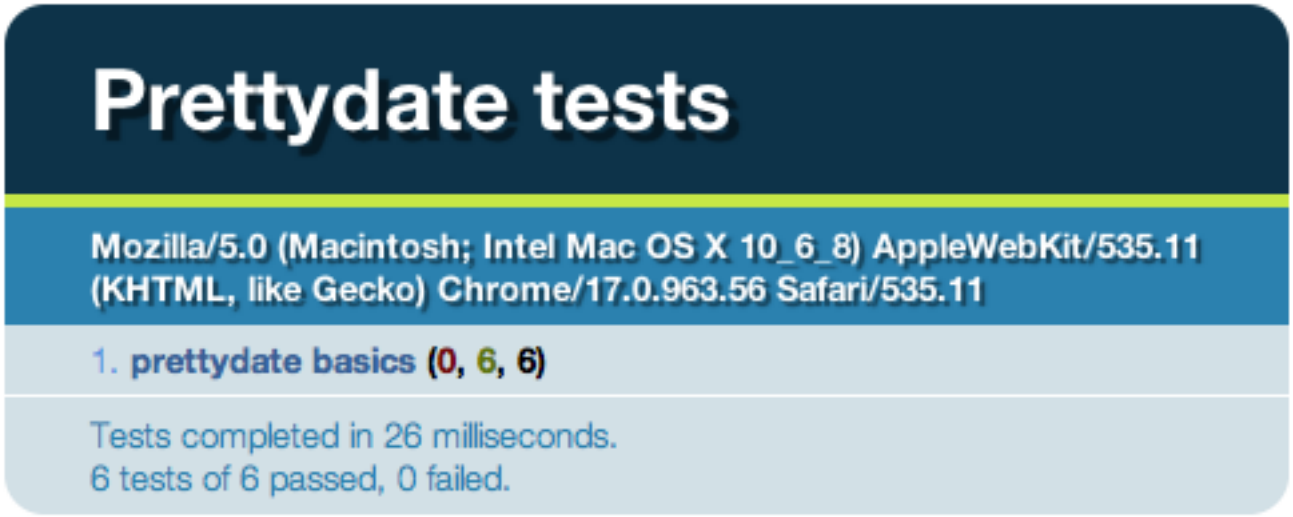
در کد بالا ابتدا فایل‌های فریم ورک و فایل prettydate.js را اضافه کردیم. برای نمایش نتیجه تست، یک تگ div با نام qunit در بین تگ body اضافه می‌کنیم.

تابع test:

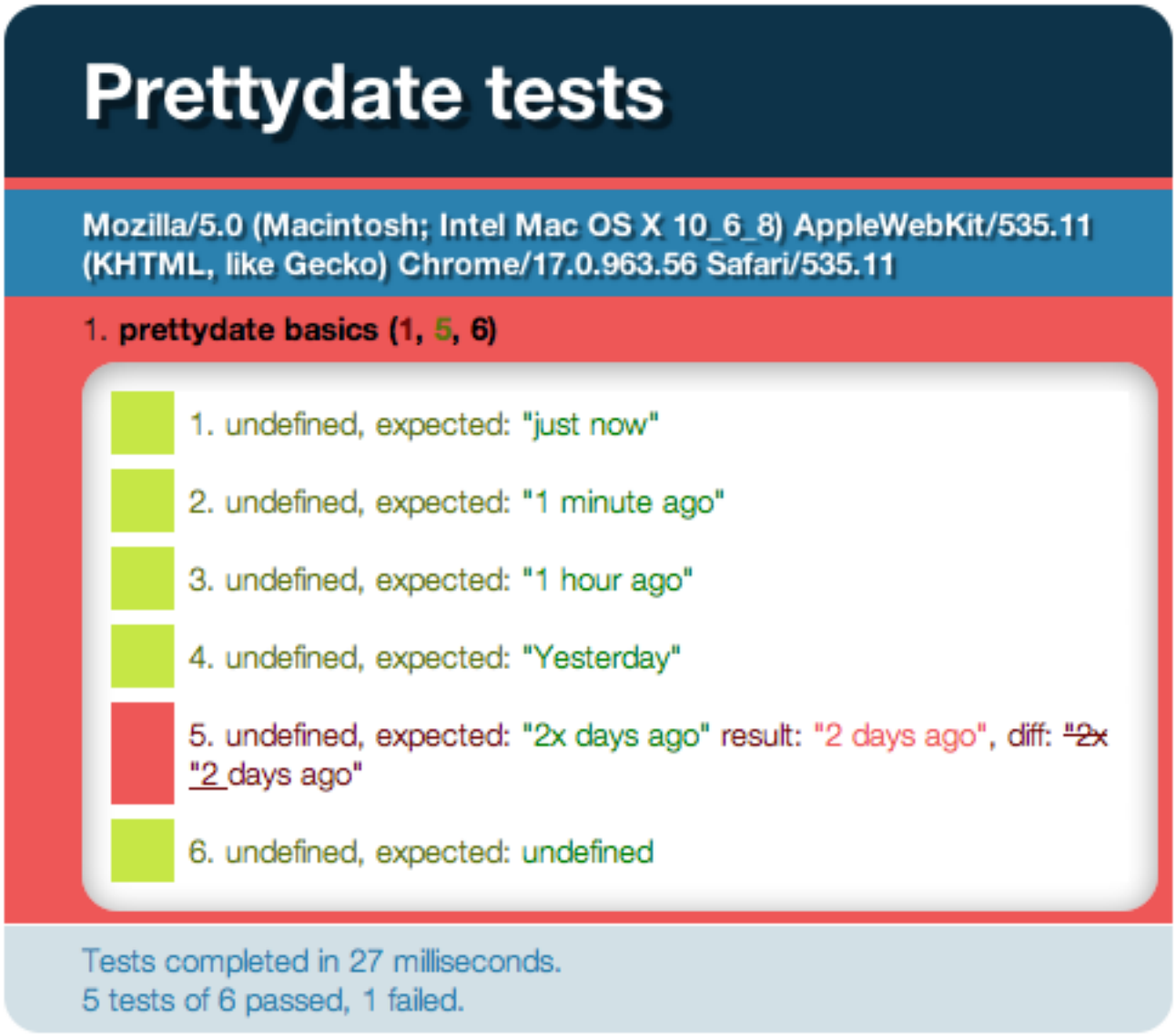
این تابع برای تست توابع نوشته شده، استفاده می‌شود. ورودی‌های این تابع، یکی عنوان تست و دومی یک متود دیگر، به عنوان ورودی دریافت می‌کند که در آن بدنه تست نوشته می‌شود.

تابع equal:

اولین تابع برای سنجش تست واحد equal است و در آن، تابعی که می‌خواهیم تست کنیم با مقدار خروجی آن مقایسه می‌شود.
فایل را با نام test.htm ذخیره و آن را در مرورگر خود باز نمایید. خروجی در شکل آورده شده است:



همین طور که در تصویر بزرگ می‌بینید اطلاعات مرورگر، زمان تکمیل تست و تعداد تست، تعداد تست پاس شده و تعداد تست شکست خورده، نشان داده شده است.



اگر یکی از تست‌ها با شکست روبرو شود رنگ پس زمینه قرمز و جزئیات شکست نمایش داده می‌شوند.

بهینه سازی، مرحله اول:

در حال حاضر تست ما کامل نیست زیرا امکان تست n weeks ago یا تعداد هفته پیش میسر نیست. قبل از آنکه این را به آزمون اضافه کنیم، تغییراتی در تست می‌دهیم

```
test("prettydate basics", function() {
  function date(then, expected) {
    equal(prettyDate("2013/01/28 22:25:00", then), expected);
  }
  date("2013/01/28 22:24:30", "just now");
  date("2013/01/28 22:23:30", "1 minute ago");
  date("2013/01/28 21:23:30", "1 hour ago");
  date("2013/01/27 22:23:30", "Yesterday");
  date("2013/01/26 22:23:30", "2 days ago");
  date("2012/01/26 22:23:30", undefined);
});
```

تابع prettyDate را در تابع دیگری به نام date قرار می‌دهیم. این تغییر سبب می‌شود تا امکان مقایسه زمان ورودی تست جاری با تست قبلی فراهم شود.

تست دستکاری عناصر DOM:

تا اینجا با تست توابع آشنا شدید، حالا می‌خواهیم تغییراتی در prettyDate دهیم تا امکان انتخاب عناصر DOM و به روزرسانی آن نیز وجود داشته باشد. فایل prettyDate2.js در زیر آورده شده است:

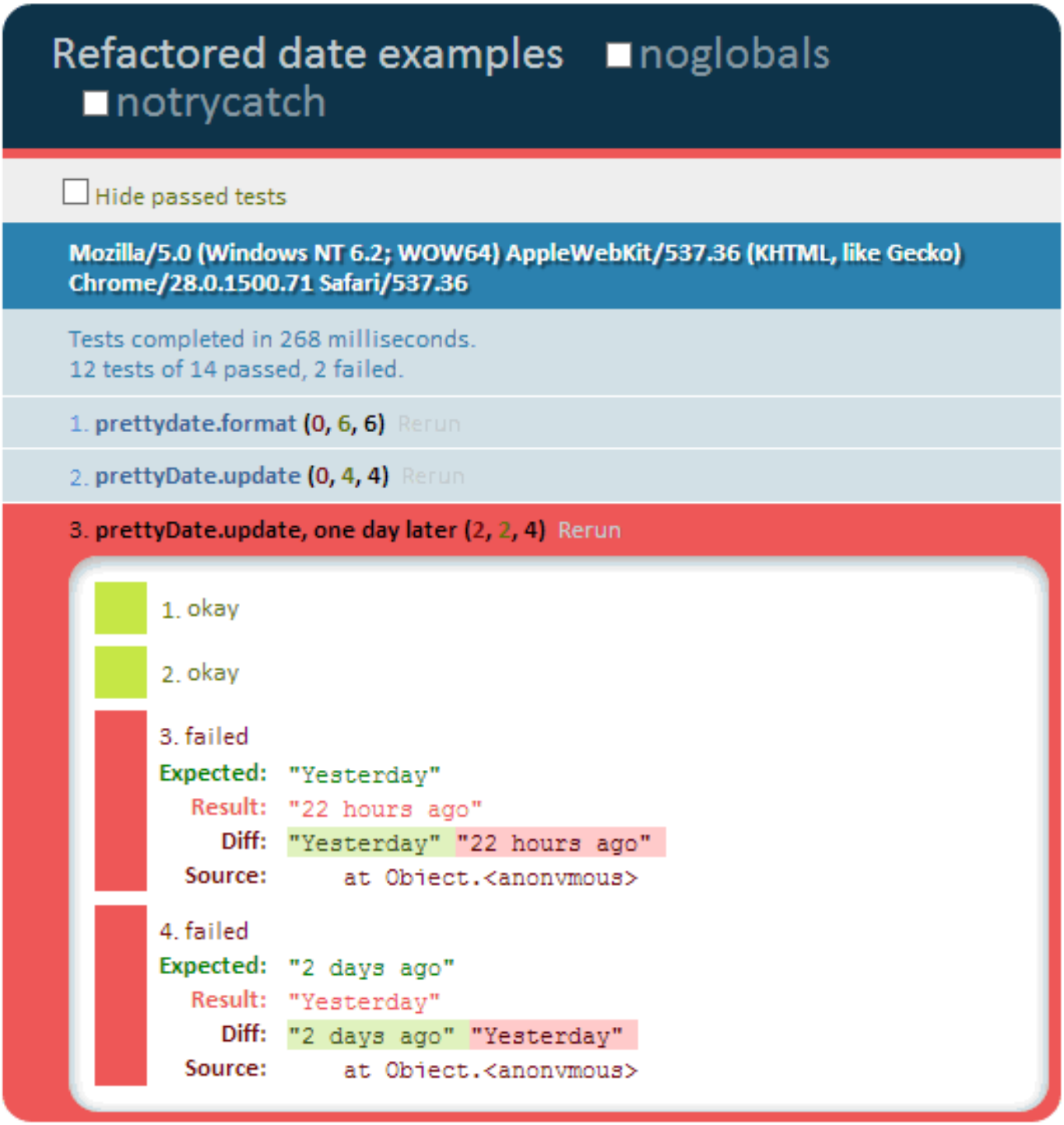
```
var prettyDate = {
  format: function(now, time){
    var date = new Date(time || ""),
      diff = (((new Date(now)).getTime() - date.getTime()) / 1000),
      day_diff = Math.floor(diff / 86400);

    if ( isNaN(day_diff) || day_diff < 0 || day_diff >= 31 )
      return;

    return day_diff === 0 && (
        diff < 60 && "just now" ||
        diff < 120 && "1 minute ago" ||
        diff < 3600 && Math.floor( diff / 60 ) +
          " minutes ago" ||
        diff < 7200 && "1 hour ago" ||
        diff < 86400 && Math.floor( diff / 3600 ) +
          " hours ago") ||
      day_diff === 1 && "Yesterday" ||
      day_diff < 7 && day_diff + " days ago" ||
      day_diff < 31 && Math.ceil( day_diff / 7 ) +
        " weeks ago";
  },

  update: function(now) {
    var links = document.getElementsByTagName("a");
    for ( var i = 0; i < links.length; i++ ) {
      if ( links[i].title ) {
        var date = prettyDate.format(now, links[i].title);
        if ( date ) {
          links[i].innerHTML = date;
        }
      }
    }
  }
};
```

prettyDate شامل دو تابع، یکی format که weeks ago به آن اضافه گردیده و تابع update که با انتخاب تگ‌ها، مقدار title را به تابع فرمت و خروجی آن را در Html هر عنصر قرار می‌دهد. حال یک تست واحد می‌نویسیم:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Refactored date examples</title>
  <link rel="stylesheet" href="../qunit.css">
  <script src="../qunit.js"></script>
  <script src="prettydate2.js"></script>
  <script>
  test("prettydate.format", function() {
    function date(then, expected) {
      equal(prettyDate.format("2013/01/28 22:25:00", then),
        expected);
    }
    date("2013/01/28 22:24:30", "just now");
    date("2013/01/28 22:23:30", "1 minute ago");
    date("2013/01/28 21:23:30", "1 hour ago");
    date("2013/01/27 22:23:30", "Yesterday");
    date("2013/01/26 22:23:30", "2 days ago");
    date("2012/01/26 22:23:30", undefined);
  });

  function domtest(name, now, first, second) {
    test(name, function() {
      var links = document.getElementById("qunit-fixture")
        .getElementsByTagName("a");
      equal(links[0].innerHTML, "January 28th, 2013");
      equal(links[2].innerHTML, "January 27th, 2013");
      prettyDate.update(now);
      equal(links[0].innerHTML, first);
      equal(links[2].innerHTML, second);
    });
```

```
  }
  domtest("prettyDate.update", "2013-01-28T22:25:00Z",
    "2 hours ago", "Yesterday");
  domtest("prettyDate.update, one day later", "2013/01/29 22:25:00",
    "Yesterday", "2 days ago");
  </script>
</head>
<body>

<div id="qunit"></div>
<div id="qunit-fixture">

<ul>
  <li id="post57">
    <p>blah blah blah...</p>
    <small>
      Posted <span>
        <a href="/2013/01/blah/57/" title="2013-01-28T20:24:17Z"
          >January 28th, 2013</a>
      </span>
      by <span><a href=""></a></span>
    </small>
  </li>
  <li id="post57">
    <p>blah blah blah...</p>
    <small>
      Posted <span>
        <a href="/2013/01/blah/57/" title="2013-01-27T22:24:17Z"
          >January 27th, 2013</a>
      </span>
      by <span><a href=""></a></span>
    </small>
  </li>
</ul>

</div>

</body>
</html>
```

همین طور که مشاهد می‌کنید در تست واحد اول خود تابع prettyDate.format را تست نموده ایم. در تست بعدی عناصر DOM نیز دستکاری و تست شده است. تابع domtest با جستجوی تگ qunit-fixture و تگ‌های a درون آن، مقدار نهایی html آن با مقدار داده شده، مقایسه شده است.



در شکل بالا نتیجه تست واحد نشان داده شده است.

عنوان: **آموزش QUnit #3**
نویسنده: مجتبی کاویانی
تاریخ: ۱۳۹۲/۰۴/۲۴ ۱۷:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: JavaScript, Unit testing, jQuery, QUnit

در قسمت‌های قبلی با مفهوم تست واحد و کتابخانه quint آشنا شدید و مثالی را نیز با هم بررسی کردیم. در ادامه به قابلیت‌های بیشتر این کتابخانه می‌پردازیم.

توابع اعلان نتایج:

qunit سه تابع را جهت اعلان نتایج تست واحد فراهم نموده است

تابع ok:

تابع پایه‌ای تست واحد، دو پارامتر را به عنوان ورودی دریافت می‌کند و در صورتیکه بررسی نتیجه پارامتر اول برابر true باشد، تست با موفقیت روبرو شده است. پارامتر دوم برای نمایش یک پیام است. در مثال زیر حالت‌های مختلف آن بررسی شده است. مقادیر non-empty string، true به معنی موفقیت و مقادیر null،"",NaN،0،false و undefined به معنی شکست تست می‌باشد. در واقع خروجی تابع ارسالی به اعلان ok یکی از نتایج بالا می‌تواند باشد.

```
//ok( truthy [, message ] )

test( "ok test", function() {
  ok( true, "true succeeds" );
  ok( "non-empty", "non-empty string succeeds" );

  ok( false, "false fails" );
  ok( 0, "0 fails" );
  ok( NaN, "NaN fails" );
  ok( "", "empty string fails" );
  ok( null, "null fails" );
  ok( undefined, "undefined fails" );
});
```

تابع equal:

این اعلان یک مقایسه ساده بین پارامتر اول و دوم تابع می‌باشد که شرط برابری(==) را بررسی می‌نماید. وقتی مقدار اول و دوم برابر باشند، اعلان موفقیت و در غیر این صورت، تست با شکست رویرو شده و هر دو پارامتر نمایش داده می‌شوند.

```
//equal( actual, expected [, message ] )

test( "equal test", function() {
  equal( 0, 0, "Zero; equal succeeds" );
  equal( "", 0, "Empty, Zero; equal succeeds" );
  equal( "", "", "Empty, Empty; equal succeeds" );
  equal( 0, 0, "Zero, Zero; equal succeeds" );

  equal( "three", 3, "Three, 3; equal fails" );
  equal( null, false, "null, false; equal fails" );
});
```

زمانی که می‌خواهید مؤکداً شرط === را بررسی نمایید از ()strictEqual استفاده کنید.

تابع deepEqual:

تکمیل شده دو تابع قبل می‌باشد و حتی امکان مقایسه دو شی را نیز با هم دارا است. علاوه بر این، امکان مقایسه NaN، تاریخ، عبارات باقاعده، آرایه‌ها و توابع نیز وجود دارند.

```
//deepEqual( actual, expected [, message ] )

test( "deepEqual test", function() {
  var obj = { foo: "bar" };
```

```
  deepEqual( obj, { foo: "bar" }, "Two objects can be the same in value" );
});
```

در صورتیکه نمی‌خواهید محتوای دو مقدار را با هم مقایسه کنید، از equal استفاده نمایید اما عموما deepEqual انتخاب بهتری می‌باشد.

تست عملیات کاربر:

گاهی لازم است رویدادهایی که از عملیات کاربران صدا زده می‌شوند تست شوند. در این موارد با صدا زدن تابع trigger جی‌کوئری، تابع مورد نظر را تست نمایید. به مثال زیر توجه نمایید:

```
function KeyLogger( target ) {
  if ( !(this instanceof KeyLogger) ) {
    return new KeyLogger( target );
  }
  this.target = target;
  this.log = [];

  var self = this;

  this.target.off( "keydown" ).on( "keydown", function( event ) {
    self.log.push( event.keyCode );
  });
}
```

این مثال یک گزارش دهنده است و در صورتیکه کاربر، کلیدی را فشار دهد، کد آن را گزارش می‌دهد و در آرایه log ذخیره می‌نماید. حال لازم است بصورت دستی این رویداد را صدا زده و تایع را تست کنیم. تست را بصورت زیر می‌نویسیم:

```
test( "keylogger api behavior", function() {

  var event,
      $doc = $( document ),
      keys = KeyLogger( $doc );

  // trigger event
  event = $.Event( "keydown" );
  event.keyCode = 9;
  $doc.trigger( event );

  // verify expected behavior
  equal( keys.log.length, 1, "a key was logged" );
  equal( keys.log[ 0 ], 9, "correct key was logged" );

});
```

برای این کار تابع KeyLogger را با شی document جی کوئری صدا زدیم و نتیجه را در متغییر keys قرار داده‌ایم. بعد رویداد keydown را با کد 9 پرکرده تایع trigger متغییر $doc را با مقدار event صدا زده‌ایم که در واقع بصورت دستی، یک رویداد اتفاق افتاده است. در آخر هم با اعلان equal تست واحد را انجام داده‌ایم.