

# 边译边学-QUnit下的JavaScript自动化单元测试

这篇文章发布于 2013年04月9日, 星期二, 22:11, 归类于 [JS相关](#)。阅读 144646 次, 今日 23 次 18 条评论

by zhangxinxu from <http://www.zhangxinxu.com>

本文地址: <http://www.zhangxinxu.com/wordpress/?p=3170>

参考原文: [Automating JavaScript Testing with QUnit](#)

原文作者: Jörn Zaefferer

## 参杂译者吐槽的正文

### 一、得了开头不吐槽不舒服的病

我: 我昨晚非一人去看《北京备胎西雅图》了, 还不错哦~ 😊

从: 擦, 又来了, 伙计。我们很忙的, 技术不相干的吐槽留着对你媳妇讲吧! 😏

我: 邵瑞啊! 😏

如果“扫瑞”有用, 那还要警察叔叔干嘛! 😏

(○○○)...



以前曾听过“测试驱动的JS开发”分享, 至于具体内容是什么, 已经忘得一干二净, 不过知道了这个东西, 就是不错的收获。经常看到听到“JS单元测试”, 至于具体怎么操作, 也是不得而知, 因为实际的工作并未使用之。昨天, 看到了一位前端牛人肉身翻墙找工作的经历, 其中就提到, 其回答Facebook的面试题时候把单元测试代码一并发过去了。

妈妈千句唠叨不及朋友一语点拨, 这里的“单元测试”一下子戳中了我的小小神经。于是, 今天, 工作之余, 开始接触“JS单元测试”相关的一些内容。学习目标为, 要对“单元测试”要足够感性整体的认识, 清楚如果在自己项目中使用单元测试, 该怎么做, 可能会遇到哪些阻碍。

目前, 有不少别人说不错的JS单元测试框架, 比方说.....额~我去, 刚那个网页被我close掉了, 大家自己搜下吧。其中有一个是QUnit, 貌似是jQuery团队捣鼓的(并不依赖jQuery)。本文就是要介绍如何使用QUnit做单元测试。至于为什么选择QUnit, 那是因为据说QUnit新手最好上手。

可否讲讲单元测试的必要性呢? 😏

这个嘛, 网上的说法是: 天赐一双金钛狗眼, 立显臭虫得瑟嘴脸🐞。我自己尚未深入尝试, 单元测试是否有必要我并不确定, 我会在文章最后回答此问题。

### 二、QUnit是什么来路?

QUnit是什么来路? 😏

QUnit出生于2008年5月, 走出testrunner嵌入jQuery核心存储库。从此, 它有了自己的名称和文档

和一个安放代码的新家。2009年末QUnit被重构，独立于jQuery，使其可用于测试各种各样的JavaScript框架和应用程序。QUnit宣称其方法现在遵循CommonJS维护规范。虽然QUnit可以运行在服务器端或命令行环境中，但它最有效的是测试浏览器中的JavaScript。

官方自然捡好话讲，说QUnit是个强大简单易用的JavaScript测试框架。jQuery, jQuery UI, jQuery Mobile都是使用QUnit做单元测试的，甚至包括它自己。我抓了抓头发日渐稀少的脑袋瓜，决定自己亲自实践，验证是否如官网上所说的强大简单易用。

### 三、没错，是Hello World!

接触新的语言等，跑通的第一个例子是Hello World! 毕竟是翻译，这里也俗套一把。在测试代码展示之前，我们需要链接几个静态资源，不难想到，是QUnit的JS资源以及CSS样式资源。目前版本是 **1.11.0**。您可以点击下面的链接下载：

- [qunit-1.11.0.js](#)
- [qunit-1.11.0.css](#)

也支持npm安装。如下代码：

```
npm install qunitjs
```

如果您懒得下次，直接include静态资源也是可以的，有CDN，代码如下，官方截图如下下：

```
<link rel="stylesheet" href="http://code.jquery.com/qunit/qunit-1.11.0.css" />
<script src="http://code.jquery.com/qunit/qunit-1.11.0.js"></script>
```



热身完毕，煮青蛙开始啦~~



如下简单基本的测试代码：

```
1. <!DOCTYPE html>
2. <link rel="stylesheet" href="qunit.css">
3. <script src="qunit.js"></script>
4. <script>
5. test("hello", function() {
6.     ok(true, "world");
7. });
8. </script>
9. <h1 id="qunit-header">QUnit Hello world</h1>
10. <h2 id="qunit-banner"></h2>
11. <ol id="qunit-tests"></ol>
```

这是一个非常简约的测试案例。定义了文档类型，包含了QUnit需要的CSS以及JS，定义了另外一个 `script` 元素，`script` 中为单个QUnit测试代码，然后页面上还有一段用来输出结果的标记（如 `<h2 id="qunit-banner">`）。`test()` 函数定义了一个名叫 `"hello"` 的测试。一旦页面载入完毕，QUnit就会运行 `test()`。传给 `test()` 的第二个参数为函数，这才是测试的真身，此处为 `ok()` 方法。`ok()` 方法接受两个参数，第一个参数用来表明测试是通过了呢还是没有，第二个参数则是需要输出的信息。上面这个例子，第一个参数为 `true`，因此，就算指鹿为马也是通过的，实际上没做任何实质性的测试。🐼

如果我们在浏览器中运行上面的代码，则会如下面这模样：

截图为虚，亲见为实，您可以狠狠地点击这里：[true下死活passed demo](#)

显然，如果是：

```
1. test("hello", function() {
2.     ok(false, "world");
3. });
```

则永远不会通过的：

从截图看以看出，“word”这个错误信息输出来了（验证了 `ok()` 第二个参数的功能），同时还标注了错误代码的位置。

截图为虚，亲见为实，too，您可以狠狠地点击这里：[false下死活failed demo](#)

有点样子，下面是不是该来点真格的啦。🍉

Bingong! 恭喜你，你都会抢答啦！

### 三、动真格的QUnit测试

只喝汤不吃肉，婉君也会皱眉头。因此，来个更加完整的例子哈：

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>QUnit测试</title>
5.     <link rel="stylesheet" href="qunit.css">
6.     <script src="qunit.js"></script>
7.     <script src="tests.js"></script>
8. </head>
9. <body>
10.    <h1 id="qunit-header">QUnit测试</h1>
11.    <h2 id="qunit-banner"></h2>
12.    <div id="qunit-testrunner-toolbar"></div>
13.    <h2 id="qunit-userAgent"></h2>
14.    <ol id="qunit-tests"></ol>
15.    <div id="qunit-fixture">测试标记</div>
16. </body>
17. </html>
```

与之前代码不同的是，这里多了个 `tests.js`，以及HTML部分有了更多的标记，如 `<h2 id="qunit-userAgent">` 等。`tests.js` 中内容为：

```
1. function format(string, values) {
2.     for (var key in values) {
3.         string = string.replace(new RegExp("\\{" + key + "}"), values[key]);
4.     }
5.     return string;
6. }
7.
```

```

8. test("basics", function() {
9.     var values = {
10.         name: "world"
11.     };
12.     equal( format("Hello, {name}", values), "Hello, world", "单个匹配" );
13.     equal( format("Hello, {name}, how is {name} today?", values),
14.         "Hello, world, how is world today?", "多个匹配" );
15. });

```

上面代码要做的事情不难理解，`format` 函数的功能就是模板HTML替换技术，MVC的基础之一。

这里使用的是 `equal()` 进行测试，`equal()` 方法作用是做比对，如何比对法呢？很简单，使用JavaScript的 `"=="` 操作符比对其前两个参数。其中第一个参数为实际的值，本例子就是 `format()` 的返回值；第二个参数为我们期望得到的值，格式化的字符串。

OK, 结果到底如何呢？我们先看截图一览究竟：



哇塞，好详细的错误信息提示啊，详细到应该什么结果实际什么结果，甚至还有错误代码位置，不错不错! 🐼

恩，确实，说实话，我开始有点喜欢QUnit了。上例子有demo的，勿浪费，您可以狠狠地点击[这里：更完整多匹配测试不通过demo](#)

发现臭虫，事情就好办多了，多匹配出错，很快就会发现原来是正则表达式那里，全局匹配的 `"g"` 给弄丢了，拿起敌敌畏喷上就嗅啦。

```

1. new RegExp("\{" + key + "\}", "g")

```

于是乎，两个测试都可以通过啦。如果是实际项目，显然，这个测试有必要做进一步的扩展，匹配的对象不止一个关键字以及对象关键字要缺失（是否显示 `undefined`）等。接下来，我们要讲讲如何避免一个测试搞砸另外的测试。coffee time~ ☕



## 四、颗粒、单元

如果仅仅是毫无副作用的功能测试，那实际上很轻松的。但是，一旦代码具有副作用，例如修改DOM或者改变全局变量，事件就变得棘手了。因此，有必要让测试颗粒相互独立，降低可能的副作用。QUnit提供的一些工具以便使用。

### 特定的DOM

如果要测试DOM修改，我们可以使用 `#qunit-fixture` 元素。`#qunit-fixture` 元素好比游戏中的小怪，你可以随便砍杀增长经验，而不要担心第二天上机的时候小怪都没了。在做测试的时候，这个静态标记你可以随便置空或 `append` 元素，每个测试结束的时候QUnit会自动把 `#qunit-fixture` 元素的 `innerHTML` 属性值设置为初始值。如果可以使用jQuery，QUnit会使用jQuery的 `html()` 方法代替，同时清除jQuery事件句柄。

举个例子，我们想测试HTML5 `placeholder` 的填充性，我们会有一个 `input` 元素并带有 `placeholder` 属性：

```
1. <div id="qunit-fixture">
2.     <input id="input" type="text" placeholder="placeholder text" />
3. </div>
```

现在，每个测试你都可以选择 `#input` 元素，打他，骂他，QUnit之后都会让它满血复活的。

### module – 模块

如果我们需要更多清除，而不仅仅是重新设置DOM，我们可以作为每个测试的一部分实现。如果在多个测试中，我们最终做的是同一个清除，我们可以使用 `module()` 方法重构。

不太懂~~ 🤔

上面是字面翻译。通俗讲就是：如果你有很多的推倒、爆菊之类，你可以每次测试搞一下。如果要搞很多次，每次都是爆菊，则你可以使用 `module()` 方法重构。

亮了！💡

`module()` 方法初衷是分组测试，例如，多个测试都测试一个特殊的方法，该方法就可以作为单个模块的一部分。这有助于过滤测试（见下面“QUnit下的开发”）。为了让测试更加颗粒化，我们可以使用 `module()` 提供的 `setup` 和 `teardown` 回调。

```
1. module("core", {
2.     setup: function() {
3.         // 在每个测试之前运行
4.     },
5.     teardown: function() {
6.         // 在每个测试之后运行
7.     }
8. });
9. test("basics", function() {
10.    // 测试.....
11. });
```

`setup` 回调都是在测试之前运行，`teardown` 回调是在测试之后运行。

我们可以使用这些回调创建对象并在测试中使用，而无需依靠闭包（或全局变量）传给测试。之所以会起作用，是因为 `setup` 和 `teardown` 以及真实的测试都在一个自定义的可以自动共享和清除的作用域中调用。

似懂非懂，举个例子呗。🐢

正有此意。下面的代码是个简单（但不完整）的库，用来处理货币值。

```
1. var Money = function(options) {
2.     this.amount = options.amount || 0;
3.     this.template = options.template || "{symbol}{amount}";
4.     this.symbol = options.symbol || "$";
5. };
6. Money.prototype = {
7.     add: function(toAdd) {
8.         this.amount += toAdd;
9.     },
10.    toString: function() {
11.        return this.template
12.            .replace("{symbol}", this.symbol)
```

```

13.         .replace("{amount}", this.amount)
14.     }
15. };
16. Money.euro = function(amount) {
17.     return new Money({
18.         amount: amount,
19.         template: "{amount} {symbol}",
20.         symbol: "EUR"
21.     });
22. };

```

上面代码创建了一个 `Money` 对象，有个为美元(dollars)创建的一个默认构造器方法，还有一个为欧元(euro)创建的工厂方法。这两个方法作用都是操控和打印。这里，我们不是为每个测试都创建一个新的 `Money` 对象，而是使用 `setup` 回调创建对象并存储在测试作用域中。

```

1. module("Money", {
2.     setup: function() {
3.         this.dollar = new Money({
4.             amount: 15.5
5.         });
6.         this.euro = Money.euro(14.5);
7.     },
8.     teardown: function() {
9.         // 可以使用this.dollar和this.euro作清除
10.    }
11. });
12.
13. test("add", function() {
14.     equal( this.dollar.amount, 15.5 );
15.     this.dollar.add(16.1)
16.     equal( this.dollar.amount, 31.6 );
17. });
18. test("toString", function() {
19.     equal( this.dollar.toString(), "$15.5" );
20.     equal( this.euro.toString(), "14.5 EUR" );
21. });

```

上代码中，`setup` 回调中创建了两个对象，名叫 `"add"` 的测试之使用了一个，`"toString"` 这个测试两个对象都使用了。本例中，`teardown` 回调没必要使用，因为没必要要移除已经创建的 `Money` 对象。结果如下截图：



您可以狠狠地点击这里：[module与单元测试demo](#)

## 五、测试异步代码

我们已经看以，QUnit可以控制什么时候真正运行测试代码，只要代码是同步运行的。然而，一旦测试的代码需要使用异步回调（如 `setTimeout` 或Ajax请求），你需要给QUnit反馈，以便停止后面的测试，知道你让它再次运行。

使用的方法很简单，就是 `start()` 和 `end()`，如下代码：

```

1. test("async", function() {
2.     stop();
3.     $.getJSON("resource", function(result) {

```

```

4.         deepEqual(result, {
5.             status: "ok"
6.         });
7.         start();
8.     });
9. });

```

这里jQuery的 `$.getJSON` 方法被用来从 "resource" 中请求数据，然后判断结果。 `deepEqual` (有别于开始的 `equal`) 用来检测结果是否完全是我们所期望的 (比方说 `var a={}, b = {};` `// a==b → false` , 但是, 使用 `deepEqual` , `a==b` 为 `true` ) 。

因为 `$.getJSON` 是异步的, 我们调用 `stop()` , 然后运行代码, 再在 `callback` 结束的时候调用 `start()` , 告诉QUnit继续运行。

运行异步代码的时候, 如果没有使用 `stop()` , 会导致本意是其他测试的任意结果, 如 `passed` 或 `failed` .

### asyncTest

如果我们使用 `asyncTest()` 代替 `test()` , 我们就可以把 `stop()` 省掉。于是, 我们一眼就可以看出这是个异步测试。

```

1. asyncTest("async2", function() {
2.     $.getJSON("resource", function(result) {
3.         deepEqual(result, {
4.             status: "ok"
5.         });
6.         start();
7.     });
8. });

```



继续, 还有其他的API方法。

### expect

测试回调的时候, 无论是不是异步, 我们都不能确定回调会在某些时候被真正调用了。为了应付这种状况, 我们可以使用 `expect()` , 定义一个测试中我们期望的判断个数。这样就可以避免本应出现两个判别结果的, 结果一个通过后就停止的情况。

```

1. asyncTest("async3", function() {
2.     expect(1);
3.     $.getJSON("resource", function(result) {
4.         deepEqual(result, {
5.             status: "ok"
6.         });
7.         start();
8.     });
9. });

```

这里, `asyncTest()` 中, `expect` 被调用, 参数为 `1` , 相对于告诉QUnit这里的测试应该只有一次判定。这有什么好处呢? 当我们同时做多个异步测试的时候, 可能其他的测试囧屁被破坏, 然后也弄出了一个判定, 此时 `expect` 就可以捕获这个问题。

`expect()` 貌似有个取巧的设置方法，就是给 `test()` 或 `asyncTest()` 的第二个参数传递一个数字，例如，上面的代码可以写成：

```
1. asyncTest("async4", 1, function() {
2.     $.getJSON("resource", function(result) {
3.         deepEqual(result, {
4.             status: "ok"
5.         });
6.         start();
7.     });
8. });
```

是不是应该放demo啦。🐼

稍安勿躁，还有一部分，介绍完了一起放出来。

### 异步信号

如果测试的结束点多个-多个回调的顺序随机-我们可以使用QUnit内置信号。调用 `stop()` 与之后调用的 `start()` 同数目，则Qunit会继续运行直到 `stop()` 增加的计数被 `start()` 减少会 `0`。

```
1. test("async semaphore", function() {
2.     stop();
3.     stop();
4.     $.getJSON("resource", function(result) {
5.         equal(result.status, "ok");
6.         start();
7.     });
8.     $.getJSON("resource", function(result) {
9.         equal(result.status, "ok");
10.        start();
11.    });
12. });
```

说好的demo呢？🐼

真\*\*猴急，您可以狠狠的点击[这里](#)：[上面所有异步测试集合demo](#) 源代码右键可查看。

## 六、更多的测试判断

`equal` 和 `deepEqual` 已经在前面登台亮相了。实际上，QUnit提供了远不止这几个判断，比方说 `strict`，使用和 `equal` 一样，除了使用的是JavaScript中的 `"==="` 严格对比操作符。当对比不同类型值的时候，例如数值和字符串比较，`equal` 就有用武之地了，这样你的测试就不会死在 `0` 等于 `"0"` 上。

有正必有邪，因此，跟 `equal`，`deepEqual` 和 `strictEqual` 相反，这里还有 `notEqual`，`notDeepEqual`，`notStrictEqual`。同样的规则，不一样的结果。

### raises

除了这些，还有 `raises` (类似CommonJS中的 `throws`，不过没有使用保留关键字)。你传递一个 `function` 作为第一个参数，其会被QUnit中的 `try-catch` 语句中调用。如果这个函数抛



出异常，则我们认为通过- `pass`，否则不通过- `fail`。你还可以使用第二个参数检测抛出的异常。可以是正则表达式，或者是构造函数（使抛出错误会 `instanceof` 这个构造函数）检测，或者是一个返回 `true` 或 `false` 的函数（此函数的第一个参数就是抛出的异常）。

```
1. test("raises", function() {
2.     function CustomError() {}
3.     raises(function() {
4.         throw new CustomError();
5.     }, CustomError, "must throw error to pass");
6. });
```

上面示例显示了 `raises` 如何通过传递预期的构造函数引用作为第二个参数来检测构造函数抛出的异常。结果如下图，通过：



之所以会有上面的结果，是因为抛出的错误 `instanceof CustomError` 为 `true`。您可以狠狠地点击这里：[抛出异常检测demo](#)



## 七、自定义判断

就是让我们直接使用 `QUnit.push()` 进行封装自定义一些判断方法, 而不是上面的 `equal` 之类。`QUnit.push()` 与 `equal` 等关系，就好比jQuery中 `$.ajax()` 方法和 `$.post()/$.get()` 之间的关系。下面例子就是使用 `QUnit.push()` 自定义的一个验证DOM的某些属性、包括子元素前后是否发生变化的方法。

```
1. function domEqual( selector, modifier, message ) {
2.     var attributes = ["class", "role", "id", "tabIndex", "aria-activedescendant"];
3.
4.     function extract(value) {
5.         var result = {};
6.         result.nodeName = value[0].nodeName;
7.         $.each(attributes, function(index, attr) {
8.             result[attr] = value.attr(attr);
9.         });
10.        result.children = [];
11.        var children = value.children();
12.        if (children.length) {
13.            children.each(function() {
14.                result.children.push(extract($(this)));
15.            });
16.        } else {
17.            result.text = value.text();
18.        }
19.        return result;
20.    }
21.    var expected = extract($(selector));
22.    modifier($(selector));
23.
24.    var actual = extract($(selector));
25.    QUnit.push( QUnit.equiv(actual, expected), actual, expected, message );
26. }
```

上面代码虽然蛮多，其实要做的事情不难理解。检测选择器 `selector` 对应的元素 `attributes` 中的属性以及子元素是否在 `modifier()` 这个方法执行之后发生了改变。下面就是测试代码（需要用到jQuery UI）：

```
1. test("destroy", function() {
2.     domEqual("#autocomplete", function() {
3.         $("#autocomplete").autocomplete().autocomplete("destroy");
4.     });
5. });
```

在页面上放了个HTML为 `<input id="autocomplete">` 的代码，结果如下图：

貌似 `modifier` 执行后，Autocomplete插件将文本框的 `className` 改成了空字符串，导致文本框前后变化不相等了，于是检测结果为 `failed`！如果默认HTML是 `<input id="autocomplete" class="">`，则结果是 `passed`！您可以狠狠地点击这里：[autocomplete下自定义domEqual方法检测demo](#)

## 八、使用QUnit做开发

我勒个去，怎么还有这么多内容，我要偷工减料。



您可能注意到结果输出面板上有个不太明显的 `Rerun` 按钮，意思是“重新运行”，页面本来就可以刷新，还需要 `Rerun` 做什么呢？在实际测试的时候，测试面板肯定有很多结果的，有时候，我们可能会对某个测试结果很关心（比方说测试未通过），我们就可以点击 `Rerun` 按钮，于是，页面就只会检测该部分的代码是否可以通过了。例如，我点击本文[第5个demo](#)的第4个 `Rerun` 按钮，则：

通过格式化的 `url` 地址筛选需要运行的检测代码，这里，就指检测了第4个 `Rerun` 按钮对应的代码。

调试工具

在[第3个demo](#)中，您可能注意到了这个：

这些就是QUnit的调试工具，至于干嘛用的，偷工减料，不解释，(\*^\_\_^\*) 嘻嘻……

伙计，这可是煤油灯，不能省的，再辛苦下呗~~😓

好吧，内心的另一个我在召唤，加油！😓

- **Hide passed tests**

“隐藏通过的测试”，很好理解，勾选后，通过的测试就不显示了，如下[第3demo](#)截图：

- **Check for Globals**

“全局检测”，如果勾选此项，在运行测试之前，QUnit会枚举 `window` 所有属性，然后与运行结束之后的 `window` 做比较，如果前后不一样，哦呵呵，就会显示不通过- `failed`，以及显示“引入或缺失全局属性”。该工具作用在于可以很容易被发现判断导致全局变量的引入。

- **No try-catch**


“不要 `try-catch`”，选中则意味着QUnit会在 `try-catch` 语句外运行回调，此时，如果测试出现异常，测试就会停止。这有什么用呢？要知道，有些浏览器的调试工具是相当弱的，尤其IE6，一个未处理的异常要比捕获的异常可以提供更多的信息。即使再次抛出，由于JavaScript不擅长异常处理，原来的堆栈跟踪在大多数浏览器里都丢失了。

这显然是一个有着特殊目的的工具。然而，一旦你遇到一个异常，同时无法追溯真正错误代码的时候，这个工具就显得很有用了。

## 九、更高效的测试驱动开发

对于测试驱动开发(TDD – [test-driven development](#))而言，QUnit是相当不错的插件。然而，如果我们稍作调整，Qunit会变得更加有用，无论是结果还是速度。

假设现在有一个巨大的测试套件，全部测试完毕需要10秒钟，然后某处失败了，改好后，你重新测试，又是10秒钟，结果又发现一个错误，可能就是30秒甚至1分钟，所谓测试驱动开发难免沦为鸡肋。虽然 `Rurun` 过滤可以有针对性测试，但是，双拳难敌四手，错误很多的话根本应付不来。

那该怎么办呢？

这种情形，我们可以借助调试工具中的**Hide passed tests**这个选项提高开发效率。

看上面的介绍，这个工具很一般啊，这里为什么变超人了呢？

其实**Hide passed tests**可不是隐藏显示那么简单，其使用了HTML5 `sessionStorage` 技术，在不算离谱的浏览器下，会记住之前未通过的测试，然后页面重新载入的时候只测试之前那部分为通过的代码。显然，就不会出现上面那种改一下10秒跑一遍，改一下10秒钟跑一遍的情况。

## 十、回答开始的问题、结语

QUnit相关知识整理下来，我就可以很肯定地回答一开始抛出的问题：单元测试的必要性？答案是：有必要。

实际上，虽然我之前没有接触过单元测试相关的知识，但是，自己在实际写组件、方法的时候都是做过单元测试的，但是，属于一种业余的做法。比较好的单元测试应该是以代码去验证代码，测试代码保留，以便后续可以继续使用；而自己，纯粹是伪造各种可能的数据，然后去跑代码，是否报错之类，然后跑通之后，之间的测试数据什么的都删掉了，只剩下真正需要的运行代码。

实际上这是有问题的：

1. 如果这个方法后期内部逻辑有大改，回头测试的时候，自己岂不是要重新再造一遍数据，哦，我宝贵的时间啊~~；
2. 如果这个方法后期内部逻辑小改，实际测试的时候，精力成本等原因，往往就只会测试修改的这部分。很可能就会导致回归错误（A处改动触发B处的旧bug）。如果保留有完整的单元测试代码，因为想省事而出现的回归错误显然就无所遁形了！

可见，就个人而言，单元测试的思想是有的，但是，单元测试的做法有问题。借助类似Qunit这样的工具是很好的选择。因此，接下来，我会在团队中推广单元测试。磨刀不误砍柴工，大家权衡成本收益等，可以在关键方法（如果刚接触，没必要所有的方法）上试试单元测试，获取，你会喜欢上它的。

走遍万水千山，终于.....看到了一家渔具店，故事结束了。由于路上走得匆忙，沿途可能踩到了花花草草，欢迎指正，欢迎交流。感谢您一直阅读到此处！

《CSS世界》签名版独家发售，包邮，可指定寄语，点击显示购买码

(本篇完) // 想要打赏? 点击[这里](#)。有话要说? 点击[这里](#)。



« [梳理：提高前端性能方面的处理以及不足](#)

[JS滚轮事件\(mousewheel/DOMMouseScroll\)了解](#) »

#### 猜你喜欢

- [W3C DOM异常对象DOMException介绍](#)
- [jQuery boxy弹出层对话框插件中文演示及讲解](#)
- [新版无图片版zxbox jQuery弹出框插件](#)
- [最近整的MooTools库下Mbox弹框插件](#)
- [翻编-JavaScript有关的10个怪癖和秘密](#)
- [IE6下png背景不透明问题的综合拓展](#)
- [jQuery-很酷的弹出层效果js插件](#)
- [reflection.js-实现图片投影倒影效果js插件](#)
- [jQuery-单击文字或图片内容放大显示效果插件](#)
- [jQuery-实现图片的放大镜显示效果](#)
- [翻译 - 逐渐消失的Flash网站](#)

分享到: 1

标签: [jQuery](#), [QUnit](#), [单元测试](#), [回调函数](#), [报错](#)

#### 发表评论 (目前18条评论)

名称 (必须)

邮件地址(不会被公开) (必须)

网站

提交评论

1. 也许还十年说道:  
2016年11月7日 10:21

想问下博主，动真格的QUnit单元测试下面的那个例子，是把需测试的函数和测试用例放在同一个文件（也就是tests.js）中了吗，我一直以为这两个文件需要分开

[回复](#)



2. 也许还十年说道:  
2016年10月22日 10:43

博主,我的测试结果是这样:Uncaught Error: The global `test` is removed in QUnit 2.0.  
Details in our upgrade guide at <https://qunitjs.com/upgrade-guide-2.x/@> 1 ms  
Source:  
<http://localhost:63342/web/bsbs/%E8%8F%9C%E5%8D%95%E6%A0%8F1/qunit/qunit-2.0.1.js:2158>  
发不了图片就复制文字了求指点啊

[回复](#)



3. 从此就是我的学习偶像说道:  
2016年09月5日 23:14

关注你很久了，也学到了太多的东西 谢谢

[回复](#)



4. pp说道:  
2016年04月26日 23:20

谢谢分享 但是还是不知道怎么用在实际的项目中

[回复](#)



5. optimuszhang说道:  
2016年01月18日 15:09

受教了。

[回复](#)



6. 小影说道:  
2015年10月14日 16:11

想请教下，如果是频繁操作DOM的功能性代码，非异步不与后端交互的，这个有必要写测试用例么？那又该如何写？我一点思路都没有。。。

[回复](#)



7. 馍馍币说道:  
2014年08月8日 17:44

不错，不错，fff

[回复](#)



8. nimojs说道:  
2014年06月15日 16:48

感谢分享，很受用

[回复](#)



9. **bugknighttyp**说道:  
2014年01月29日 11:33

开启一个module测试后，他后边的所有的测试用例都归类到这个模块下了，那如果我想在某一个测试用例前终止模块的话，怎么做？

[回复](#)



10. **红色石头**说道:  
2013年09月5日 16:38

先mark一下，以后会用到的，问下你是做测试的吗？

[回复](#)



张 鑫旭说道:  
2013年09月5日 22:29

@红色石头 不是。

[回复](#)



11. **hellospume**说道:  
2013年06月6日 17:40

楼主所说基本是把qunit的官方文档翻译并给予解释哈，对于独立的函数，qunit当然是很容易进行测试。我想知道qunit对nodejs服务器程序怎么测试呢？对于那种需要等待硬件反馈某些信号回来然后又经过诸多传递的程序怎么测试呢？

[回复](#)



12. **gyrate**说道:  
2013年05月17日 09:21

好教程，lz会不会得精神分裂

[回复](#)



张 鑫旭说道:  
2013年05月17日 09:43

@gyrate 何出此言？

[回复](#)



**Bob**说道:  
2015年04月18日 12:32

估计说的是一人饰多角的对话吧

[回复](#)



13. **yangming**说道:  
2013年04月12日 18:37

“其实Hide passed tests可不是隐藏显示那么简单，其使用了HTML5 sessionStorage技术，在不算离谱的浏览器下”  
这个整体的意思是IE6、7、8等不支持html5 sessionStorage技术的，就用不上呗？

[回复](#)



张 鑫旭说道:  
2013年04月15日 09:31

@yangming IE8支持的。是的，IE6-IE7手短。

[回复](#)



14.

zwhu说道：

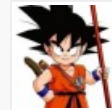
2013年04月10日 12:41

qunit确实不错啊！

我自己本身也用了大半年，不过基本只用OK来做测试判断。

而且用grunt来开发js就更不错了，集成了qunit, jshint, jsdoc, min等等使用grunt watch之后基本上只要保存js文件，这些插件都会跑一遍，很方便

[回复](#)



#### 最新文章

- » [常见的CSS图形绘制合集](#)
- » [粉丝群第1期CSS小测点评与答疑](#)
- » [分享三个纯CSS实现26个英文字母的案例](#)
- » [小tips: 纯CSS实现打字动画效果](#)
- » [CSS/CSS3 box-decoration-break属性简介](#)
- » [CSS :placeholder-shown伪类实现Material Design占位符交互效果](#)
- » [从天猫某活动视频不必要的3次请求说起](#)
- » [CSS vector-effect与SVG stroke描边缩放](#)
- » [CSS ::backdrop伪元素是干嘛用的？](#)
- » [周知：CSS -webkit-伪元素选择器不再导致整行无效](#)

#### 今日热门

- » [常见的CSS图形绘制合集](#) <sup>(193)</sup>
- » [未来必热：SVG Sprite技术介绍](#) <sup>(120)</sup>
- » [粉丝群第1期CSS小测点评与答疑](#) <sup>(115)</sup>
- » [HTML5终极备忘大全（图片版+文字版）](#) <sup>(93)</sup>
- » [让所有浏览器支持HTML5 video视频标签](#) <sup>(86)</sup>
- » [Selectivizr-让IE6~8支持CSS3伪类和属性选择器](#) <sup>(82)</sup>
- » [CSS3下的147个颜色名称及对应颜色值](#) <sup>(80)</sup>
- » [视区相关单位vw, vh..简介以及可实际应用场景](#) <sup>(77)</sup>
- » [写给自己看的display: flex布局教程](#) <sup>(76)</sup>
- » [小tips: 纯CSS实现打字动画效果](#) <sup>(76)</sup>

#### 今年热议

- » [《CSS世界》女主角诚寻靠谱一起奋斗之人](#) <sup>(76)</sup>
- » [不借助Echarts等图形框架原生JS快速实现折线图效果](#) <sup>(64)</sup>
- » [看，for..in和for..of在那里吵架！](#) <sup>(60)</sup>
- » [是时候好好安利下LuLu UI框架了！](#) <sup>(47)</sup>
- » [原来浏览器原生支持JS Base64编码解码](#) <sup>(35)</sup>
- » [妙法攻略：渐变虚框及边框滚动动画的纯CSS实现](#) <sup>(33)</sup>
- » [炫酷H5中序列图片视频化播放的高性能实现](#) <sup>(31)</sup>
- » [CSS scroll-behavior和JS scrollToView让页面滚动平滑](#) <sup>(30)</sup>
- » [windows系统下批量删除OS X系统.DS\\_Store文件](#) <sup>(26)</sup>
- » [写给自己看的display: flex布局教程](#) <sup>(26)</sup>

#### 猜你喜欢

- [W3C DOM异常对象DOMException介绍](#)

- jQuery boxy弹出层对话框插件中文演示及讲解
- 新版无图片版zxxbox jQuery弹出框插件
- 最近整的MooTools库下Mbox弹框插件
- 翻编-JavaScript有关的10个怪癖和秘密
- IE6下png背景不透明问题的综合拓展
- jQuery-很酷的弹出层效果js插件
- reflection.js-实现图片投影倒影效果js插件
- jQuery-单击文字或图片内容放大显示效果插件
- jQuery-实现图片的放大镜显示效果
- 翻译 - 逐渐消失的Flash网站