


Mocha.js官方文档翻译 —— 简单、灵活、有趣



Awey

关注

0.633

2016.10.08 16:10:31

字数 6,016

阅读 12,189

本次翻译时间为2016年9月底，目前Mocha的版本为3.1.0。
官方文档地址：<http://mochajs.org/>
—— Awey

1. 简介

Mocha是一个能够运行在Node和浏览器中的多功能的JavaScript测试框架，它让异步测试简单且有趣。Mocha连续地运行测试，并给出灵活而精确的报告，同时能够将错误精确地映射到测试用例上。它托管在[GitHub](#)上。

2. 支持者

觉得Mocha很有帮助？成为[支持者](#)并以每月捐赠的形式支持Mocha。

3. 赞助商

公司正在使用Mocha？询问你的上司或者市场部看看他们是否捐赠过Mocha。（如果捐赠过）你们公司的LOGO会展示在[npmjs.com](#)和我们的[github仓库](#)

4. 特性

- 支持浏览器
- 支持简单异步，包括promise
- 测试覆盖率报告
- 支持字符串比较
- 提供JavaScript API来运行测试
- 为持续集成等需求提供适当的退出状态
- non-ttys 自动检测和禁用颜色
- 异步测试超时
-（后略）

5. 目录

略。
本次翻译并未完全按照官方文档的文档结构进行。

6. 开始使用

6.1 安装

npm 全局安装：


百度智能云

云服务器3个月仅售16元

免费送12120元感恩福袋

立即秒杀

广告



Awey

总资产151 (约13.45元)

关注

Vue.js-组件化前端开发新思路
阅读 6,067

《SQL初学者指南》读书笔记
阅读 614

推荐阅读

- var 为什么会被 let 取代
阅读 8,595
- Promise的特性及实现原理
阅读 3,102
- 初步部署个人博客到github
阅读 108
- ES6特性第一部分
阅读 803
- 平凡的世界
阅读 25

百度智能云

1212岁末感恩季

云服务器3个月

仅售16元

立即注册 领12120元感恩福袋

广告

或者作为开发依赖安装在项目中：

```
1 | npm install mocha --save-dev
```

安装Mocha v3.0.0或者更新的版本，你需要v1.4.0或者更新版本的npm。此外，运行Mocha的Node版本不能低于v0.10

Mocha也能通过Bower安装，还可通过cdnjs进行引用。

6.2 起步

```
1 | npm install mocha
2 | mkdir test
3 | $EDITOR test/test.js # 或者使用你喜欢的编辑器打开
```

在编辑器中：

```
1 | var assert = require('assert')
2 | describe('Array', function () {
3 |   describe('#indexOf()', function() {
4 |     it('未找到值时应当返回-1', function () {
5 |       assert.equal(-1, [1, 2, 3].indexOf(4))
6 |     })
7 |   })
8 | })
```

回到命令行：

```
1 | Array
2 |   #indexOf()
3 |     ✓ 未找到值时应当返回-1
4 |
5 | 1 passing (9ms)
```

6.3 断言

Mocha允许你使用你喜欢的断言库。在之后的例子中，我们使用了Node中内置的断言模块——但通常情况下，只要它能抛出异常就行^[1]。这意味着你可以使用下列断言库：

- should.js - BDD风格贯穿始终
- expect.js - expect() 风格的断言
- chai - expect()、assert()和 should风格的断言
- better-assert - C风格的自文档化的assert()
- unexpected - “可扩展的BDD断言工具”

6.4 异步代码

```
1 describe('User', function () {
2   describe('#save()', function () {
3     it('应当正常保存', function () {
4       var user = new User('Luna')
5       user.save(function (err) {
6         if (err) done(err)
7         else done()
8       })
9     })
10  })
11 })
```

简便起见，`done()` 函数接受一个error参数，所以上面的代码可以这么写：

```
1 describe('User', function () {
2   describe('#save()', function () {
3     it('应当正常保存', function () {
4       var user = new User('Luna')
5       user.save(done)
6     })
7   })
8 })
```

Promise

有时，与其使用 `done()` 回调函数，你会想在你的异步代码中返回一个Promise^[3]，当你正在测试的API是返回一个Promise而不是使用回调时这会很有帮助：

```
1 beforeEach(function () {
2   return db.clear()
3     .then(function () {
4       return db.save([tobi, loki, jane])
5     })
6 })
7
8 describe('#find()', function () {
9   it('返回匹配的记录', function () {
10    return db.find({ type: 'User' }).should.eventually.have.length(3)
11  })
12 })
```

接下来的例子将会使用`chai-as-promised`来获得流畅的promise断言

在Mocha 3.0及更新的版本中，同时返回一个Promise和调用 `done()` 会导致一个异常，下面的代码是错误的：

```
1 const assert = require('assert')
2
3 it('应该结束这个测试用例', function (done) {
4   return new Promise(function (resolve) {
5     assert.ok(true)
6     resolve()
7   })
8   .then(done)
9 })
```

6.5 同步代码

当测试同步代码的时候，省略回调函数^[4]，Mocha就会自动执行下一条测试。

```
1 describe('Array', function () {
2   describe('#indexOf()', function () {
3     it('没有找到时应当返回-1', function () {
4       [1, 2, 3].indexOf(5).should.equal(-1)
5       [1, 2, 3].indexOf(0).should.equal(-1)
6     })
7   })
8 })
```

6.6 箭头函数

不建议在Mocha中使用箭头函数（“lambdas”）。由于（箭头函数特殊的）`this` 绑定语法，箭头函数无法访问Mocha的上下文。例如，下面的代码会因为使用了箭头函数而执行失败：

```
1 describe('my suit', () => {
2   it('my test', () => {
3     // 应当设置1000毫秒延迟，而不是执行失败
4     this.timeout(1000)
5     assert.ok(true)
6   })
7 })
```

<small>当然如果你不需要使用Mocha的上下文，使用lambdas就没有问题了。然而这样的做的结果是你的测试代码将难以重构</small>

6.7 钩子函数

Mocha默认使用“BDD”风格的接口，提供了 `before()`，`after()`，`beforeEach()` 和 `afterEach()` 四个钩子函数。这些函数可以用来（在测试前）做预处理工作或在测试后清理工作。

```
1 describe('hooks', function () {
2   before(function () {
3     // 在这个作用域的所有测试用例运行之前运行
4   })
5
6   after(function () {
7     // 在这个作用域的所有测试用例运行完之后运行
8   })
9
10  beforeEach(function () {
11    // 在这个作用域的每一个测试用例运行之前运行
12  })
13
14  afterEach(function () {
15    // 在这个作用域的每一个测试用例运行之后运行
16  })
17
18  // 测试用例
19 })
```

钩子函数的描述参数

所有的钩子在调用时都可以提供一个可选的“描述信息”的参数，以便在你的测试中更精确地定位错误。如果给一个钩子函数传入一个命名函数，当未提供“描述信息”参数的时候，这个命名函数的名称将被作为描述信息。

```
1 beforeEach(function () {
2   // beforeEach hook
3 })
4
5 beforeEach(function namedFun () {
6   // beforeEach: namedFun
7 })
8
9 beforeEach('一些描述信息', function () {
10   // beforeEach: 一些描述信息
11 })
```

异步钩子

所有钩子（`before()`，`after()`，`beforeEach()`，`afterEach()`）既可以是同步的也可以是异步的，（这一点上）它们的行为与普通的测试用例非常相似^[5]：

```
1 describe('连接', function () {
2   var db = new Connection,
3       tobi = new User('tobi'),
4       loki = new User('loki'),
5       jane = newUser('jane')
6
7   beforeEach(function (done) {
8     db.clear(function (err) {
9       if (err) return done(err)
10      db.save([tobi, loki, jane], done)
11    })
12  })
13
14  describe('#find()', function () {
15    it('返回匹配的记录', function (done) {
16      db.find({type: 'User'}, function (err, res) {
17        if (err) return done(err)
18        res.should.have.length(3)
19      })
20    })
21  })
22 })
```

全局钩子

你可以在任意（测试）文件中添加“全局”级别的钩子函数，例如，在所有 `describe()` 作用域之外添加一个 `beforeEach()`，它的回调函数会在所有的测试用例运行之前运行，无论（这个测试用例）处在哪个文件（这是因为Mocha有一个隐藏的 `describe()` 作用域，称为“根测试套件 root suite”）。

```
1 beforeEach(function () {
2   console.log('在所有文件的所有测试用例之前运行')
3 })
```

```
1 setTimeout(function () {
2   // 一些设置
3
4   describe('我的测试套件', function () {
5     // ...
6   })
7
8   run()
9 }, 5000)
```

6.8 挂起测试 (Pending Tests)

“Pending”——“有人最终会编写这些测试用例”——没有传入回调函数的测试用例^[7]：

```
1 describe('Array', function () {
2   describe('#indexOf()', function () {
3     // 挂起的测试用例
4     it('未找到时应当返回-1')
5   })
6 })
```

挂起的测试用例会在报告中出现“pending”状态。

6.9 独占测试

通过向测试套件或测试用例函数添加 `.only` 后缀，独占特性允许你只运行指定的测试套件或测试用例。下面是一个独占测试套件的例子：

```
1 describe('Array', function () {
2   describe.only('#indexOf()', function () {
3     // ...
4   })
5 })
```

<small>注意：所有嵌套（在 `.only` 套件中的）测试套件仍旧会运行</small>

下面是一个运行单个测试用例的例子：

```
1 describe('Array', function () {
2   describe('#indexOf()', function () {
3     it.only('除非找到否则返回-1', function () {
4       // ...
5     })
6
7     it('找到后应当返回下标', function () {
8       // ...
9     })
10  })
11 })
```

在v3.0.0版本以前，`only()` 使用字符串匹配来决定哪些测试需要执行。v3.0.0以后的版本 `only()` 可以使用多次来定义测试用例的子集去运行：

```
6
7     it.only('should return the index when present', function() {
8         // 这个测试用例也会运行
9     })
10
11     it('should return -1 if called with a non-Array context', function() {
12         // 这个测试用例不会运行
13     })
14 })
15 })
```

你也可以选择多个测试套件：

```
1 describe('Array', function() {
2     describe.only('#indexOf()', function() {
3         it('should return -1 unless present', function() {
4             // 这个测试用例会运行
5         })
6
7         it('should return the index when present', function() {
8             // 这个测试用例也会运行
9         })
10    })
11
12    describe.only('#concat()', function () {
13        it('should return a new Array', function () {
14            // 这个测试用例也会运行
15        })
16    })
17
18    describe('#slice()', function () {
19        it('should return a new Array', function () {
20            // 这个测试用例不会运行
21        })
22    })
23 })
```

但测试会存在优先级^[8]：

```
1 describe('Array', function() {
2     describe.only('#indexOf()', function() {
3         it.only('should return -1 unless present', function() {
4             // 这个测试用例会运行
5         })
6
7         it('should return the index when present', function() {
8             // 这个测试用例不会运行
9         })
10    })
11 })
```

<small>注意，如果提供了钩子函数，钩子函数仍会执行</small>

<small>注意不要把 `.only` 提交到版本控制上，除非你明确知道你在做什么</small>

6.10 跳过测试

这个功能是 `only()` 的反面。通过后缀 `skip()` 就可以让Mocha忽略这个测试套件或测试用例。所有

写下你的评论...

评论4

赞30

...

```
1 describe('Array', function() {
2   describe.skip('#indexOf()', function() {
3     // ...
4   })
5 })
```

下面是一个跳过测试用例的例子：

```
1 describe('Array', function() {
2   describe('#indexOf()', function() {
3     it.skip('should return -1 unless present', function() {
4       // 这个测试用例不会运行
5     })
6
7     it('should return the index when present', function() {
8       // 这个测试用例会运行
9     })
10  })
11 })
```

最佳实践：使用 `skip()` 而不是直接将测试注释掉

你也可以使用 `this.skip()` 在运行时跳过测试。如果测试需要的环境或配置没办法提前检测，可以考虑使用运行时跳过。例如：

```
1 it('应该仅在正确的环境配置中测试', function () {
2   if (/* 测试环境正确 */) {
3     // 编写断言
4   } else {
5     this.skip()
6   }
7 })
```

因为这个测试什么也没做^[9]，它会被报告为 `passing`。

最佳实践：不要什么也不做^[10]！一个测试应当编写断言或者使用 `this.skip()`

如果想以这种方式跳过多个测试^[11]，可以在一个 `before()` 钩子函数中调用 `this.skip()`：

```
1 before(function() {
2   if (/* check test environment */) {
3     // setup code
4   } else {
5     this.skip()
6   }
7 })
```

在Mocha v3.0.0版本以前，钩子函数和异步测试中不支持 `this.skip()`

这个特性会重新运行 `beforeEach()` / `afterEach()` 钩子，但不会运行 `before()` / `after()` 钩子。

注意：下面的例子使用了Selenium webdriver（为Promise链式调用改写了Mocha的全局钩子）。

```
1 describe('retries', function() {
2   // 测试套件中的所有测试用例将被重试4次
3   this.retries(4)
4
5   beforeEach(function () {
6     browser.get('http://www.yahoo.com');
7   })
8
9   it('should succeed on the 3rd try', function () {
10    // 指定这个测试用例仅重试2次
11    this.retries(2)
12    expect($('.foo').isDisplayed()).to.eventually.be.true
13  })
14 })
```

6.12 动态生成测试

可以使用 `Function.prototype.call` 和函数表达式来定义测试套件和测试用例，以动态生成测试而不需要其它的特殊语法——简单的JavaScript就能用于实现你可能在其它测试框架中见到过的类似“参数化”测试的功能。

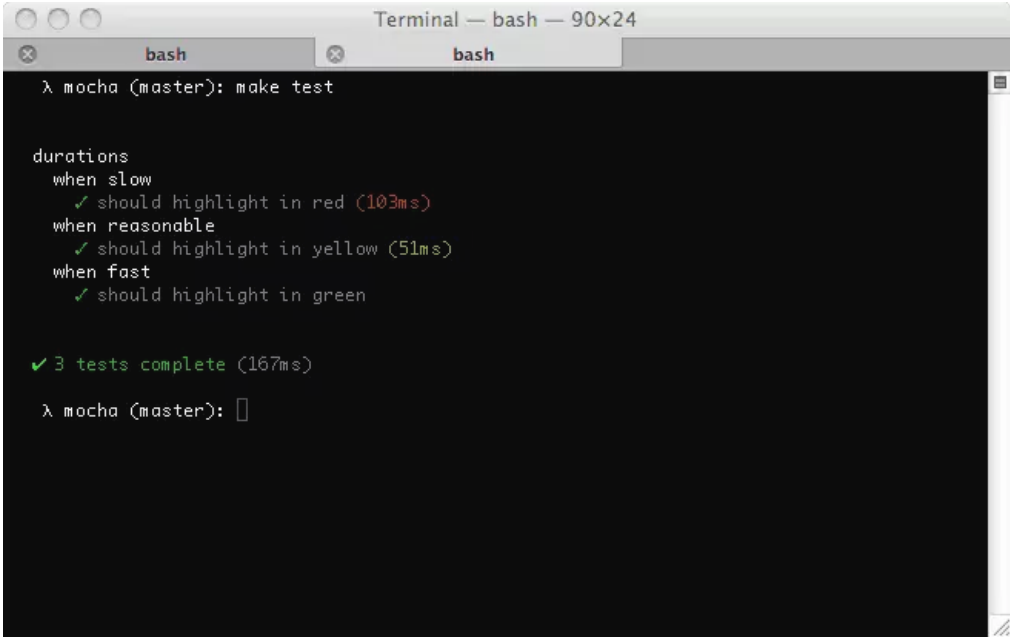
例如：

```
1 var assert = require('chai').assert
2
3 function add() {
4   return Array.prototype.slice.call(arguments).reduce(function(prev, curr) {
5     return prev + curr
6   }, 0)
7 }
8
9 describe('add()', function() {
10   var tests = [
11     {args: [1, 2],      expected: 3},
12     {args: [1, 2, 3],   expected: 6},
13     {args: [1, 2, 3, 4], expected: 10}
14   ]
15
16   tests.forEach(function(test) {
17     it('correctly adds ' + test.args.length + ' args', function() {
18       var res = add.apply(null, test.args)
19       assert.equal(res, test.expected)
20     })
21   })
22 })
```

上面的代码将会生成一个带有三个测试用例的测试套件：

```
1 $ mocha
2
3 add()
4   ✓ correctly adds 2 args
5   ✓ correctly adds 3 args
6   ✓ correctly adds 4 args
```

许多测试报告都会显示测试耗时，并且标记出那些耗时较长的测试，就像下面的报告显示的那样：



测试耗时

你可以使用 `slow()` 方法来定义到底多久才算“耗时较长”：

```
1 describe('something slow', function() {
2   this.slow(10000)
3
4   it('它的耗时应该足够我去做个三明治了', function() {
5     // ...
6   })
7 })
```

6.14 测试超时

套件级别

套件级别的超时应用于整个测试套件，你也可以通过 `this.timeout(0)` 来取消超时限制。如果没有覆盖这个值的话^[12]，所有嵌套的测试套件和测试用例都会继承这个超时限制。

```
1 describe('a suite of tests', function() {
2   this.timeout(500)
3
4   it('应当不超过500毫秒', function(done){
5     setTimeout(done, 300)
6   })
7
8   it('也应当不超过500毫秒', function(done){
9     setTimeout(done, 250)
10  })
11 })
```

用例级别

写下你的评论...

```
3   setTimeout(done, 500)
4 })
```

钩子级别

当然也可以设置钩子级别的超时：

```
1 describe('一个测试套件', function() {
2   beforeEach(function(done) {
3     this.timeout(3000); // 一个用时很长的环境设置操作。
4     setTimeout(done, 2500)
5   })
6 })
```

同样，使用 `this.timeout(0)` 来取消超时限制

在v3.0.0或更新的版本中，给 `this.timeout()` 传递一个大于[最大延迟值](#)的参数会让超时限制失效

6.15 差异比较

Mocha支持断言库抛出的 `AssertionErrors` 的两个属性 `err.expected` 和 `err.actual`。Mocha会尝试显示期望（的代码）和断言库真正看到的的代码之间的差异。这里有一个“string”差异的例子：

```
mocha — bash

1) diffs should display a word diff for large strings:

  actual expected

   1 | body {
   2 |   font: "Helvetica Neue", Helvetica, arial, sans-serif;
   3 |   background: black;
   4 |   color: #fffwhite;
   5 | }
   6 |
   7 | a {
   8 |   color: blue;
   9 | }
  10 |
  11 | foo {
  12 |   bar: 'baz';
  13 | }

at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:302:10)
at Context.<anonymous> (/Users/tj/projects/mocha/test/acceptance/diffs.js:22:18)
at Test.run (/Users/tj/projects/mocha/lib/runnable.js:156:32)
at Runner.runTest (/Users/tj/projects/mocha/lib/runner.js:272:10)
at /Users/tj/projects/mocha/lib/runner.js:316:12
at next (/Users/tj/projects/mocha/lib/runner.js:199:14)
at /Users/tj/projects/mocha/lib/runner.js:208:7
at next (/Users/tj/projects/mocha/lib/runner.js:157:23)
at Array.0 (/Users/tj/projects/mocha/lib/runner.js:176:5)
at EventEmitter._tickCallback (node.js:192:40)

make: *** [test-unit] Error 1
λ mocha (feature/diffs):
```

“string”差异

7. 命令行

写下你的评论...

评论4 赞30 ...

```
3
4  Commands:
5
6  init <path>  initialize a client-side mocha setup at <path>
7
8  Options:
9
10 -h, --help          显示使用帮助
11 -V, --version       显示版本信息
12 -A, --async-only    强制所有测试带有回调（异步）或返回一个promise
13 -c, --colors        强制启用颜色
14 -C, --no-colors     强制关闭颜色
15 -G, --growl         启用弹出消息
16 -O, --reporter-options <k=v,k2=v2,...> 测试报告工具详细设置
17 -R, --reporter <name> 指定测试报告工具
18 -S, --sort          测试文件排序
19 -b, --bail          第一次测试不通过立即结束测试
20 -d, --debug         开启node的debugger模式，同node --debug
21 -g, --grep <pattern> 只运行匹配<pattern>的测试
22 -f, --fgrep <string> 只运行包含<string>的测试
23 -gc, --expose-gc    暴露gc扩展
24 -i, --invert        反转--grep 和--fgrep 匹配
25 -r, --require <name> 加载指定模块
26 -s, --slow <ms>     以毫秒为单位定义“慢” 测试门槛 [75]
27 -t, --timeout <ms>  以毫秒为单位设置测试用例超时时间 [2000]
28 -u, --ui <name>     指定用户接口（bdd|tdd|qunit|exports）
29 -w, --watch         监测文件变动
30 --check-leaks       检查全局变量泄露
31 --full-trace        显示完整的跟踪堆栈
32 --compilers <ext>:<module>,... 使用指定的模块编译文件
33 --debug-brk         在首行启用node的debugger断点
34 --globals <names>  allow the given comma-delimited global [names]（没办法强制
35 --es_staging        开启所有过时的特性
36 --harmony<_classes,_generators,...> all node --harmony* flags are available
37 --preserve-symlinks 命令模块加载器在解析和缓存模块时保留符号链接
38 --icu-data-dir      包括ICU数据
39 --inline-diffs      在行内显示实际/预期的字符差异
40 --interfaces        显示可用的接口（bdd|tdd|qunit|exports）
41 --no-deprecation    禁用警告
42 --no-exit           请求一个彻底的事件循环终止：Mocha不会调用 process.exit
43 --no-timeouts       禁用超时，隐含 --debug
44 --opts <path>       指定选项路径
45 --perf-basic-prof    enable perf linux profiler (basic support)
46 --prof             记录统计分析信息
47 --log-timer-events  记录包含外部回调的时间点
48 --recursive        包含子目录
49 --reporters         显示可用的测试报告工具
50 --retries <times>   设置重试未通过的测试用例的次数
51 --throw-deprecation 当使用了废弃的方法时抛出异常
52 --trace            追踪函数借调
53 --trace-deprecation 显示废弃的跟踪堆栈
54 --use_strict        强制严格模式
55 --watch-extensions <ext>,... --watch 上附加的监控扩展
56 --delay            等待异步套件定义
```

`-w` , `--watch`

初始化后，监测文件变动运行测试

`--compilers`

CoffeeScript不再被直接支持。这类预编译语言可以使用相应的编译器扩展来使用，比如CS1.6：

`--compilers coffee:coffee-script` 和CS1.7+：`--compilers coffee:coffee-script/register`

`babel-register`

如果你的gcc模块只以...为扩展名的...你可以...然后

写下你的评论...

评论4

赞30

...

只对首个异常感兴趣？使用 `--bail`

`-d` , `--debug`

开启node的调试模式，这会用 `node debug <file ...>` 来执行你的脚本，允许你逐行调试代码并用 `debugger` 声明来打断点。注意 `mocha debug` 和 `mocha --debug` 的区别：`mocha debug` 会启动node内置的debug客户端，`mocha --debug` 则允许你使用其它调试工具——比如Blink Developer Tools。

`--globals <name>`

接受一个以逗号分隔的全局变量名，例如，如果你的应用有意暴露一个全局变量名 `app` 或 `YUI`，你可能就会使用 `--globals app,YUI`。它还接受通配符。`--globals '*bar'` 会匹配 `foobar`, `barbar` 等。你也可以简单地传入 `*` 来忽略所有全局变量。

`check-leaks`

在运行测试时，Mocha默认不会检查全局变量泄露，可以使用 `--check-leaks` 来开启这一功能，使用 `--globals` 来指定接受的全局变量比如 `--globals jQuery,MyLib`。

`-r` , `--require <module-name>`

`--require` 选项对诸如`should.js`一类的库很有用，所以你可以使用 `--require should` 而不是在每一个测试文件中都调用 `require('should')`。需要注意的是因为 `should` 是增强了 `Object.prototype` 所以可以正常使用，然而假如你希望访问某模块的输出你就只能 `require` 它们了，比如 `var should = require('should')`。此外，还可以使用相对路径，比如 `--require ./test/helper.js`

`-u` , `--ui <name>`

`--ui` 选项让你指定想使用的接口，默认为“bdd”。

`-R` , `--reporter <name>`

`--reporter` 选项允许你指定希望使用的报告器，默认为“spec”。这个标记也可以用来使用第三方的报告器。例如，如果你 `npm install mocha-locv-reporter`，你可以 `--reporter mocha-locv-reporter`。

`-t` , `--timeout <ms>`

指定测试用例超时时间，默认为2秒。你可以传入毫秒数或者一个带 `s` 单位后缀的秒数进行覆盖，例如 `--timeout 2s` 和 `--timeout 2000` 是等价的。

`s` , `--slow <ms>`

指定“慢”测试阈值，默认为75毫秒。Mocha用这个去高亮那些耗时过长的测试。

`-g` , `--grep <pattern>`

指定 `--grep` 选项让Mocha只运行匹配 `<pattern>` 的测试，`<pattern>` 将会作为正则表达式进行解析。

假如，像下面的片段那样，你有一些“api”相关的测试和一些“app”相关的测试；则前者可以使用 `--grep api` 来运行，后者可使用 `--grep --app` 来运行

```
1 | describe('api', function() {
```

```
7  })
8
9  describe('app', function() {
10    describe('GET /users', function() {
11      it('respond with an array of users', function() {
12        // ...
13      })
14    })
15  })
```

8.接口^[13]

Mocha的“接口”系统允许开发者选择习惯的风格或DSL。Mocha有BDD, TDD, Exports, QUnit和Require风格的接口。

8.1 BDD

BDD接口提供 `describe()`, `context()`, `it()`, `specify()`, `before()`, `after()`, `beforeEach()` 和 `afterEach()`。

`context()` 只是 `describe()` 的别名, 二者表现也是一致的; 它只是为了让测试可读性更高。同样 `specify()` 也是 `it()` 的别名。

前文所有的示例都是使用BDD接口编写的

```
1  describe('Array', function() {
2    before(function() {
3      // ...
4    })
5
6    describe('#indexOf()', function() {
7      context('when not present', function() {
8        it('should not throw an error', function() {
9          (function() {
10            [1,2,3].indexOf(4)
11          }).should.not.throw()
12        })
13        it('should return -1', function() {
14          [1,2,3].indexOf(4).should.equal(-1);
15        })
16      })
17      context('when present', function() {
18        it('should return the index where the element first appears in the array', function() {
19          [1,2,3].indexOf(3).should.equal(2)
20        })
21      })
22    })
23  })
```

8.2 TDD

TDD接口提供 `suite()`, `test()`, `suiteSetup()`, `suiteTeardown()`, `setup` 和 `teardown()` :

```
1  suite('Array', function() {
2    setup(function() {
3      // ...
4    })
```

```
10 | })
11 | })
```

8.3 EXPORTS

EXPORTS接口很像Mocha的前身`expresso`，键值 `before`，`after`，`beforeEach`，`afterEach` 是特殊用例，对象类型的属性值是测试套件，方法类型的属性值是测试用例：

```
1 | module.exports = {
2 |   before: function() {
3 |     // ...
4 |   },
5 |
6 |   'Array': {
7 |     '#indexOf()': {
8 |       'should return -1 when not present': function() {
9 |         [1,2,3].indexOf(4).should.equal(-1)
10 |       }
11 |     }
12 |   }
13 | }
```

8.4 QUNIT

类`QUnit`接口与QUnit的“扁平化”外观相匹配^[14]，测试套件只需要简单地在测试用例之前定义就行。和TDD类似，它使用 `suite()` 和 `test()`，但又类似于BDD，也包含了 `before()`，`after()`，`beforeEach()` 和 `afterEach()`。

```
1 | function ok(expr, msg) {
2 |   if (!expr) throw new Error(msg)
3 | }
4 |
5 | suite('Array')
6 |
7 | test('#length', function() {
8 |   var arr = [1,2,3]
9 |   ok(arr.length == 3)
10 | })
11 |
12 | test('#indexOf()', function() {
13 |   var arr = [1,2,3]
14 |   ok(arr.indexOf(1) == 0)
15 |   ok(arr.indexOf(2) == 1)
16 |   ok(arr.indexOf(3) == 2)
17 | })
18 |
19 | suite('String')
20 |
21 | test('#length', function() {
22 |   ok('foo'.length == 3)
23 | })
```

8.5 REQUIRE

`require` 风格接口允许你直接用 `require` 语句引入 `describe` 等函数并在任意位置使用它们。当你希望在你的测试中禁用全局变量时这也会很有用。

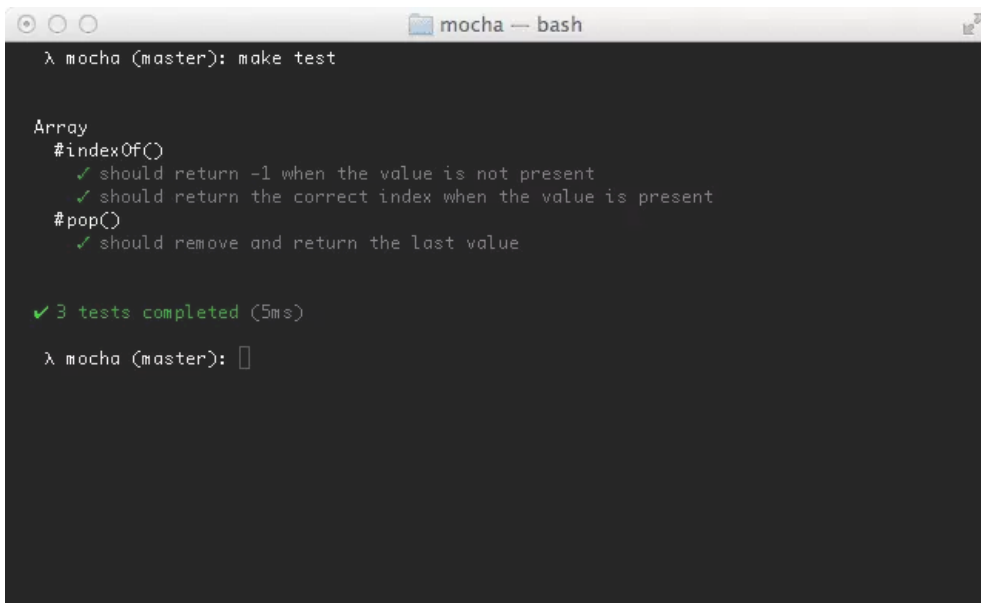
```
3 var assertions = require('mocha').it
4 var assert = require('chai').assert
5
6 testCase('Array', function() {
7   pre(function() {
8     // ...
9   });
10
11   testCase('#indexOf()', function() {
12     assertions('should return -1 when not present', function() {
13       assert.equal([1,2,3].indexOf(4), -1)
14     })
15   })
16 })
```

9. 测试报告

Mocha的测试报告与命令行窗口适配，且当标准输出串口没有关联到打印机时始终禁用ANSI-escape颜色。

9.1 SPEC

这是默认的测试报告。“SPEC”测试报告输出与测试用例一致的嵌套视图。



```
mocha — bash
λ mocha (master): make test

Array
  #indexOf()
    ✓ should return -1 when the value is not present
    ✓ should return the correct index when the value is present
  #pop()
    ✓ should remove and return the last value

✓ 3 tests completed (5ms)

λ mocha (master):
```

spec 测试报告

带有失败状态的spec 测试报告

9.2 Dot Matrix

Dot Matrix（或者Dot）测试报告使用一串简单字符来表示测试用例，失败的（failing）以红色叹号（!）表示，挂起的（pending）以蓝色逗号（,）表示，长时的以黄色表示。当你希望最小化输出的时候这就很好。

```
$ mocha test/ --reporter dot

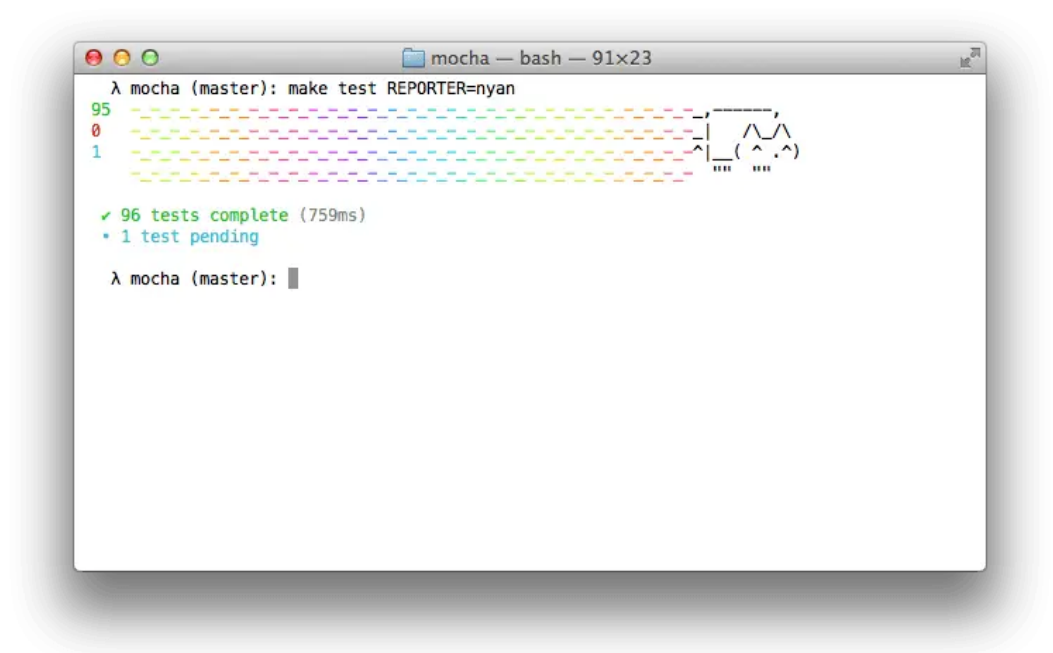
.....,
.....!
..

176 passing (549ms)
1 pending
1 failing
```

dot matrix 测试报告

9.3 NYAN

NYAN测试报告就是你想的那样（一只猫）：



js nyan cat 测试报告

9.4 TAP

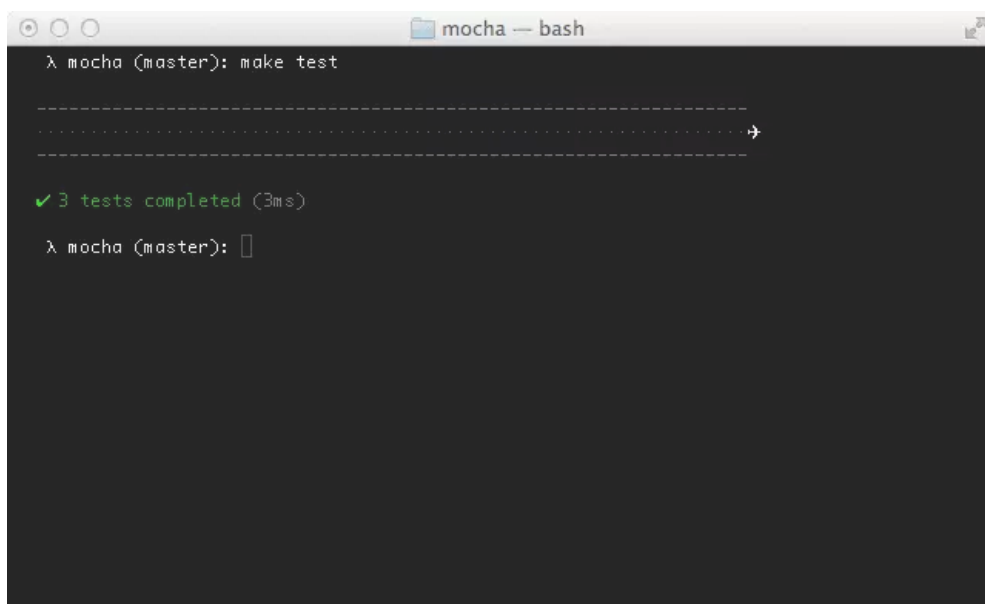
TAP测试报告的输出很适合Test-Anything-Protocol的用户。

```
λ mocha (master): make test
1..3
ok 1 Array #indexOf() should return -1 when the value is not present
ok 2 Array #indexOf() should return the correct index when the value is present
ok 3 Array #pop() should remove and return the last value
λ mocha (master):
```

test anything protocol

9.5 Landing Strip

landing Strip (landing) 测试报告是一个非正式的测试报告器，它用unicode字符模仿了飞机降落的情景。



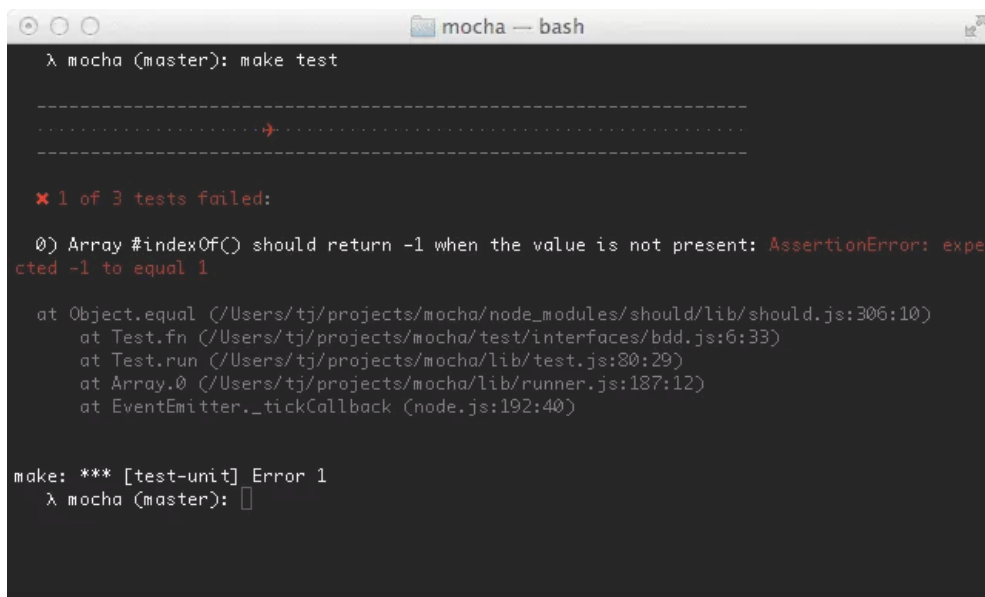
```
mocha — bash
λ mocha (master): make test

-----
.....
-----

✓ 3 tests completed (3ms)

λ mocha (master):
```

landing strip plane 测试报告



```
mocha — bash
λ mocha (master): make test

-----
.....
-----

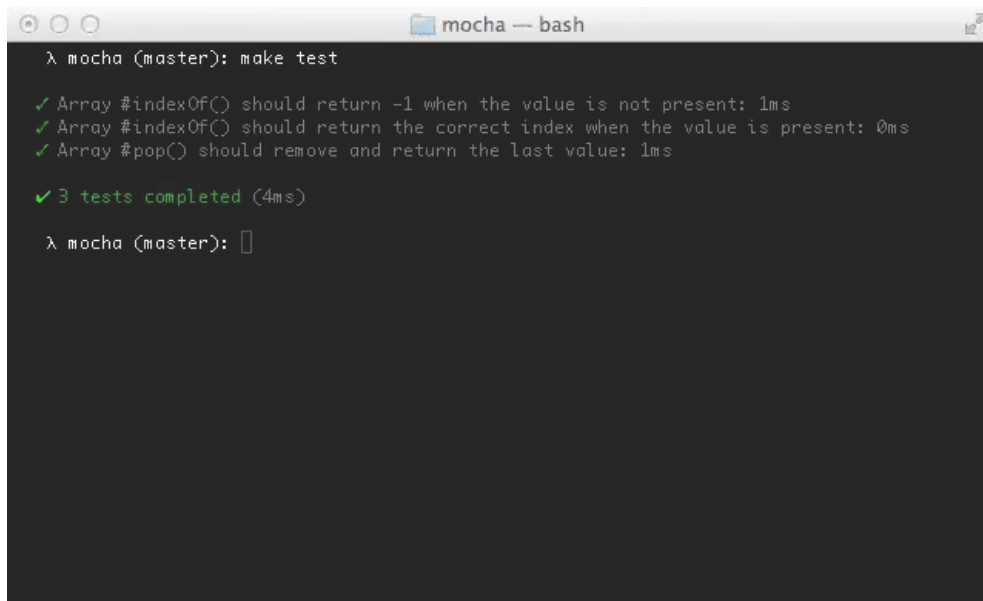
✖ 1 of 3 tests failed:

0) Array #indexOf() should return -1 when the value is not present: AssertionError: expected -1 to equal 1

at Object.equal (/Users/tj/projects/mocha/node_modules/should/lib/should.js:306:10)
at Test.fn (/Users/tj/projects/mocha/test/interfaces/bdd.js:6:33)
at Test.run (/Users/tj/projects/mocha/lib/test.js:80:29)
at Array.0 (/Users/tj/projects/mocha/lib/runner.js:187:12)
at EventEmitter._tickCallback (node.js:192:40)

make: *** [test-unit] Error 1
λ mocha (master):
```

list测试报告输出一个测试用例通过或失败的简洁列表，并在底部输出失败用例的详情。



```
λ mocha (master): make test

✓ Array #indexOf() should return -1 when the value is not present: 1ms
✓ Array #indexOf() should return the correct index when the value is present: 0ms
✓ Array #pop() should remove and return the last value: 1ms

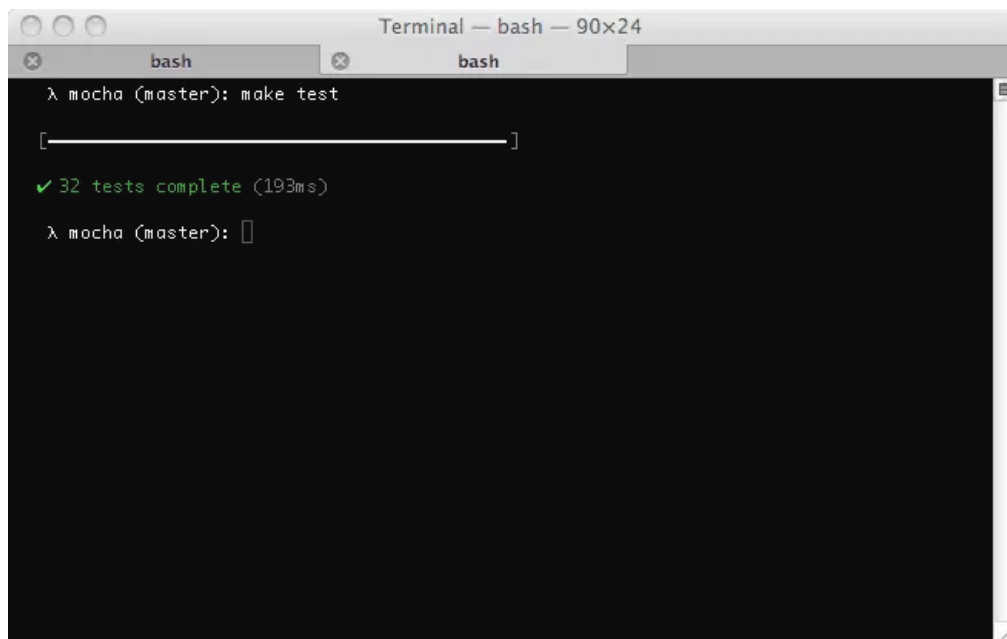
✓ 3 tests completed (4ms)

λ mocha (master):
```

list测试报告

9.7 PROGRESS

progress测试报告展示一个简单进度条。



```
Terminal — bash — 90x24

λ mocha (master): make test

[ ]

✓ 32 tests complete (193ms)

λ mocha (master):
```

progress bar

9.8 JSON

JSON测试报告在测试完成后输出一个大JSON对象。

```
"end":{"2011-11-19T20:27:09.042Z","duration":2},"tests":[{"title":"should return -1 when the value is not present","fullTitle":"Array #indexOf() should return -1 when the value is not present","duration":1}, {"title":"should return the correct index when the value is present","fullTitle":"Array #indexOf() should return the correct index when the value is present","duration":0}, {"title":"should remove and return the last value","fullTitle":"Array #pop() should remove and return the last value","duration":1}], "failures":[], "passes":[{"title":"should return -1 when the value is not present","fullTitle":"Array #indexOf() should return -1 when the value is not present","duration":1}, {"title":"should return the correct index when the value is present","fullTitle":"Array #indexOf() should return the correct index when the value is present","duration":0}, {"title":"should remove and return the last value","fullTitle":"Array #pop() should remove and return the last value","duration":1}]}
λ mocha (master):
```

json reporter

9.9 JSON STREAM

JSON stream测试报告输出根据“事件”断行了的JSON，以“start”事件开始，紧跟着测试通过或失败，最后是“end”事件。

```
λ mocha (master): make test
["start",{"total":3}]
["pass",{"title":"should return -1 when the value is not present","fullTitle":"Array #indexOf() should return -1 when the value is not present","duration":0}]
["pass",{"title":"should return the correct index when the value is present","fullTitle":"Array #indexOf() should return the correct index when the value is present","duration":0}]
["pass",{"title":"should remove and return the last value","fullTitle":"Array #pop() should remove and return the last value","duration":1}]
["end",{"suites":4,"tests":3,"passes":3,"failures":0,"start":"2011-11-19T20:27:27.538Z","end":"2011-11-19T20:27:27.542Z","duration":4}]
λ mocha (master):
```

json stream reporter

9.10 MIN

min测试报告仅显示结果摘要，当然也输出失败时的错误信息。当与 `--watch` 一起使用时很棒，它会清空你的命令行让测试摘要始终显示在最上方。

✓ 62 tests complete (63ms)

min reporter

9.11 DOC

doc测试报告输出一个层级化的HTML来表示你的测试结果。使用header, footer和一些样式来包裹测试结果, 然后你就有了一份惊艳的测试报告文档!

```
λ mocha (master): make test
<section class="suite">
  <h1>Array</h1>
  <dl>
    <section class="suite">
      <h1>#indexOf()</h1>
      <dl>
        <dt>should return -1 when the value is not present</dt>
        <dd><pre><code>
[1,2,3].indexOf(5).should.equal(-1);
[1,2,3].indexOf(0).should.equal(-1);</code></pre></dd>
        <dt>should return the correct index when the value is present</dt>
        <dd><pre><code>
[1,2,3].indexOf(1).should.equal(0);
[1,2,3].indexOf(2).should.equal(1);
[1,2,3].indexOf(3).should.equal(2);</code></pre></dd>
      </dl>
    </section>
    <section class="suite">
      <h1>#pop()</h1>
      <dl>
        <dt>should remove and return the last value</dt>
        <dd><pre><code>
var arr = [1,2,3];
```

doc reporter

例如, 假定你有下面的JavaScript:

```
1 describe('Array', function() {
2   describe('#indexOf()', function() {
3     it('should return -1 when the value is not present', function() {
4       [1,2,3].indexOf(5).should.equal(-1);
5       [1,2,3].indexOf(0).should.equal(-1);
6     });
7   });
8 });
```

在命令行输入 `mocha --reporter doc array` 会输出:

```
1 <section class="suite">
2   <h1>Array</h1>
3   <dl>
4     <section class="suite">
5       <h1>#indexOf()</h1>
6       <dl>
7         <dt>should return -1 when the value is not present</dt>
8         <dd><pre><code>[1,2,3].indexOf(5).should.equal(-1);
9 [1,2,3].indexOf(0).should.equal(-1);</code></pre></dd>
10        </dl>
11      </section>
12    </dl>
13  </section>
```

```
1 test-docs:
2   $(MAKE) test REPORTER=doc \
3     | cat docs/head.html - docs/tail.html \
4     > docs/test.html
```

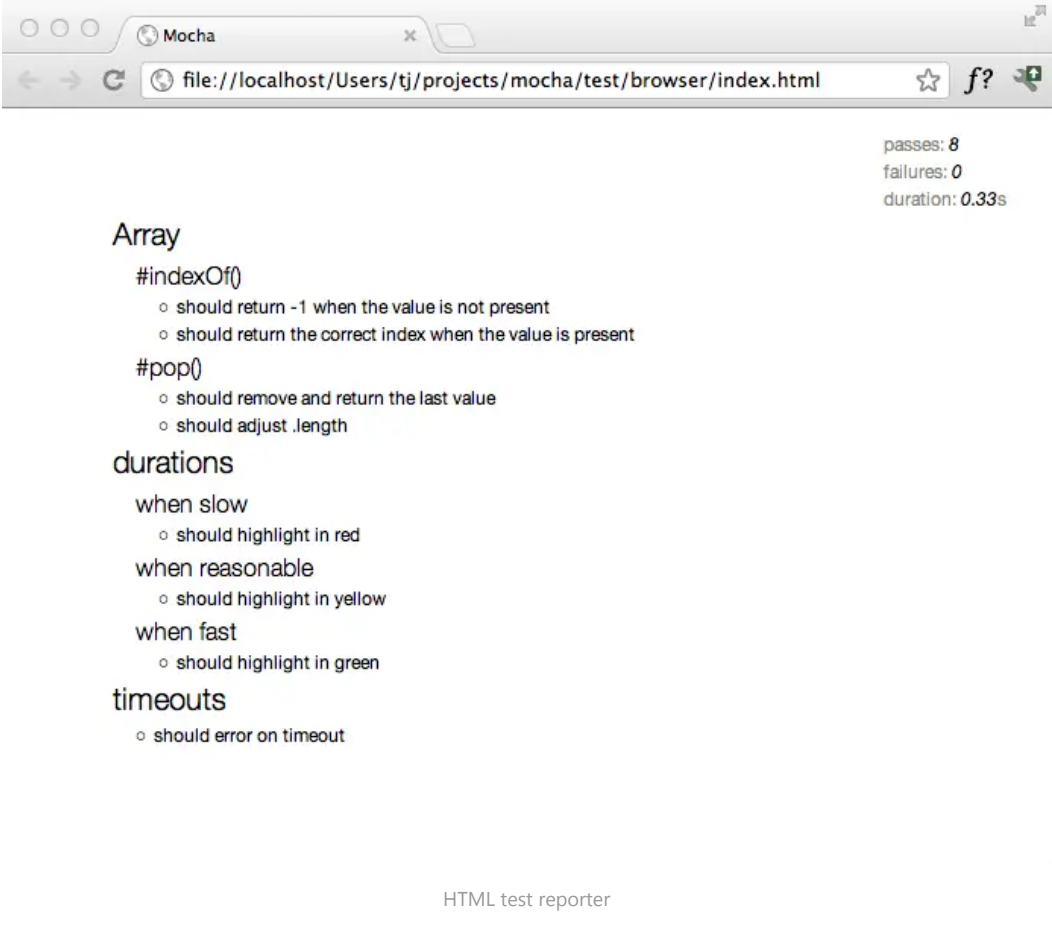
View the entire [Makefile](#) for reference.

9.12 MARKDOWN

markdown测试报告为你的测试讨价能生成一个markdown TOC。当你希望将你的测试结果放在Github的wiki或仓库中时，这会很好用。这是一个例子的链接[测试输出](#)

9.13 HTML

HTML测试报告是当前Mocha唯一支持的浏览器测试报告，长得像这样：



9.14 未记载的测试报告

XUnit测试报告也是可以用的。默认地，它输出到console。想直接写入文件，使用 `--reporter-options output=filename.xml`。

9.15 第三方的测试报告

Mocha允许自定义第三方的测试报告生成器。浏览[wiki](#)获取更多信息。一个例子是[TeamCity](#)

Mocha可以运行在浏览器中。Mocha的每个释出版本都会有 `./mocha.js` 和 `./mocha.css` 来在浏览器中使用。

10.1 浏览器专用方法

下面的方法仅在浏览器环境中有效：

`mocha.allowUncaught()`：如果调用，未捕获的错误不会被error handler处理。

一个典型的设置看起来可能像下面这样，在载入测试脚本，在 `onload` 中用 `mocha.run()` 运行它们之前，我们调用 `mocha.setup('bdd')` 来使用BDD风格的接口。

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Mocha Tests</title>
5   <link href="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.css" rel="stylesheet" />
6 </head>
7 <body>
8   <div id="mocha"></div>
9
10  <script src="https://cdn.rawgit.com/jquery/jquery/2.1.4/dist/jquery.min.js"></script>
11  <script src="https://cdn.rawgit.com/Automattic/expect.js/0.3.1/index.js"></script>
12  <script src="https://cdn.rawgit.com/mochajs/mocha/2.2.5/mocha.js"></script>
13
14  <script>mocha.setup('bdd')</script>
15  <script src="test.array.js"></script>
16  <script src="test.object.js"></script>
17  <script src="test.xhr.js"></script>
18  <script>
19    mocha.checkLeaks();
20    mocha.globals(['jQuery']);
21    mocha.run();
22  </script>
23 </body>
24 </html>
```

10.2 grep

浏览器中也可以使用 `--grep` 功能。在你的URL上添加一个请求参数：`?grep=api`。

10.3 浏览器配置

Mocha的选项可以通过 `mocha.setup()` 来配置。比如：

```
1 // Use "tdd" interface. This is a shortcut to setting the interface;
2 // any other options must be passed via an object.
3 mocha.setup('tdd');
4
5 // This is equivalent to the above.
6 mocha.setup({
7   ui: 'tdd'
8 });
9
10 // Use "tdd" interface, ignore leaks, and force all tests to be asynchronous
11 mocha.setup({
12   ui: 'tdd',
13   ignoreLeaks: true,
14   asyncOnly: true
15 });
```

code。

mocha.opts

回到服务器，Mocha会试图加载 `./test/mocha.opts` 作为Mocha的配置文件。文件是由命令行参数按行拼接起来的。命令行参数也是有优先级的，例如，假定你有下面的 `mocha.opt` 文件：

```
1 | --require should
2 | --reporter dot
3 | --ui bdd
```

它会将默认测试报告设置为dot，加载 `should` 断言库，并使用BDD风格的接口。然后你可能会继续带参数运行Mocha，这里是开启Growl支持，并将测试报告更换为list：

```
1 | $ mocha --reporter list --growl
```

11. test/ 文件夹

Mocha默认会全局寻找 `./test/*.js` 和 `./test/*.coffee`，所以你可能需要将你的测试文件放到 `./test` 文件夹中

12. 编辑器插件

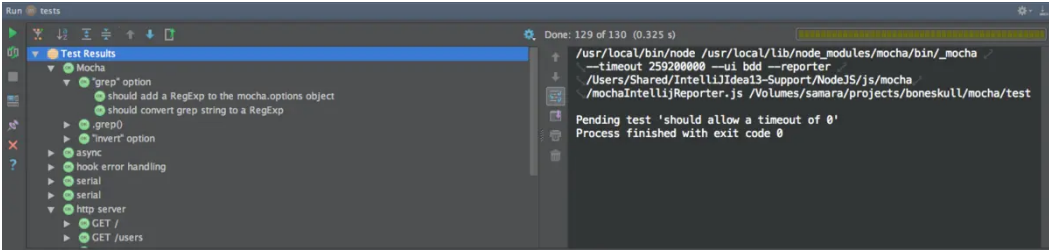
下面的编辑器插件package可用：

12.1 TextMate

Mocha的TextMate包包含了能够加速测试编写的代码片段。克隆Mocha repo并运行 `make tm` 来安装这个包

12.2 JetBrains

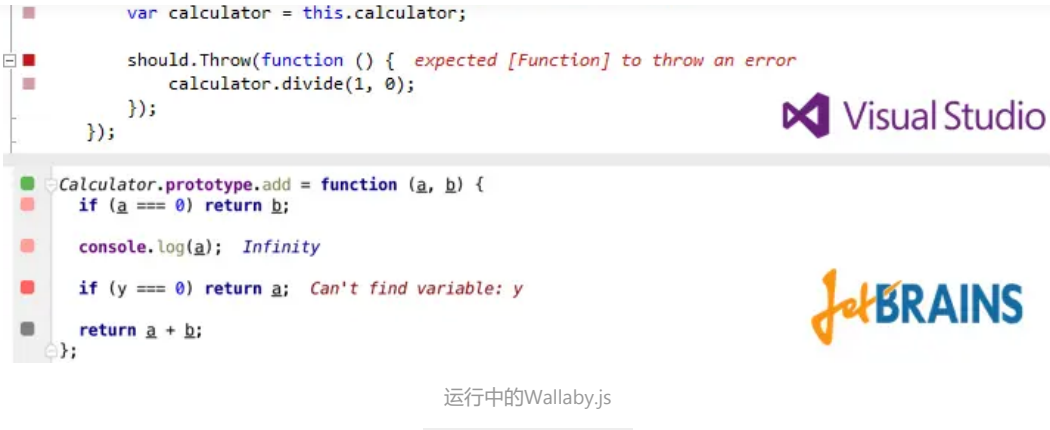
JetBrains为它们的IDE套件（IntelliJ IDEA，WebStorm等）提供了一个NodeJS插件，包含了一个Mocha test runner，和一些周边。



运行中的JetBrains Mocha Runner Plugin

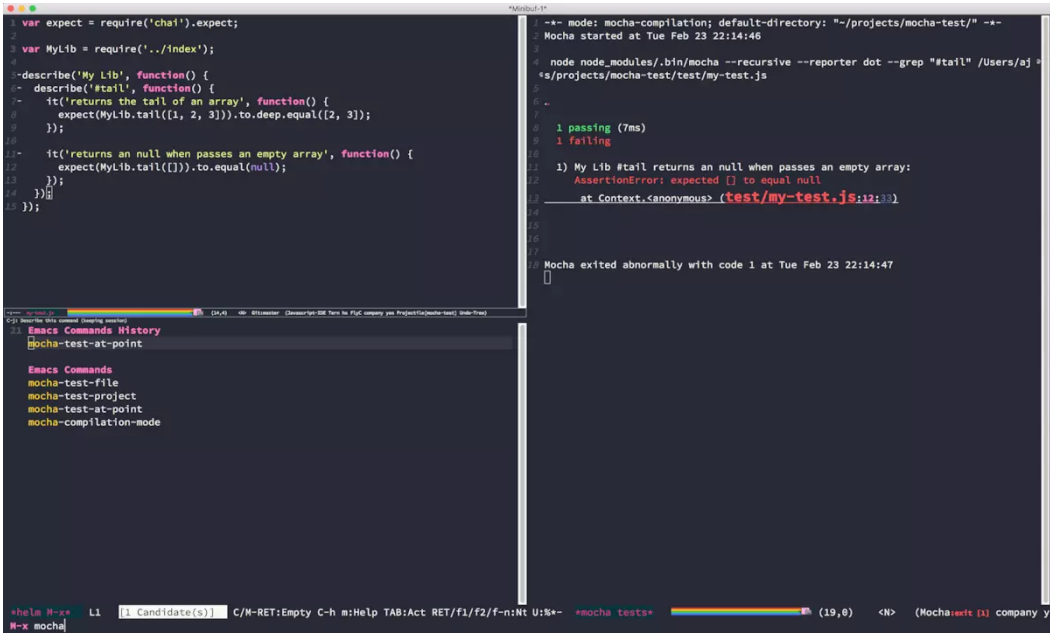
插件名为NodeJS，并且可以通过Preference > Plugins来安装...如果你的许可允许的话。

12.3 Wallaby.js



12.4 Emacs

Emacs支持通过第三方插件mocha.el来运行Mocha测试。插件可以在MELPA上找到，也可通过 `M-x package-install mocha` 来安装。



运行中的Emacs Mocha Runner

13. 案例

真实案例代码：

- Express
- Connect
- SuperAgent
- WebSocket.io
- Mocha

14. 测试Mocha

运行Mocha本身的测试，你可能需要GUN Make或者其它兼容的环境；Cygwin应该就可以。



使用不同的测试报告：

```
1 | $ REPORTER=nyan npm test
```

15. 更多信息

除了在[Gitter](#)上与我们交谈，也可以去GitHub上的[Mocha Wiki](#)获取诸如 using spies、mocking和 shared behaviours等更多信息。加入[Google Group](#)进行讨论。查看[example/tests.html](#)获取 Mocha运行实例。查看[source](#)获取 JavaScript API。

- 译者注：这里的意思是，只要这个断言库在遇到测试不通过时会抛出异常（throws an Error），它就可以使用在Mocha中。这意味着即使你不使用任何断言库，完全自己实现测试代码，只要在遇到测试部不通过时你抛出异常，也是可以的，不过通常没有人愿意这么费力不讨好 [🔗](#)
- 译者注：在it()函数的第二个参数，也就是it的回调函数中，添加一个参数，这个参数也是一个回调函数，命名随意，通常都会命名为 `done`，然后在异步代码运行完毕后调用它即可 [🔗](#)
- 译者注：这是为了避开无穷的回调漩涡 [🔗](#)
- 译者注：这里指的是省略done()这个回调函数 [🔗](#)
- 译者注：还记得前文提到的done()回调函数么，在这些钩子中处理异步代码与在测试用例中处理异步代码是一样的 [🔗](#)
- 译者注：这是指在命令行运行 mocha 命令时带上 --delay参数，下文的setTimeout仅做演示用，并非真实的异步操作 [🔗](#)
- 译者注：这里的意思是，可以使用挂起的测试用例来占个位置，先写上测试用例，但可能由于某些原因，比如被测代码还未实现，这个测试用例的内容将在稍后编写，这样的测试用例既不会pass也不会fail，而是处于pending状态 [🔗](#)
- 译者注：这里指的是嵌套的only()会存在优先级 [🔗](#)
- 译者注：这里所谓的“什么也没做”指的是当测试环境不正确时，测试代码什么也没做，只是简单的跳过了 [🔗](#)
- 译者注：这里的“什么也不做”与前文的“什么也不做”不是一个意思，这里的意思是既不写断言也不跳过，只是简单地空在那里（简单地空在那里这个测试的状态也会是passing） [🔗](#)
- 译者注：这种方式指的是，判断测试环境是否准本好并使用this.skip()来跳过测试 [🔗](#)
- 译者注：“覆盖这个值”指的是在嵌套的测试套件或者测试用例中再次通过this.timeout()来进行超时限制从而覆盖父级套件的设置 [🔗](#)
- 译者注：指的是接口风格 [🔗](#)
- 译者注：这里指的是编程风格扁平化，即没有多层嵌套 [🔗](#)

"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



Awey 好吧，简书还没凉，以后文章就同步更新了。掘金地址：<https://juejin.im/us...>
总资产151 (约13.45元) 共写了5.6W字 获得469个赞 共294个粉丝

关注



文档管理软件



文档管理系统



前端是干什么的



文档共享



忧郁早期症状



写下你的评论...

全部评论 4

只看作者

按时间倒序 按时间正序



星光繁绕

5楼 12.16 14:32

非常感谢！

👍 赞 🗨 回复



jimi董星辰

4楼 2018.02.02 15:14

感谢楼主分享！

👍 赞 🗨 回复



EzioShiki

3楼 2017.11.23 15:30

支持，很有用

👍 赞 🗨 回复



huangsw

2楼 2017.08.10 13:55

赞

👍 赞 🗨 回复

被以下专题收入，发现更多相似内容



技术干货



前端



ericsson



javascript



前端之旅



无忧万阅

写下你的评论...

评论4

赞30



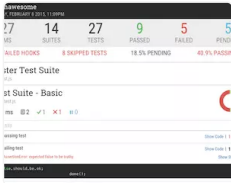
推荐阅读

更多精彩内容>

测试框架 Mocha 实例教程

原文链接：<http://www.ruanyifeng.com/blog/2015/12/a-mocha-tutor...>

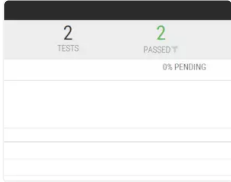
 butterflyq 阅读 773 评论 1 赞 3



mocha 的基本介绍&&expect风格断言库的基本语法

mocha 介绍 mocha 是一个功能丰富的javascript测试框架，可以运行在nodejs和浏览器环境，能...

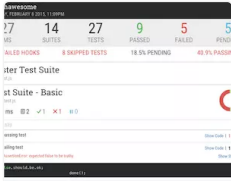
 艾伦先生 阅读 5,652 评论 2 赞 12



测试框架 Mocha 实例教程

原作者：阮一峰原文地址：<http://www.ruanyifeng.com/blog/2015/12/a-moc...>

 Simon王小白 阅读 303 评论 0 赞 0



自动化测试之前端js单元测试框架jest

大多数开发者都知道需要写单元测试，但是不知道每个单元测试应用的主要内容以及如何做单元测试，在介绍jest测试框架前...

 糖小工 阅读 3,100 评论 0 赞 10

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...

 卡卡罗2017 阅读 72,193 评论 14 赞 116

