2017.08.07-09.14

单元测试框架 Mocha

学信网 - 设计部 黄卉
huangh@chsi.com.cn

simple, flexible, fun

单元测试框架Mocha

- 1、Mocha单元测试框架介绍
- 2、Mocha单元测试用法
- 3、Mocha命令行用法

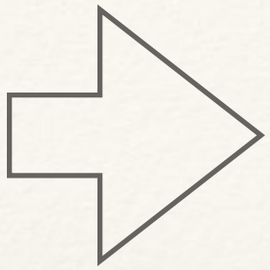
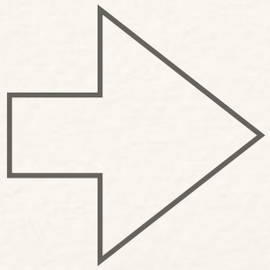
1、Mocha单元测试框架介绍

- 1.1 单元测试的基本概念
- 1.2 单元测试的重要性
- 1.3 为什么选择Mocha
- 1.4 Mocha特点

1.1 单元测试的基本概念

- 单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。（百度百科）
- 单元：相对独立功能模块，类、模块、方法。
- 又称为模块测试，是针对程序模块(软件设计的最小单位)来进行正确性检验的测试工作。
- 用来检验程式的内部逻辑，也称为个体测试、结构测试或逻辑驱动测试。

- 如果要测试的函数里有Click事件操作怎么办?
  - Click事件属于DOM（Document Object Model 文档对象模型）操作
  - DOM并不属于Javascript语言本身
  - ECMAScript单元测试，不是Javascript单元测试
  - 集成测试，测试框架selenium+python/java（咱测试部门的自动化测试）

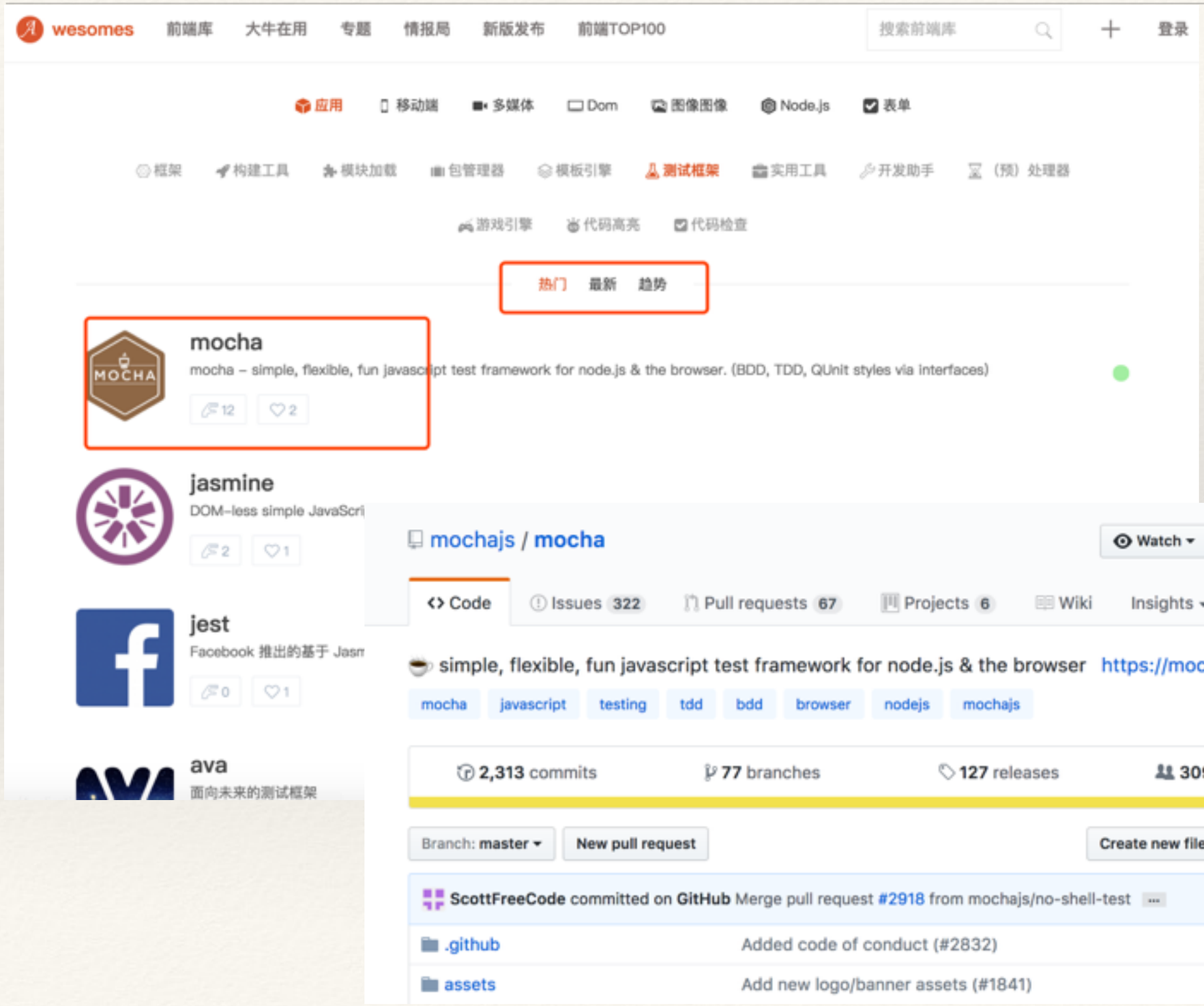
单元测试  集成测试  功能测试

- 单元测试，单个组件正常工作，开发阶段；
- 集成测试，不同组件互相合作，中间阶段；
- 功能测试，主要测试界面，开发完成。

1.2 单元测试的重要性

- 黑盒测试
  - bug无法捕捉、重现
  - 费力，工作量大
  - 覆盖面低
  - 反复出现bug
- 单元测试
  - 并不是所有的 js 都需要单元测试。中大型项目
  - 并不是所有的 js 都能够单元测试。良好的模块化和解耦

1.3 为什么选择Mocha



1.4 Mocha特点

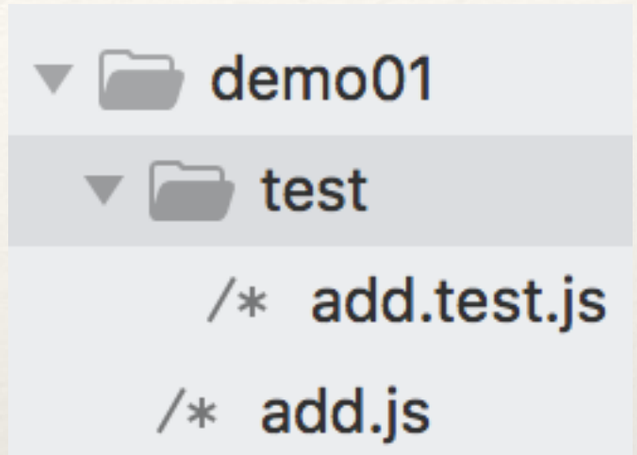
- 多环境使用：浏览器、Node环境；
- 既可以测试简单的JavaScript函数，又可以测试异步代码；
- 可以自动运行所有测试，也可以只运行特定的测试；
- 使用任意的断言库；
- 可以通过周期函数来编写初始化代码；
- 可通过配置文件来实现默认命令行的执行；
- 方法够用，简单，易入手。

2、Mocha单元测试用法

- 2.1 单元测试大致流程
- 2.2 单元测试基本用法
- 2.3 单元测试生命周期
- 2.4 测试管道管理
- 2.5 动态生成测试用例
- 2.6 chai断言库
- 2.7 expect断言风格

2.1 单元测试大致流程

- 全局安装：npm install --global mocha
- 创建项目：
  - 新建文件夹demo01，需要测试脚本 add.js
  - 新建文件夹test，放测试脚本，新建测试脚本add.test.js
  - 项目依赖：
    - nam install --save-dev mocha //测试框架
    - npm install --save-save chai //断言库
- 终端运行，进入到demo01目录下，执行：mocha



- 运行结果
  - 只有两种：测试通过或者测试不通过。
  - 显示基本信息：测试条数、测试结果、测试时间、报错信息等。

A、测试通过

```
➜  demo01 git:(master) ✗ mocha


test/add.test.js-- 加法函数的测试
  ✓ 1 加 1 等于 2
  ✓ 1 加 2 不等于 2


2 passing (9ms)
```

- 只有所有的测试用例通过了，该单元测试成功。
- 只要有一条测试用例不通过，则该单元测试失败。

B、测试不通过

```
➜  demo01 git:(master) ✗ mocha


test/add.test.js-- 加法函数的测试
  ✓ 1 加 1 等于 2
  ✓ 1 加 2 不等于 2
  1) 3 加 2 等于 2


  2 passing (12ms)
  1 failing

  1) test/add.test.js-- 加法函数的测试 3 加 2 等于 2:

      AssertionError: expected 5 to equal 2
      + expected - actual

      -5
      +2

      at Context.<anonymous> (test/add.test.js:12:26)
      at callFn (/usr/local/lib/node_modules/.mocha_npm
/lib/runnable.js:348:21)
```

2.2 单元测试基本用法

A、需要测试的脚本：add.js

```
add.test.js   add.js
1  function add(x, y) {
2    return x + y;
3  }
4
5  module.exports = add;
6
```

NodeJS知识点：

module.exports 导出模块接口

require 导入接口

chai断言库：

expect、assert 等写法

B、单元测试脚本：add.test.js

```
单元测试.md   add.test.js   package.json
1  var add = require('../src/add.js');
2  var expect = require('chai').expect;
3                  测试套件          测试函数
4  describe('加法函数的测试', function() {
5    it('1 加 1 应该等于 2', function() {   测试用例
6      expect(add(1, 1)).to.be.equal(2);
7    });        断言
8
9    it('任何数加0应该等于自身', function() {
10     expect(add(1, 0)).to.be.equal(1);
11   });
12 });
13
14
15
```

测试脚本可以独立执行。
一个测试脚本里面应该包括一个或多个describe块，每个describe块应该包括一个或多个it块。

- 单元测试语法
  - "测试套件"（test suite）： describe (moduleName, testDetails)。可以嵌套使用，明白、易懂即可。
  - "测试用例"（test case）： it (info, function)。具体的测试语句，可多个。
    - info，写期望的正确输出的简要一句话文字说明。
    - function，具体测试函数，一个测试用例内部，包含一个或多个断言（assert）。
  - assert.equal (exp1, exp2)： 断言判断exp1结果是否等于exp2，这里采取的等于判断是== 而并非 === ，即 assert.equal(1, '1') 认为是True。

单元测试.md | add.test.js | package.json

```
var add = require('../src/add.js');
var expect = require('chai').expect;
        测试套件                    测试函数
describe('加法函数的测试', function() {
  it('1 加 1 应该等于 2', function() {                测试用例
    expect(add(1, 1)).to.be.equal(2);
  });   断言

  it('任何数加0应该等于自身', function() {
    expect(add(1, 0)).to.be.equal(1);
  });
});
```

```
var assert = require('chai').assert;
describe('Array-assert方法', function() {
  describe('测试indexOf()', function() {
    it('不包含时，返回-1', function() {
      assert.equal(-1, [1,2,3].indexOf(1));
      assert.equal(-1, [1,2,3].indexOf(0));
    });
  });
});
```

2.3 单元测试生命周期

- 每个测试块（describe）有4个周期函数：before、beforeEach、afterEach、after

周期函数	存在周期	主要功能
before()	在本区块的所有测试用例之前执行	用于同一的桩数据导入等功能
beforeEach()	在本区块的每个测试用例之前执行	用于清理测试环境，删除或回滚相关数据
afterEach()	在本区块的每个测试用例之后执行	可以用于准备测试用例所需的前置条件
after()	在本区块的所有测试用例之后执行	可以用于准备测试用例所需的后置条件

demo2

2.4 测试管道管理

add.test.js × multiply.test.js ×

```
1  var add = require('../src/add.js');
2  var expect = require('chai').expect;
3
4  describe('加法函数的测试', function() {
5    it.only('1 加 1 应该等于 2', function() {
6      expect(add(1, 1)).to.be.equal(2);
7    });
8
9    it('任何数加0应该等于自身', function() {
10     expect(add(1, 0)).to.be.equal(1);
11   });
12 });
13
```

```
→  demo03 git:(master) ✗ mocha


  add-加法函数的测试
    ✓ 1 加 1 应该等于 2


  1 passing (9ms)
```

- only方法：表示只运行某个测试套件或测试用例。 demo3

add.test.js × multiply.test.js ×

```
1  var multiply = require('../../src/multiply');
2  var expect = require('chai').expect;
3
4  describe('乘法函数的测试', function() {
5    it('1 乘 1 应该等于 1', function() {
6      expect(multiply(1, 1)).to.be.equal(1);
7    });
8    it.skip('0 乘 任何数 应该等于 0', function() {
9      expect(multiply(0, 0)).to.be.equal(0);
10     expect(multiply(0, 1)).to.be.equal(0);
11   });
12 })
```

```
[→  demo03 git:(master) ✗ mocha test/dir/multiply.test.js


  乘法函数的测试
    ✓ 1 乘 1 应该等于 1
    - 0 乘 任何数 应该等于 0


  1 passing (9ms)
  1 pending
```

- skip方法：表示跳过指定的测试套件或测试用例。demo3

2.5 动态生成测试用例

add.test.js

```
//动态生成用例
var assert = require('chai').assert;

function add() {
  return Array.prototype.slice.call(arguments).reduce(function(prev, curr) {
    return prev + curr;
  }, 0);
}

describe('a.test.js--数组和add()', function() {
  var tests = [
    {args: [1, 2],       expected: 3},
    {args: [1, 2, 3],    expected: 6},
    {args: [1, 2, 3, 4], expected: 10},
    {args: [1, 2, 3, 4, 10], expected: 20}
  ];

  tests.forEach(function(test) {
    it('集合中添加参数的个数:  ' + test.args.length + ' 个', function() {
      var res = add.apply(null, test.args);
      assert.equal(res, test.expected);
    });
  });
});
```

```
→  demo04 git:(master) ✗ mocha


add.test.js--数组和add()
  ✓ 集合中添加参数的个数:  2 个
  ✓ 集合中添加参数的个数:  3 个
  ✓ 集合中添加参数的个数:  4 个
  ✓ 集合中添加参数的个数:  5 个


4 passing (11ms)
```

- forEach：有多个测试数据需要显示的时候

demo04

2.6 断言库

- 断言指的是对代码行为的预期，会返回一个布尔值，表示代码行为是否符合预期。

```
var assert = require('chai').assert;
describe('Array-assert方法', function() {
  describe('测试indexOf()', function() {
    it('不包含时，返回-1', function() {
      assert.equal(-1, [1,2,3].indexOf(1));
      assert.equal(-1, [1,2,3].indexOf(0));
    });
  });
});
```

实际值（-1）和期望值（[1,2,3].indexOf(5)）是一样的，断言为true。

第一个不成功，第二个成功。最后为测试失败。

- mocha 允许开发者使用任意的断言库。
- 常用断言库：should.js、expect.js、**chai.js**、better-assert、assert（nodejs 原生模块）。

- chai.js断言库：接口丰富，文档齐全，可以对各种接口进行断言。
- 三种风格的断言：
  - TDD测试驱动开发风格 assert ，不支持链式调用。
  - BDD行为驱动开发风格的expect和should（不支持IE），链式语言。

```
assert.equal(variable, "value");
expect(variable).to.equal("value");
variable.should.equal("value");
```

demo05

2.7 expect断言风格

- ok：检查是否为真
- true：检查对象是否为真
- to.be、to：作为连接两个方法的链式方法
- not：链接一个否定的断言，如 expect(false).not.to.be(true)
- a/an：检查类型（也适用于数组类型）
- include/contain：检查数组或字符串是否包含某个元素
- below/above：检查是否大于或者小于某个限定值

通常写同一个断言会有几个方法

比如：expect(response).to.be(true) 和 expect(response).equal(true)

3、Mocha命令行用法

- 3.1 基本用法
- 3.2 异步请求测试
- 3.3 ajax异步请求测试
- 3.4 定制生成格式
- 3.5 添加默认配置
- 3.6 浏览器显示测试结果
- 3.7 单元测试实例
- 3.8 在gulp中运行Mocha测试

3.1 基本用法

- mocha命令的基本格式是：mocha [debug] [options] [files]
- 基本用法
  - mocha：默认运行test子目录里面的测试脚本，不包括子文件
  - mocha add.test.js：当前目录下面的该测试脚本。
  - mocha file1 file2 file3： mocha命令后面紧跟测试脚本的路径和文件名，可以指定多个测试用例。
- 通配符
  - mocha spec/{my,awesome}.js
  - mocha test/unit/*.js
- 命令行参数 mocha --help：查看所有命令
  - mocha --recursive：执行test子目录下面的所有的测试用例。
  - mocha --watch：-w 监控执行
  - mocha --timeout 5000 timeout.test.js：-t 异步测试中需要，默认时间为2000。

demo06

3.2 异步请求测试

```
var expect = require('chai').expect;

describe('timeout.test.js - 超时测试', function() {
    it('测试应该 1000 毫秒后结束', function(done) {
        var x = true;
        expect(x).to.be.ok;
        var f = function() {
            x = false;
            expect(x).to.be.not.ok;
            done();
        };
        setTimeout(f, 1000);
    });

    it('测试应该 2000 毫秒后结束', function(done) {
        var x = true;
        expect(x).to.be.ok;
        var f = function() {
            x = false;
            expect(x).to.be.not.ok;
            done();
        };
        setTimeout(f, 2000);
    });
});
```

- done：一个it里面只有一个done。done标识回调的最深处，也是嵌套回调函数的末端。必须有，否则报错。
- 超时解决：mocha --timeout 3000 timeout.test.js

```
[→  demo07 git:(master) ✗ mocha timeout.test.js ]


  timeout.test.js - 超时测试
    ✓ 测试应该 1000 毫秒后结束 (1007ms)
    1) 测试应该 2000 毫秒后结束


  1 passing (3s)
  1 failing

  1) timeout.test.js - 超时测试 测试应该 2000
毫秒后结束:
     Error: Timeout of 2000ms exceeded. For asy
nc tests and hooks, ensure "done()" is called;
if returning a Promise, ensure it resolves.
      at null.<anonymous> (/usr/local/lib/node_
modules/.mocha_npminstall/mocha/3.4.2/mocha/lib
/runnable.js:232:19)
```

```
→  demo07 git:(master) ✗ mocha --timeout 5000
timeout.test.js


  timeout.test.js - 超时测试
    ✓ 测试应该 1000 毫秒后结束 (1004ms)
    ✓ 测试应该 2000 毫秒后结束 (2003ms)


  2 passing (3s)
```

demo07

3.3 ajax异步请求测试

```
var request = require('superagent') ;
var expect = require('chai').expect;
describe('promise.test.js - 异步测试', function() {

    it('测试head方法', function(done){
      request
        .head('http://kl.chdi.com.cn/favicon.ico')
        .end(function(err,res){
            if (err || !res.ok) {
                console.log('返回错误，状态为: '+res.status);
            } else {
                console.log("返回成功: "+res.status);
            }
            done();
        })
});

it('测试get方法', function(done){
  request
    .get("http://kl.chdi.com.cn/search/allSearch.action?keywords='身份证重复'")
    .end(function(err,res){
        if (err || !res.ok) {
            console.log('返回错误，状态为: '+res.status);
        } else {
            console.log('获取内容为:  '+ res.body.flag);
            console.log('，一共多少条数据: '+res.body.o.knows.length);
            //console.log('获取内容为:  ' + JSON.stringify(res.body));
        }
        done();
    })
});
```

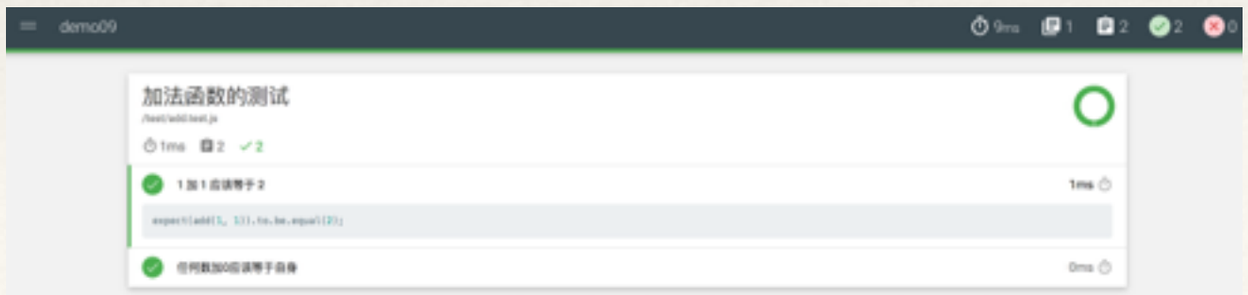
```
[→  demo07 git:(master) ✗ mocha promise.test.js


  promise.test.js - 异步测试
返回成功: 200
    ✓ 测试head方法
    ✓ 测试get方法 (728ms)
    - 测试post方法


  2 passing (761ms)
  1 pending
```

- del/head/get/post/put和别的http动作都可以使用。
- request、end、send、set，均可使用，详见：SuperAgent中文使用文档

demo07

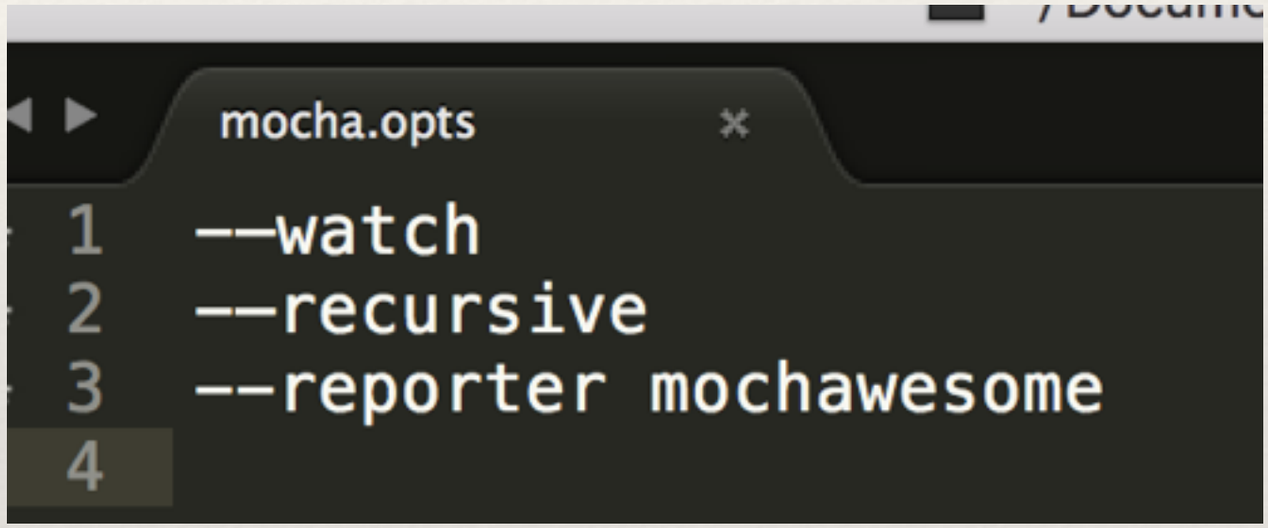
3.4 定制生成格式

- mocha --reporter spec：默认为spec格式，可设置其他格式。
- -- reporter 简写 -R
- 生成MD格式
  - mocha --recursive -R markdown > spec.md 。
    - 上面命令根据test目录的所有测试脚本，生成一个规格文件spec.md。-R markdown参数指定规格报告是markdown格式。
  - mocha --recursive -R doc > spec.html
  - 详见demo08
- 添加mochawesome格式，网页版查看
  - 安装项目依赖：npm install --save-dev mochawesome
  - 执行：mocha --reporter mochawesome
  - 详见 demo09



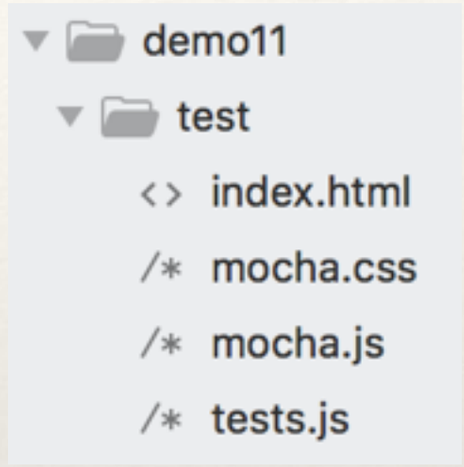
3.5 添加默认配置

- test测试目录下新建mocha.opts
- 例如：mocha --reporter mochawesome --recursive --watch
- 执行：mocha
- 详见demo10



3.6 浏览器显示测试结果

- 动态生成模板：mocha init test，自动生成一个test目录
- 其中tests.js是空的，需要自己写（建议将tests.js改成target.test.js）
- 在html中添加断言库、对应需要测试的target.js。
  - 添加断言库：<script src=“http://chaijs.com/chai.js"></script>
  - 添加需要测试的js：<script src="target.js"></script>
- 在浏览器里面打开index.html，就可以看到测试脚本的运行结果。



index.html — demo/test

```
<!DOCTYPE html>
<html>

<head>
    <title>Mocha</title>
    <meta http-equiv="Content-Type" content="text/
    html; charset=UTF-8">
    <meta name="viewport" content="
    width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mocha.css" />
</head>

<body>
    <div id="mocha"></div>
    <script src="mocha.js"></script>
    <script>
    mocha.setup('bdd');
    </script>
    <script src="tests.js"></script>
    <script>
    mocha.run();
    </script>
</body>

</html>
```

index.html — demo11

```
 1  <!DOCTYPE html>
 2  <html>
 3
 4  <head>
 5      <title>Mocha</title>
 6      <meta http-equiv="Content-Type" content="text/
        html; charset=UTF-8">
 7      <meta name="viewport" content="
        width=device-width, initial-scale=1.0">
 8      <link rel="stylesheet" href="mocha.css" />
 9  </head>
10
11  <body>
12      <div id="mocha"></div>
13      <script src="mocha.js"></script>
14      <script>
15      mocha.setup('bdd');
16      </script>
17      <script src="add.js"></script>
18      <script src="http://chaijs.com/chai.js"></script>
19      <script src="tests.js"></script>
20      <script>
21      mocha.run();
22      </script>
23  </body>
24
25  </html>
```

deme11

3.7 单元测试实例

```
describe('获取href链接getHref () 方法',function(){
    var tests = [
        {i:0,j:2,url:'https://my.chsi.com.cn/archive/gdjy/ky/show.action'},
        {i:2,j:0,url:'https://my.chsi.com.cn/archive/rzbg/index.action'},
        {i:3,j:1,url:'https://my.chsi.com.cn/archive/gjhz/utf/index.action'},
        {i:4,j:3,url:'http://gaokao.chsi.com.cn/zyk/myd/specAppraisalWelcome.action'}
    ];
    tests.forEach(function(t) {
        it('hrefJson中的序号为: '+ t.i +', ' + t.j + ', 对应链接: '+ t.url, function() {
            var res = getHref(t.i,t.j);
            assert.equal(res, t.url);
        });
    });
});
```

passes: 7 failures: 0 duration: 0.02s 100%

获取二级菜单选中的序列号getSecondIndex方法

- ✓ 当前链接为：gjhz/index.action一级栏目： gjhz，二级栏目：0
- ✓ 当前链接为：gjhz/utf/index.action一级栏目： gjhz，二级栏目：1
- ✓ 当前链接为：gjhz/apply.action一级栏目： gjhz，二级栏目：0

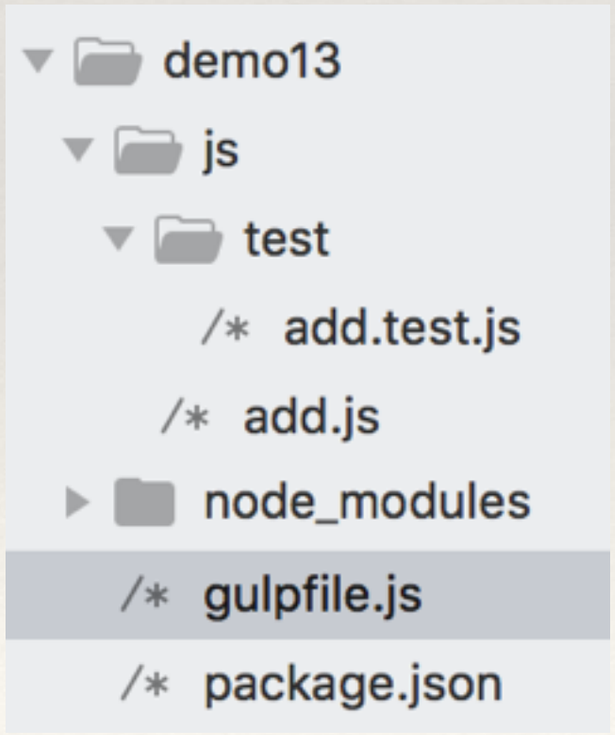
获取href链接getHref （） 方法

- ✓ hrefJson中的序号为：0，2，对应链接：https://my.chsi.com.cn/archive/gdjy/ky/show.action
- ✓ hrefJson中的序号为：2，0，对应链接：https://my.chsi.com.cn/archive/rzbg/index.action
- ✓ hrefJson中的序号为：3，1，对应链接：https://my.chsi.com.cn/archive/gjhz/utf/index.action
- ✓ hrefJson中的序号为：4，3，对应链接：http://gaokao.chsi.com.cn/zyk/myd/specAppraisalWelcome.action

demo12

3.8 在 gulp中运行Mocha测试

- 建立下图所示目录结构。
- 安装gulp-mocha插件：cnpm install - -save-dev gulp-mocha
- 编写“mocha”任务，如右图。
- 执行：gulp



gulpfile.js

```
mocha = require('gulp-mocha'),        ///单元测试

ArrAll = {
    lessDir: 'css/less',//需要解析的less文件目录
    cssDir : 'css/custom',          //需要解析的less文件目录
    jsSrc  : 'js/common',           //js目录
    jsTest : 'js/test/',            //单元测试目录
    imgSrc : 'images'    //原图片路径
};

//mocha
gulp.task('mocha', function() {
  return gulp.src(ArrAll.jsTest+['/*.test.js'], { read: false })
    .pipe(mocha({reporter: 'list'}))
});

//即时编译文件文件
gulp.task('watch',function(){
  gulp.watch(ArrAll.jsTest+['/*.test.js'],['mocha']);
});

//默认任务
gulp.task('default',function(){
    gulp.start(['mocha','watch']);
});
```

demo13

参考资料

- mocha官网：https://mochajs.org/
- chai断言库
- 测试框架 Mocha 实例教程
- 【Mocha.js 101】钩子函数
- 异步测试：SuperAgent中文使用文档

The end , Thanks!

huangh@chsi.com.cn