# Karma+mocha单元测试

### 一、基础

#### 什么是Karma

karma 是一个提升测试效率的工具，帮助开发者更好更快速地在多种环境下执行测试代码，拿到测试结果。在运行的时候，它会自动启动配置好的浏览器，同时也会启动一个 node 服务器，然后在启动好的浏览器中执行测试代码，并将测试代码执行结果传回给 node 服务器，然后 node 服务器在打印出收到的执行结果。

#### 什么是mocha

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，使异步测试变得简单有趣。mocha 串联运行测试，允许灵活和精确地报告结果，同时映射未捕获的异常用来纠正测试用例

#### 3、环境的搭建

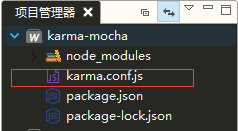
1. 安装node.js
2. 全局安装karma（npm install -g karma-cli）
3. 安装项目所需要的插件

使用 Karma Mocha Chai（启动器、测试框架、断言库）组合。

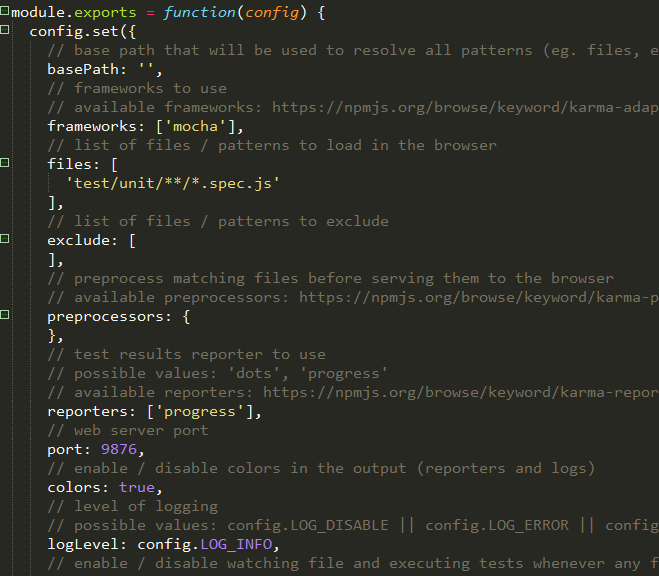
npm i -D chai mocha

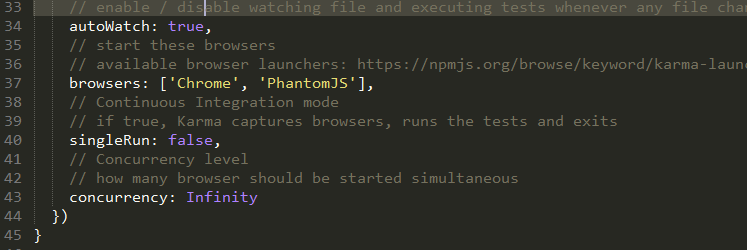
npm i -D karma-mocha karma-chai

生成了测试配置文件karma.config.js



配置文件内容karma.Config.js





d、es6支持

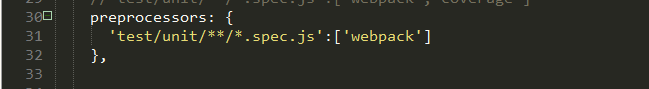
默认情况下，Karma+mocha是不支持es6的

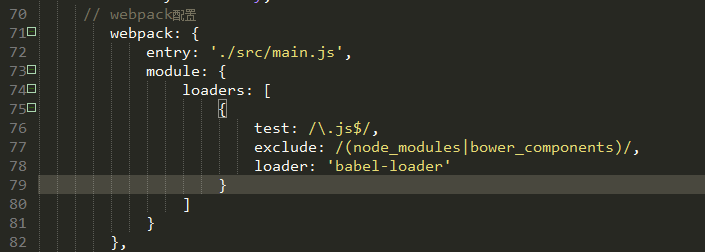
//添加es6支持

npm install karma-webpack -D// 安装 babel 相关的套件

npm install babel-core babel-loader babel-preset-es2015 --save-dev

安装依赖完成后，修改配置文件新增webpack支持es6



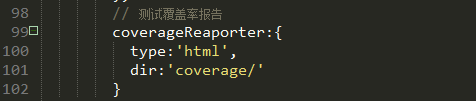
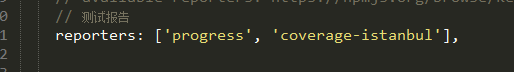


1. 新增测试覆盖报告

Npm install karma-coverage

Webpack 2.x版本以上

npm install istanbul-instrumenter-loader karma-coverage-istanbul-reporter --save-dev



#### 4、测试模块的基本结构

至少包括一个it模块

①、Describe(“对该测试的描述”,()=>{

It(“方法名或属性名”,()=>{

......

/\*断言 \*/

Expect().to.be

})

})

②、Describe(“对该测试的描述”,()=>{

It(“方法名或属性名”,()=>{

......

/\*断言 \*/

Expect().to.be

})

It(“方法名或属性名”,()=>{

......

/\*断言 \*/

Expect().to.be

})

})

#### describe 的钩子（生命周期）

describe('hooks', function() {

before(function() {

// 在本区块的所有测试用例之前执行

});

after(function() {

// 在本区块的所有测试用例之后执行

});

beforeEach(function() {

// 在本区块的每个测试用例之前执行

});

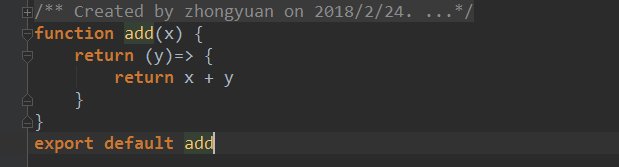
afterEach(function() {

// 在本区块的每个测试用例之后执行

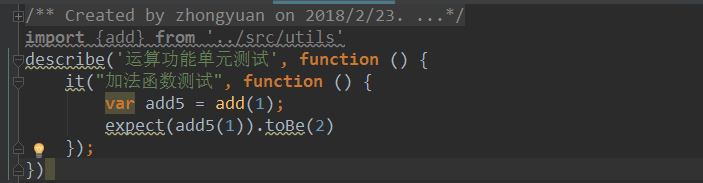
}); // test cases });

### 基本使用

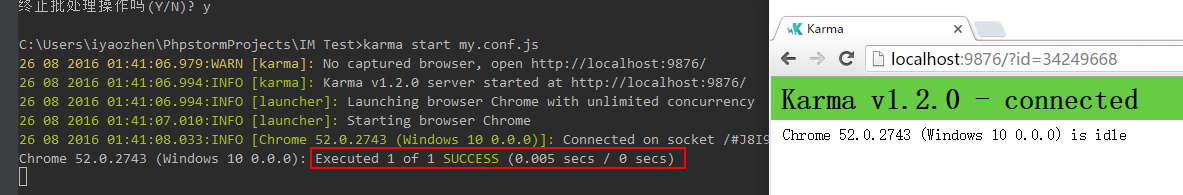
#### 测试用例编写

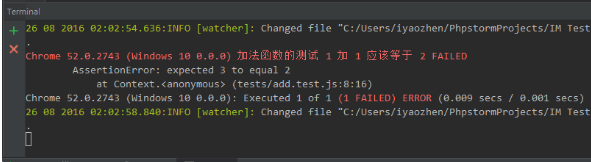


Test.spec.js



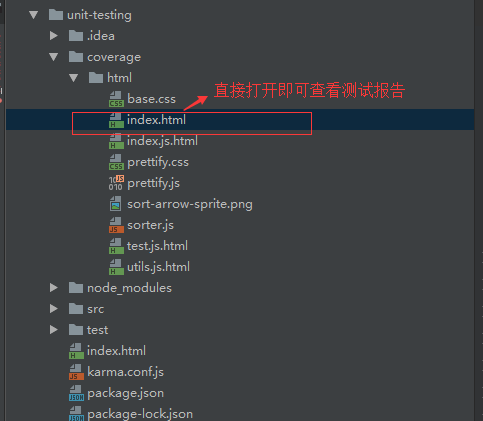
#### 查看测试结果

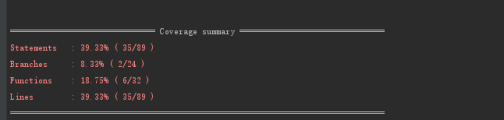
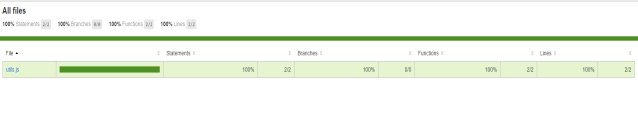




#### 查看测试报告覆盖率

直接打开/coverage下面的index.html，在浏览器中显示如下图





### 三、基于webpack搭建测试环境

#### 1、搭建测试环境

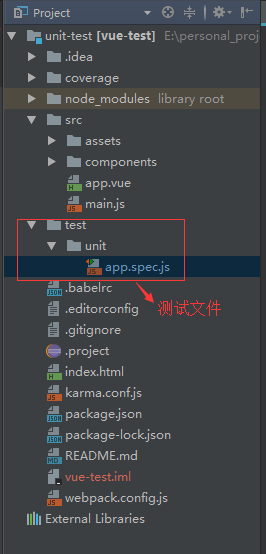
a、npm install karma-cli -g(全局安装)

b、安装karma+mocha相关依赖

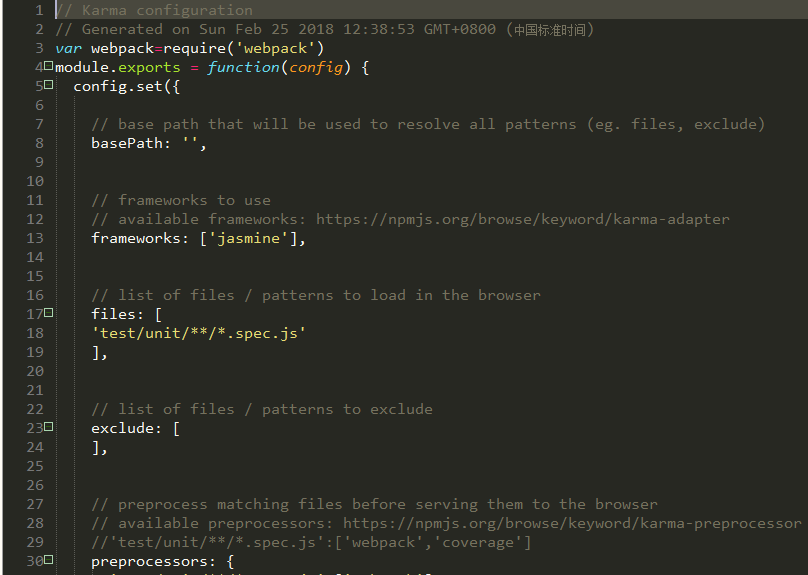
npm install karma karma-mocha karma-chai --save-dev

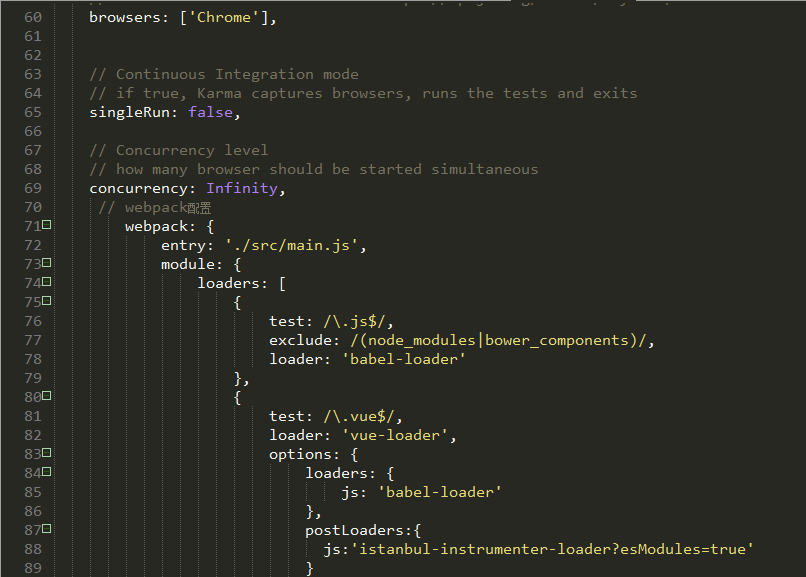
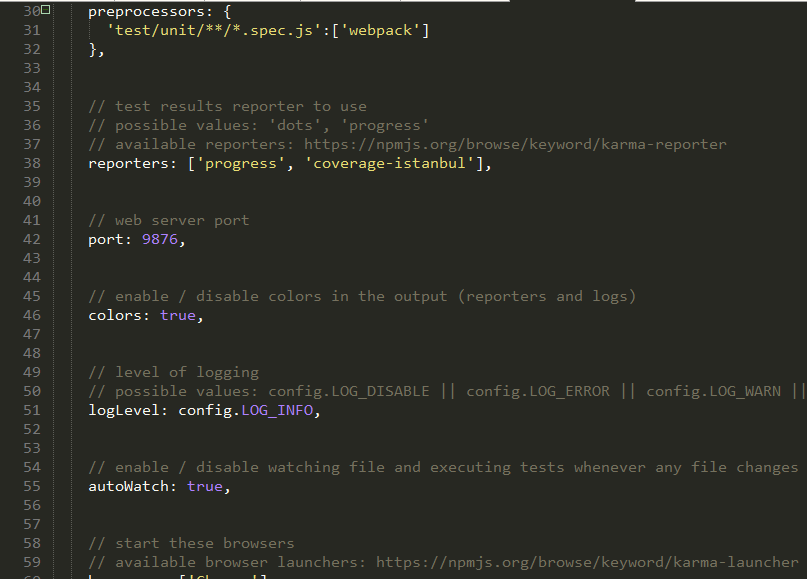
npm install mocha chai --save-dev

项目结构如下：



#### 2、修改配置文件





注：测试文件\*.spec.js命名必须和配置文件中的格式一样

### 四、在vue项目中运用

注：使用vue-cli搭建vue项目时（vue init webpack projectName ），是否使用单元测试，点击yes,选择karma+mocha即可，不需要再去修改配置文件，直接运行npm run test即可查看单元测试

#### 1、测试关注点

1. vue组件加载后，各数据模型是否符合预期
2. 定义的方法是否可用
3. Filter是否可用
4. 带有props的组件，数据能否正常传递
5. 异步更新DOM的情况

#### 2、vue测试

##### a、实例化Vue实例

对vue的单元测试，需要将组件实例化为一个Vue实例，有时候还需要挂在到DOM上

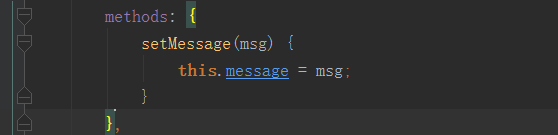
Import Hello from ‘Hello’

const Constructor=Vue.extend(Hello) //获取Hello组件实例

Const vm=new Constructor().$mount() // 将组件挂载到DOM上

##### b、测试方法是否正确(methods)

App.vue

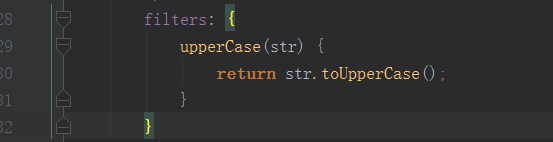


Test.sepc.js



##### c、测试过滤（filter）

App.vue

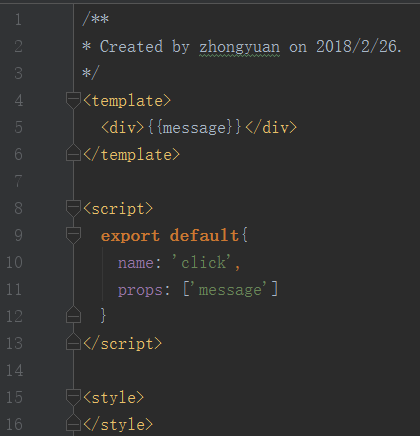


Test.spec.js

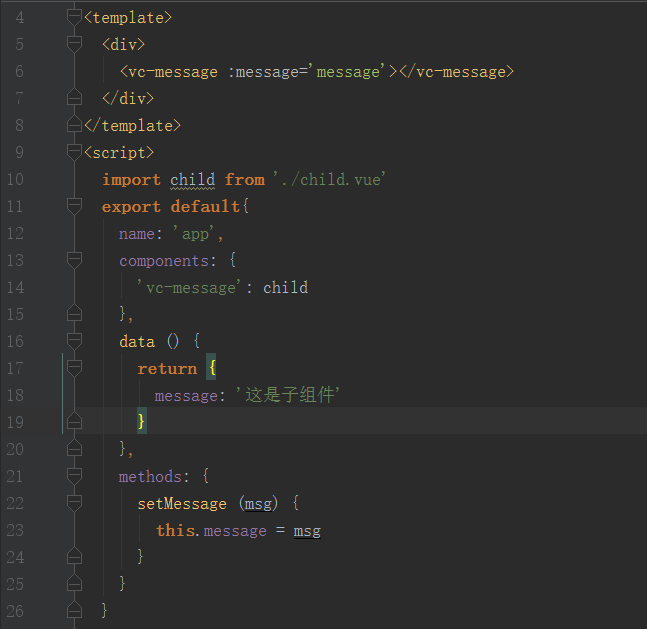


##### d、测试带有props的组件

Child.vue



App.vue



props.spec.js



##### 测试异步更新

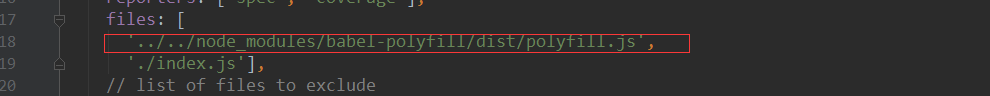
Test.spec.js



#### 3、配合vuex使用

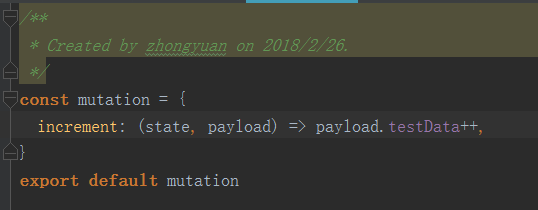
##### 引入vuex

在项目中使用vuex时，需要下载babel-pollyfill（npm install babel-pollyfill），然后在karma.conf.js修改配置文件



##### mutaions测试

Mutaions.js

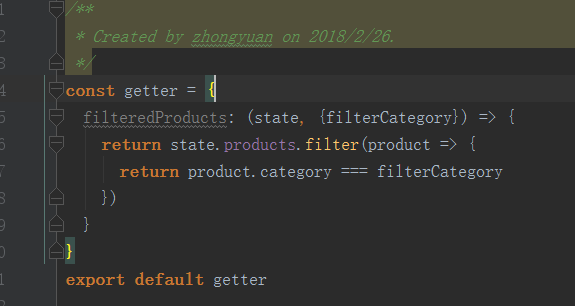


mutations.spec.js

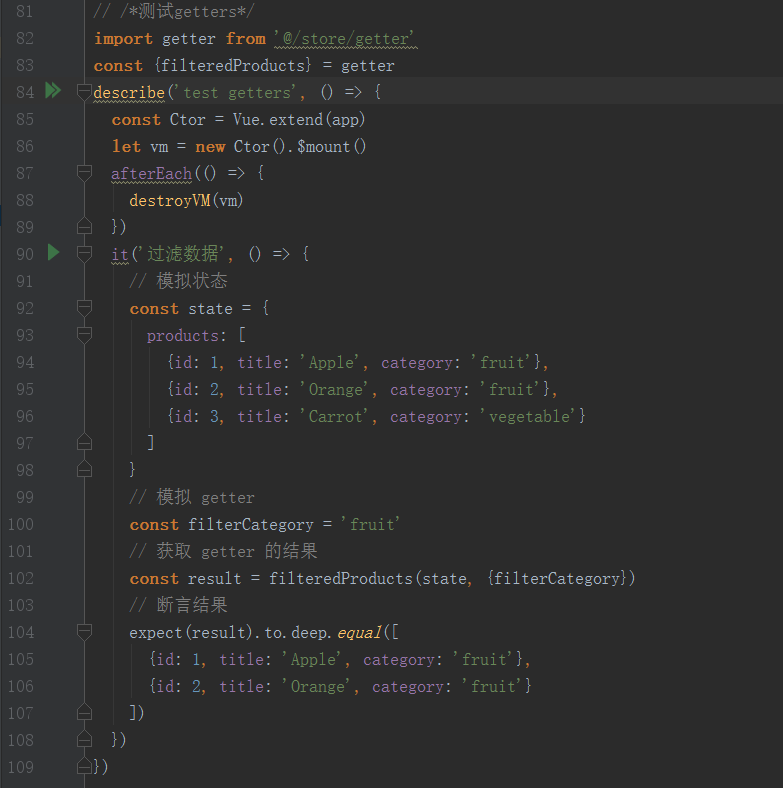


##### getters测试

getters.js

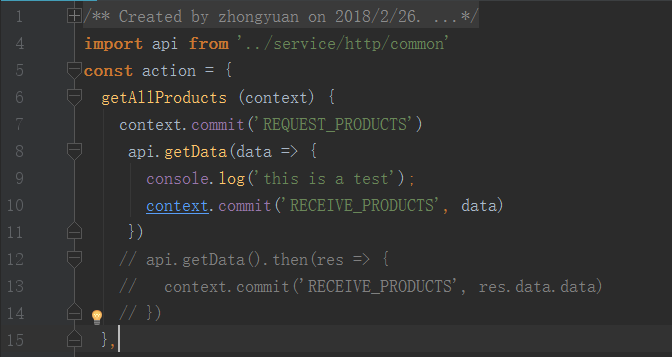


getters.sepc.js



##### actions测试

Actions.js

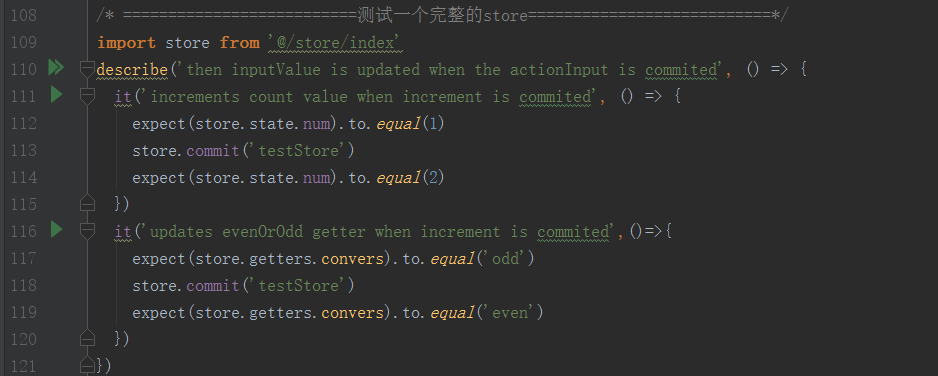


Actions.sepc.js





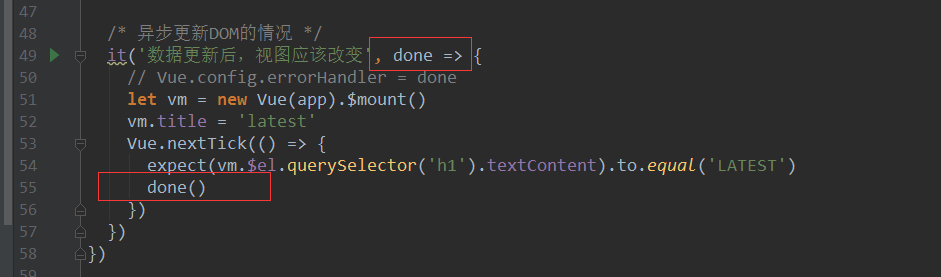
##### 测试运行中的store



### 五、使用过程中易错点

#### 异步操作

对异步操作，必须调用done()这个回调函数，告诉Mocha测试结束了，否则，mocha不知道测试是否结束，会一直等到报错.



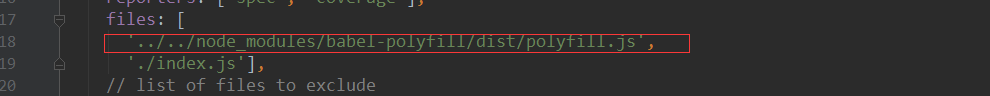
#### 2、跨域问题

须在karma.confi.js中配置proxies



#### 3、引入vuex

在项目中使用vuex时，需要下载babel-pollyfill（npm install babel-pollyfill），然后在karma.conf.js修改配置文件



### 六、学习参考文档

1、Vue.js官网

链接：<https://cn.vuejs.org/v2/guide/>

1. Vuex

链接：<https://vuex.vuejs.org/zh-cn/>

3、Vue.js学习系列六 —— Vue单元测试Karma+Mocha学习笔记

链接：<https://www.jianshu.com/p/073d25a3bba0>

4、Chai.js断言库API中文文档

链接：https://www.jianshu.com/p/f200a75a15d2

5、聊一聊前端自动化测试

链接：<https://github.com/tmallfe/tmallfe.github.io/issues/37>

6、vue-test-utils 是 Vue.js 官方的单元测试实用工具库

链接：https://vue-test-utils.vuejs.org/zh-cn/