# 项目地址：<https://oss.navercorp.com/zhaoyehong/test-framework>

# 1.为什么要进行代码测试

1. 节约时间

原本开发每修改一次代码，就需要从头到尾的测试，现在程序员每修改一次代码，只需要执行一遍测试用例，即可完成测试。

2.便于代码功能质量提高

程序员写的代码，黑盒测试是无法将所有代码逻辑测试完毕，因此出现了单元测试的。尽量覆盖代码逻辑，提高代码质量。

3.便于开发过失误操作

避免在项目进行二次，三次迭代过程中，可能不小心改到不该改的地方，自己却不知道。

4.便于回归测试

当进行迭代后，只需要跑一次测试用例，而不是页面一个一个去对比查找问题。

缺点：会增加很多代码，增加代码工作量。

# 2.TDD and BDD

**TDD： Test Drive Development**

先针对每个功能点抽象出接口代码，然后编写单元测试代码，接下来实现接口，运行单元测试代码，循环此过程，直到整个单元测试都通过。这一点和敏捷开发有类似之处。

对自己而言的作用：不只是为了不让别人觉得你写的代码有多糟糕，而是避免你错误地认为自己已经完成了编码，却留下一大堆Bug

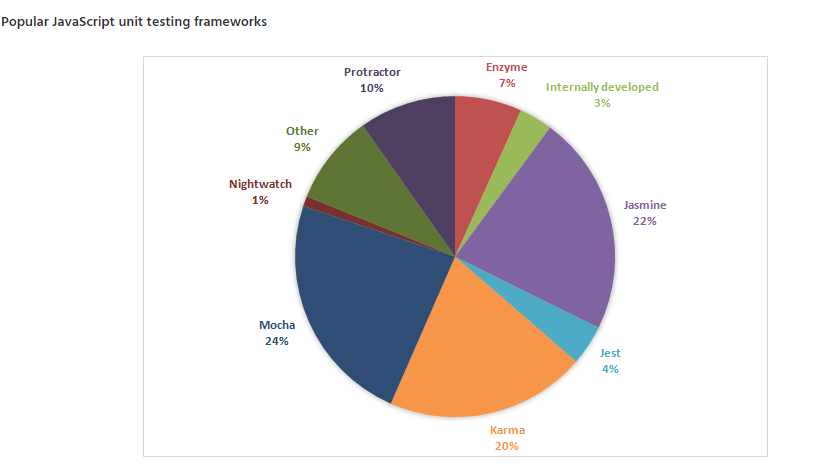
**BDD:Behavior Drive Development**

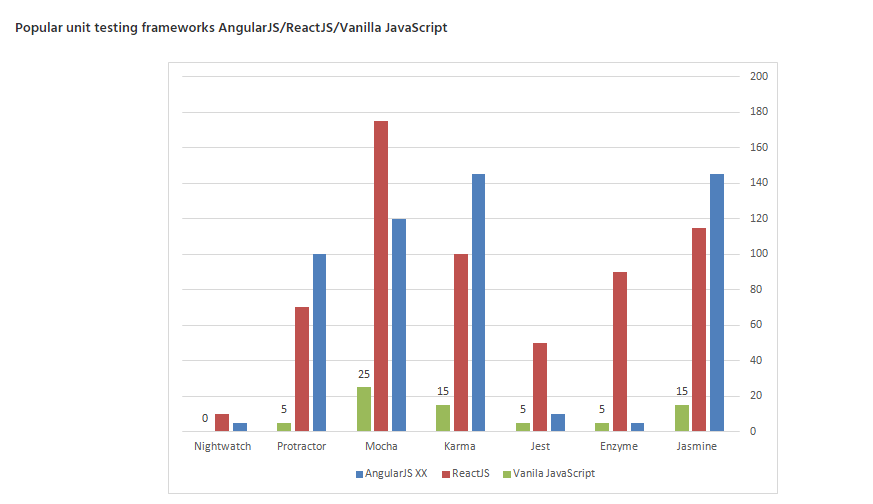
BDD会从业务目标着手，作用是把利益关系人、交付团队等不同方面的项目相关人员集中到一起，形成共同的理解，共同的价值观以及共同的期望值。与TDD相比更接近人的思考方式。有人认为它是验收测试。

# 3.技术选型

<https://blog.testproject.io/2016/09/01/front-end-development-unit-test-automation-trends2/>

这边文章分析了当前前端测试框架的欢迎程度。下面是两个重要图片。





最终我选择：

Mocha+ Chai + Sinon +Enzyme+ jsdom

[mocha](https://mochajs.org/)：一个JS的测试框架，类似于Java中的Junit、Python中的nose。

chai:是一个单元测试的验证框架，它有3种不同形式的校验：expect、should和assert。

sinon: 它可以对任何对象进行mock，不用再先解决一堆依赖和环境问题，把精力集中在要测试的单元上来编写 test case ，同时也缩短测试用例执行的时间，做到最小化测试。

enzyme:是专门用于测试React的一种三方库，对（react-addon-test-utils）进行了封装。

jsdom: 虚拟运行环境。不依赖浏览器为基础的测试运行器.它将会使用纯javascirpt创建一个假DOM,这样做让我们的测试更加快速。

# 4.环境搭建

mochawesome:用于生成漂亮的报告。

 babel-register: 能够使用 ES6 编写测试，需要在运行前对代码进行转译

a.包安装

npm install mocha –save-dev

npm install chai –save-dev

npm install sinon –save-dev

npm install mochawesome –save-dev

npm install babel-register –save-dev

b.修改package.js启动命令：

页面报告

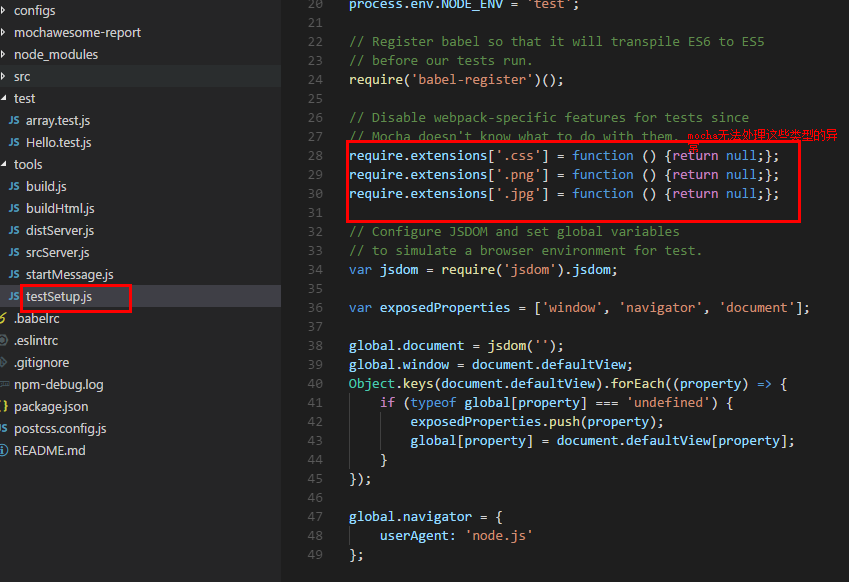


or

控制台报告



c.jsdom的文件



# 5.Mocha测试实例

## 5.1 同步测试

## 5.2 异步测试

注意坑：调用接口时使用mockjs会报错，必须使用服务器。

1. 使用done callback函数。
2. 直接返回promise方式
3. Mocha-generators方式（我推荐）
4. 4.async-await方式（我推荐）

## 5.3测试用例钩子

1.执行顺序

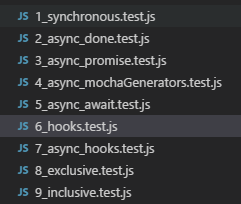
2.description hooks作用（当钩子函数抛出异常，能够很快定位哪里出现问题）

3.异步钩子

## 5.4 only,skip

过滤测试用例进行执行

例子详情请见demo



## 5.5 Mocha使用命令



--compilers js:babel-register ：通过es6编写测试代码

--recursive：测试文件递归测试，默认只测试一级目录

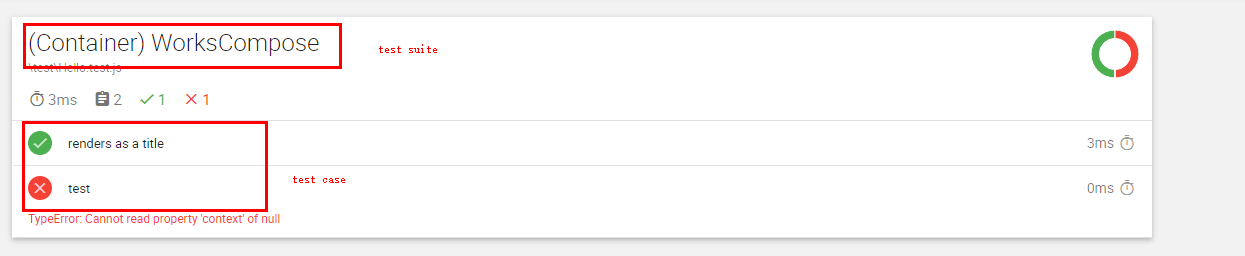
--reporter spec ：报告种类，spec 是在控制台显示

--reporter mochawesome ：报告种类，生成漂亮的html报告

tools/testSetup.js：jsdom的配置文件

\"test/\*\*/\*.test.js\" ： 测试文件目录

网页生成测试报告



# 6 Chai断言库

## 6.1简介

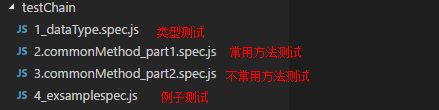
[Chai](http://chaijs.com/)是一个单元测试的验证框架，它有3种不同形式的校验：expect、should和assert。

expect和should的方式让写出来的测试代码更像自然语言，让业务人员也可以看懂。

assert方式是传统单元测试断言的方式，如果以前习惯写Java的单元测试会对这种方式比较熟悉。

## 6.2 实例

我主要是使用expect这方式。因此在写实例代码时全部使用expect. 详细实例请看demo.



## 6.3执行命令



# 7.Enzyme

## 7.1Enzyme测试介绍

Enzyme是专门用于测试react组件的。提供了三种测试方法。

1.Shallow: shallow方法就是官方的shallow rendering的封装,是浅渲染，不会间接地宣称子组件的行为。

2.Render: 方法将React组件渲染成静态的HTML字符串，然后分析这段HTML代码的结构，返回一个对象。它跟shallow方法非常像. 它是完整的DOM渲染。适用于需要完整的生命周期才能完全测试组件的情况。

3.Mount: mount方法用于将React组件加载为真实DOM节点，会渲染子组件。

## 7.2 react组件测试方式

对react的测试：是对我们的Reac应用渲染什么以及它如何响应用户交互的断言过程。可以从三方面入手：单元测试，功能测试，集成测试。  
react组件单元测试：

用一组给定的输入 (状态和属性), 输出是否符合我们对虚拟 dom 中 *应该* 的期望。

react组件功能测试：

重点是测试组件的行为。例如当用户登录后，显示一个LogOut按钮。没有登录显示Login按钮。

React组件集成测试：

之前的我们都是在虚拟dom之中进行测试，需要对实时运行的浏览器运行预期。

当测试响应组件时, 我们将测试我们对虚拟 dom 中包含的内容的期望, 以及实际 dom 中反映的内容。

**测试粒度选择：**

不能太细，太细会增加太多测试代码，会导致维护测试代码的困难。不能太粗，不然没有任何意义。在选择测试粒度时，我建议将重要正常流程进行测试。异常流程类型太多，难以维护。

## 7.3 使用Enzyme

# 8 Sinon

Sinon是一个测试实用程序库, 它为我们提供了一种编写间谍(spies)、存根(stubs)和 模拟(mock) 的方法。

1.使用jsdom而不使用Karma

2.component测试用例编写

a.模拟传入props，验证产生结果

2.reducer模块进行模块测试（模拟发送action，产出store）