有两次电面的时候，都被问道了如何测试程序的，现在想想，自己的经验真的是太不足了，最快速的方法其实一点也不好

手动测试：费时、费力，而且有些东西有可能并没有被记录下来。并不是一种best practice。

# 单元测试

测试驱动开发（test driven development）和行为驱动开发（behavior driven development）

* 关注点.测试驱动开发关注于所有的功能是否被正确实现。每一个功能都具备对应的测试用例。行为驱动开发关注于整体行为是否符合预期
* 表达方式不同。测试驱动开发表达方式更偏向于功能说明书的风格。行为驱动开发的表述方式更接近于自然语言的风格。

## 在nodejs中的测试框架

**测试框架的作用？**

用于测试服务，它本生并不参与测试，主要用于管理测试用例和生成测试报告，提升测试用例的开发速度，提高测试用例的的可维护性和可读性，以及一些周边性的工作。

**在Nodejs中的测试框架有哪些**

mocha

<https://github.com/mochajs/mocha>

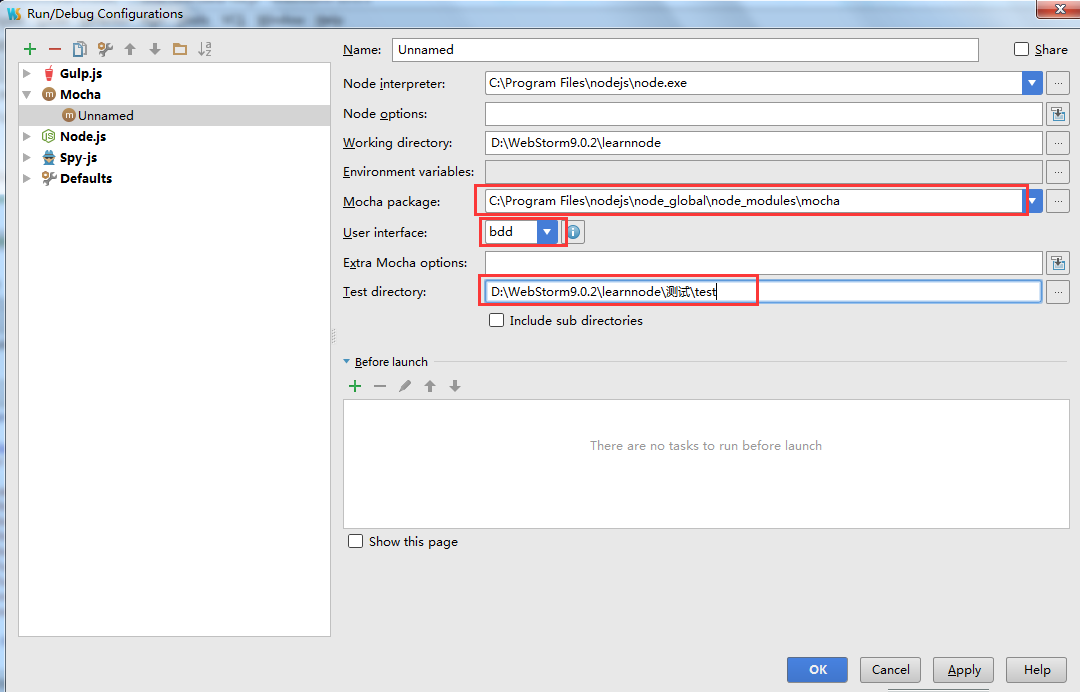
文档：<http://mochajs.org/>

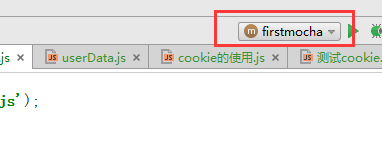
should

<https://github.com/shouldjs/should.js>

在webstorm下如何使用mocha

<http://www.tuicool.com/articles/2URvem6>





### should.js简介

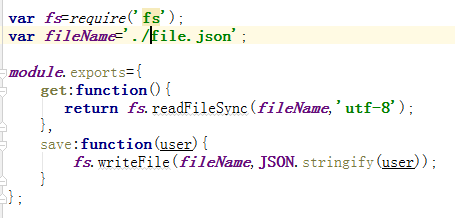
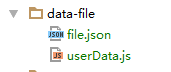
下面是使用should.js的一个实际例子，should.js具体的语法，可以参考官网。

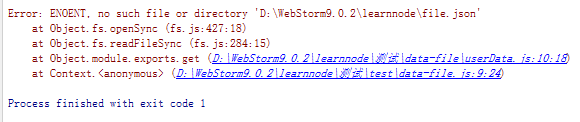
**var *should*** = require(**'should'**);  
**var *validator*** = require(**'../src/check\_functions.js'**);  
  
  
describe(**'IsNotEmpty'**, **function** () {  
 it(**'should return true when the value is "a" '**, **function** () {  
 ***validator***.IsNotEmpty(**'a'**).should.equal(**true**);  
 });  
 it(**'should return false when the value is null'**,**function**(){  
 ***validator***.IsNotEmpty(**null**).should.equal(**false**);  
 });  
 it(**'should return false when the value is undefined'**,**function**(){  
 ***validator***.IsNotEmpty(**undefined**).should.equal(**false**);  
 });  
 it(**'should return false when the value is "" '**,**function**(){  
 ***validator***.IsNotEmpty(**""**).should.equal(**false**);  
 });  
});  
  
describe(**'IsNumber'**,**function**(){  
 it(**'should return true when the input is 38'**,**function**(){  
 ***validator***.IsNumber(38).should.equal(**true**);  
 });  
 it(**'should return true when the input is -38'**,**function**(){  
 ***validator***.IsNumber(-38).should.equal(**true**);  
 });  
 it(**'should return true when the input is -3.8'**,**function**(){  
 ***validator***.IsNumber(-3.8).should.equal(**true**);  
 });  
 it(**'should return false when the input is "a" '**,**function**(){  
 ***validator***.IsNumber(**'a'**).should.equal(**false**);  
 });  
 it(**'should return false when the input is null '**,**function**(){  
 ***validator***.IsNumber(**null**).should.equal(**false**);  
 });  
});  
  
  
describe(**'IsInteger'**,**function**(){  
 it(**'should return true when the input is 38'**,**function**(){  
 ***validator***.IsInteger(38).should.equal(**true**);  
 });  
 it(**'should return true when the input is -38'**,**function**(){  
 ***validator***.IsInteger(-38).should.equal(**true**);  
 });  
 it(**'should return false when the input is -3.8'**,**function**(){  
 ***validator***.IsInteger(-3.8).should.equal(**false**);  
 });  
 it(**'should return false when the input is "a" '**,**function**(){  
 ***validator***.IsInteger(**'a'**).should.equal(**false**);  
 });  
 it(**'should return false when the input is null '**,**function**(){  
 ***validator***.IsInteger(**null**).should.equal(**false**);  
 });  
});

## 测试用例

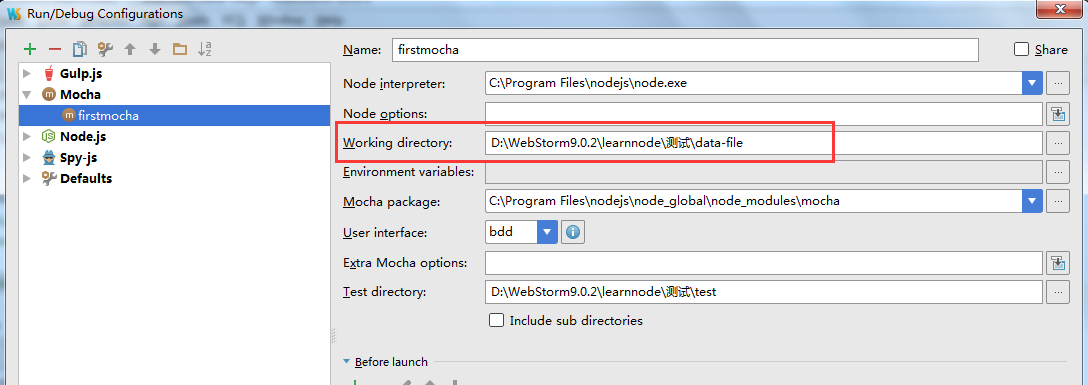
### 数据层面的测试。

测试的时候，文件路径应该要放在什么地方？





解决方法：设置“工作路径”



**should.js**

**断言：**

用于检查程序在运行时是否满足期望。

### 异步测试

* 异步编程中异常的解决  
  Node在处理异常上形成一种约定，把回调函数的第一个实参进行传递，如果第一个参数为空的话，则说明异步调用没有异常
* done的设置  
  Testing asynchronous code with Mocha could not be simpler! Simply invoke the callback when your test is complete. By adding a callback (usually named done) to it() Mocha will know that it should wait for completion.  
  在进行异步测试的时候，在回调函数的一个参数的名字为done，Mocha将会知道需要等待这个调用的完成。若果没有写参数done的话。Mocha不会等待。
* 为什么在异步测试的时候，需要进行超时设置？  
  回调函数的执行时间是无法预期的，我们无法知道done（）在什么时候执行，如果代码偶然出错的，导致done（）无法执行，会造成后续的所有测试用例都处于暂停状态。这并不是框架所需要的。  
  一般情况下mocha的默认超时时间是2000毫秒=2秒。  
  可以通过mocha –t <ms>设置所有用例的超时时间  
  在单个的测试用例it中，可以通过this.timeout(ms)来实现。