

---

# Table of Contents

Introduction	1.1
前言	1.2
從測試出發	1.3
誰需要前端測試	1.4
基礎環境安裝	1.5
建立 BDD 前端測試 - mocha	1.6
建立前後端共用測試 - mocha & npm test	1.7
淺談工具 - Codeception.js	1.8
前端整合測試環境安裝	1.9
第一個前端 e2e 整合測試	1.10

# 前端測試實做手冊

獻給所有正在希望持續導入前端測試，或者正在前端測試，整合測試大門徘徊的開發者們。提供一個簡單容易的開端，讓開發者瞭解前端開發的訣竅及入門方法。

從觀念到實際範例程式碼，讓開發者可以直接導入到目前專案，以輕鬆的方式進行前端測試，讓原本測試流程中缺少的那塊拼圖完整吧。

本書籍由 Caesar Chi 本人規劃編寫，特別感謝支持的家人及朋友們，以及上課的學員及設群的朋友給予的寶貴建議，讓本書資源能夠越來越豐富。

不論你之前的經驗是什麼，這邊都會希望接下來的流程，都從 `測試開始`，以 `測試先行` 的角度開始進行開發模式，跟著步驟跟著流程，一步一步往前進行，如果遇到任何問題，歡迎與我聯絡。

# 前端測試實做手冊

獻給所有正在希望持續導入前端測試，或者正在前端測試，整合測試大門徘徊的開發者們。提供一個簡單容易的開端，讓開發者瞭解前端開發的訣竅及入門方法。

從觀念到實際範例程式碼，讓開發者可以直接導入到目前專案，以輕鬆的方式進行前端測試，讓原本測試流程中缺少的那塊拼圖完整吧。

本書籍由 Caesar Chi 本人規劃編寫，特別感謝支持的家人及朋友們，以及上課的學員及設群的朋友給予的寶貴建議，讓本書資源能夠越來越豐富。

不論你之前的經驗是什麼，這邊都會希望接下來的流程，都從 `測試開始`，以 `測試先行` 的角度開始進行開發模式，跟著步驟跟著流程，一步一步往前進行，如果遇到任何問題，歡迎與我聯絡。

## 從測試出發

會開始讀取本書讀者，大部分都是已經具備基本測試經驗的開發者，或者是比較瞭解前端的開發者。

本書主要以實做為主，順勢導入觀念為輔，希望各位讀者能夠跟著一起進行動手做，從實做中開始瞭解測試的奧妙，天下武功，唯只有經過不斷的實戰與思考才能融會貫通，測試流程也是一樣。

透過本書的實做方式，以及範例程式，提供大家一個明確的路線，瞭解前端測試開發奧妙，讓大家知道如何將前端測試導入，提供更穩定的環境與服務給使用者，這才是身為開發者應該做到的事情。

## 測試，就從今天開始。

就讓我們開始進行一系列的實際開發流程，希望帶給各位不一樣的開發視野觀點，同時也希望各位能夠真正動手開始進行實做。

本書中所有程式範例都可以 <https://github.com/clonn/frontend-test-exercise> 進行取得。

如有任何錯誤之處，還請到[此處提出 issue](#) 進行討論。

# 誰需要前端測試

在開始進行之前，首先定義一下你有以下的問題

- 每次程式進行更新感覺到不安
- Bug 永遠跟不完，修 A 壞 B 的狀況時常發生
- 追求完美程式架構者
- 苦尋不著前端測試方法者

如果你有以上症狀，或者有相同感受者，請繼續研讀此書，將會令你獲益良多。

希望透過前端測試，讓你測試環境更為完美，到達整體流程的境界

## env install

以下列表為基本環境必須要安裝項目，

- git - [git install](#)
- node.js - <https://nodejs.org/en/> (請以 LTS 版本為主)
- bower - <https://bower.io/#install-bower>
- Web server - chrome extension
- Java

安裝完成後，開始先試著透過 `node` `npm` 安裝整體接下來會用到的指令，

```
npm install -g mocha  
npm install -g grunt-cli  
npm install -g gulp  
npm install -g codeceptjs  
npm install -g webdriver-manager
```

安裝完以上指令之後，就可以透過例如 `mocha`，`gulp` 等指令直接在終端機上面執行，接下來的每個動作都會相互在編輯器，及終端機上面互相執行，所以當各位如果有看到 \$ 表示指令是在終端機底下執行。

# Mocha 基本前端單元測試

首先我們要建立基本前端單元測試環境，因此需要透過 `bower` 進行 mocha 安裝，請輸入以下指令。

```
npm i -g bower
```

安裝完成後，我們接著就會採用 bower 進行安裝 mocha, chai 等套件，而且會將這些套件使用於前端開發測試流程中。

## 什麼是 Mocha

- [mocha.js](#)

關於官方簡單描述如下，

Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

Mocha 是一種基礎於 JavaScript 的測試框架，同時讓 Node.js 及前端 JavaScript 進行測試使用，使用情境並不限定於前端或者後端，是一個很方便，輕巧的測試框架。

接下來我們都會透過 mocha 進行前端測試，請跟步驟一步一步進行。

## 設定 mocha 環境

可以跟著 程式碼中的 `03_mocha-basic` 範例程式進行，當然你也可以透過複製，轉移到自己的框架流程中，讓自己方便使用。

接下來我們要透過剛才安裝好的 `bower` 進行安裝 mocha, chai 套件，請在 terminal 輸入底下指令，

```
cd 03_mocha-basic  
bower i mocha  
bower i chai
```

安裝完成後會看到資料夾中，會顯示結果如下，表示安裝完成

```
bower cached      https://github.com/mochajs/mocha.git#3.0.  
2  
bower validate    3.0.2 against https://github.com/mochajs/  
mocha.git#*  
bower cached      https://github.com/chaijs/chai.git#3.5.0  
bower validate    3.5.0 against https://github.com/chaijs/c  
hai.git#*  
bower install     chai#3.5.0  
bower install     mocha#3.0.2  
  
chai#3.5.0 bower_components/chai  
  
mocha#3.0.2 bower_components/mocha
```

並且會將安裝好的內容放置於 `bower_components` 資料夾中，也就是待會我們會使用引入到前端的測試套件。

接著建立一個檔案名稱為 `index.html`，內容如下，

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <div id="mocha"></div>
    <script src=".bower_components/chai/chai.js"></script>
    <script src=".bower_components/mocha/mocha.js"></script>
    <script src=".js/add.js"></script>
    <script>mocha.setup('bdd')</script>
    <script src=".test/add.spec.js"></script>
    <script>
        mocha.run();
    </script>
</body>
</html>
```

主要結構載入了 mocha, chai 兩個套件，並且設定 mocha 測試條件，同時將主要程式 `js/add.js` 一個簡易加法程式直接引入到頁面中。

接著引入 `test/add.spec.js`，這段測試程式，主要用於驗證前端邏輯測試結果。

通常命名上，我們會將測試程式放置於 `test` 資料夾中，並且測試檔案都會取名為 `xxx.spec.js` 這是一個約定的方式，當然你也可以使用自己習慣方式進行命名，名稱並不是主要的關鍵，只要日後可以進行識別都會是很棒的名稱。

最後再設定 `mocha.run()` 就會執行程式，很快的就可以把程式運行起來。

接著我們再繼續將 `test/add.spec.js` 的內容先補起來，再接著實做 `add.js` 本身的程式架構，接下來都會採用測試先行的方式將測試流程補充完畢之後，才會進行實際 `function` 的編寫。

## Test first 流程

建立一個 test 資料夾，並且在資料夾中建立名為 add.spec.js 的檔案，裡面主要放置了 add.js 相關的測試範疇。

```
mkdir test  
touch test/add.spec.js
```

add.spec.js 裡面測試內容如下

```
var expect = chai.expect;  
var should = chai.should();  
describe('Test add function', function() {  
    it('test simple add basic', function () {  
        var result = add(2, 1);  
        result.should.be.equal(3);  
    })  
});
```

整體架構規範採用於 mocha 框架流程，當我們採用 mocha 之後，就直接可以使用 describe it 這兩個描述句型，主要用於描述 test 本身的用法及範圍，本身除了就是程式之外，透過友善文字性的描述，可以之後維護者更容易瞭解當初的場景，以及設定的測試情境。

在測試中，我們直接進行呼叫 add(2, 1) 直接進行 add 程式呼叫，透過 chai 驗證結果。

當我們設定好 add 程式中的預期輸入及預期輸出之後，準備回到 add 程式本身，我們開始進行 add.js 主要程式內容編寫，指令如下，

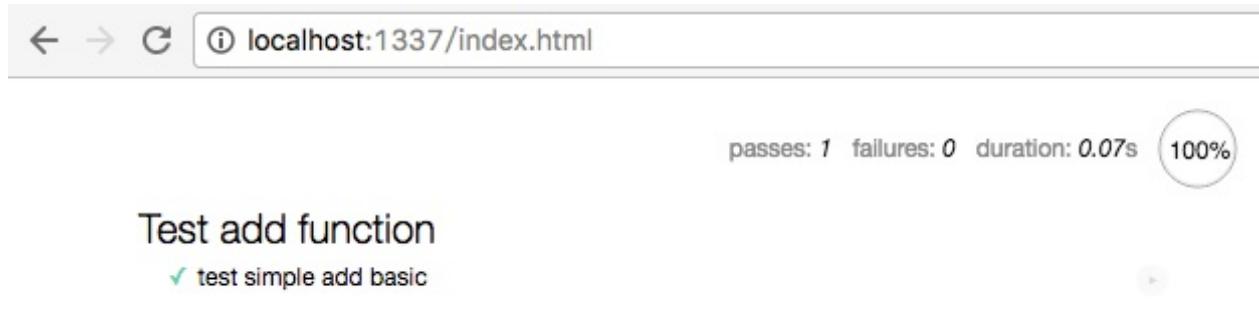
```
mkdir js  
touch js/add.js  
# then edit js/add.js
```

修改 `js/add.js` 修改內容如下，

```
function add(num1, num2) {  
    return num1 + num2;  
}
```

根據實做過程，`add` 是一個非常簡單的加法函式，直接將加入的資料進行回傳，讓我們可以看到實際的結果產出。

接著透過前端 `index.html` 結果的執行，可以看到實際產出的頁面會如下，表示為驗證結果正確。



恭喜各位，完成第一個 `test first` 的前端開發流程。

這是一個非常簡單不過的測試流程，實際上我們在進行測試開發時都應該保持同樣的方式及節奏，讓開發流程盡量符合期待及規範，保持有一定的預期輸入及預期輸出，讓程式保持一定的簡潔度，才有辦法進行測試。

## 讓 mocha 同時前後端執行

剛才已經有設定了關於前端執行項目，但是 mocha 實際上也可以在後端進行測試，我們就來改善一下設定，讓我們的程式也可以照常進行吧

```
npm init
```

輸入後過程中會有一些對話，內容可以全部留白，內容大致上會是如下描述，

```
name: (mocha-basic)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/chicaesar/workspace/mokayo/frontend-test-exercise/mocha-basic/package.json:

{
  "name": "mocha-basic",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Is this ok? (yes)

接著環境中就會多出一個 `package.json` 針對於 node.js 後端專案的設定檔。

接下來就開始繼續安裝後端測試套件，並且設定到環境中吧

```
npm i bower chai --save-dev
```

## mocha 前後端共用測試

前面我們已經開始了前端的測試流程，現在我們開始可以進入到『前後端共用』模組架構，讓開發的可以是前端端共用結構，讓測試可以透過 mocha 指令進行。

首先在開始之前，我們先安裝 mocha 終端機指令，

```
npm install -g mocha  
npm install mocha  
npm install chai
```

## 修改成前後端共用測試 scripts

我們開始修改原本已經建立好的 `test/add.spec.js`，將整體程式碼進行 `function closure` 之後，再引入判斷條件，讓程式判別目前是在前端還是後端執行，而這邊主要的特徵在於 `node.js` 的環境中，每個獨立檔案都可以成為一個 `module.exports` 條件，依據這種特性我們可以進行測試，瞭解其中執行方向。

全部程式碼修改完，內容如下，

```
(function (isNode) {
    var chai, add;

    if (isNode) {
        chai = require('chai');
        add = require('../js/add.js');
    } else {
        add = window.add;
        chai = window.chai;
    }

    chai.should();
    describe('Test add function', function() {
        it('test simple add basic', function () {
            var result = add(2, 1);
            result.should.be.equal(3);
        });

        it('add with a string', function () {
            var result = add(2, 'kerker');
            result.should.be.equal(2);
        })
    });
}

}(typeof module !== 'undefined' && module.exports));
```

接著進行設定 `js/add.js`，修改 `add function`，這邊一樣依據 `module.exports` 這樣的特性，進行判斷，如果有 `module.export` 就會把 `add` 這個函式直接引入，如果沒有的話，就將這個 `add function` 視為一個前端執行函式。

`js/add.js` 修改後內容如下，

```
function add(num1, num2) {  
    num1 = (typeof num1 !== 'number') ? 0 : parseInt(num1, 10);  
    num2 = (typeof num2 !== 'number') ? 0 : parseInt(num2, 10);  
    return num1 + num2;  
}  
  
var module = module || {};  
if (module && module.exports)  
    module.exports = add;
```

修改完後，目前會發現架構裡面可以直接透過 `mocha` 進行測試，終端機中測試指令如下，單獨透過 `test/add.spec.js` 進行單獨測試，指令如下，

```
mocha test/add.spec.js
```

測試後，驗證結果正確，將會是如下的畫面，

```
Test add function  
  ✓ test simple add basic  
  ✓ add with a string
```

```
2 passing (8ms)
```

看到上面描述，表示進行 `add.js` 函式中驗證結果正確，恭喜各位完成一個前後端共用的測試流程。

當我們這樣開始編寫後，此時測試單元已經可以從單純前端驗證，直接透過後端進行處理，當然更深一步我們可以繼續透過類似 `webpack`, `browserify` 之類的工具編譯出前後端共用函式，模組，這個部分就不再繼續深入討論。

如果各位有興趣，可以查詢關於前後端共用模組建置，及編譯方式，希望各位能夠試著打造前後端共用的環境。



# 介紹 CodeceptJS

MODERN ERA ACCEPTANCE TESTING FOR NODEJS

CodeceptJS 是一個基於 WebDriver 全新的 E2E 測試框架。以使用者角度進行使用者行為測試，進行簡單使用者操作步驟來編寫測試

<http://codecept.io/>

## 安裝方式

```
npm install -g codeceptjs
```

透過 npm 指令就可以快速將 codeceptjs 安裝到環境中，除了安裝 codeceptjs 之外，還要安裝底層 e2e 執行環境，

例如需要優先安裝 webdriverio ，

```
npm install webdriverio
```

## 執行方法

安裝好 coeceptjs 之後，可以開始進行設定環境，透過 codeceptjs 指令

```
codeceptjs init
```

開始進行設定專案，目前我們僅需要進行處理 webdriverio 就選擇 webdriverio 項目即可，以及確定資料夾路徑。

設定流程大約會如下描述，

```
Test root is assumed to be $PATH_ENV

Welcome to CodeceptJS initialization tool
It will prepare and configure a test environment for you

Installing to $PATH_ENV/
? Where are your tests located? ./*_test.js
? What helpers do you want to use? WebDriverIO
? Where should logs, screenshots, and reports to be stored? .
/output
? Would you like to extend I object with custom steps? No
Configure helpers...
? [WebDriverIO] Base url of site to be tested http://localhost
? [WebDriverIO] Browser in which testing will be performed chrome
Config created at $PATH_ENV/codecept.json
Directory for temporary output files created at `_output`
Almost done! Create your first test by executing `codeceptjs
gt` (generate test) command
```

設定好之後，就會看到資料夾中多出了一個 `codecept.json` 檔案，這也就是 `codecept.json` 的執行環境。

大概描述內容如下，

```
{  
    "tests": "./*_test.js",  
    "timeout": 10000,  
    "output": "./output",  
    "helpers": {  
        "WebDriverIO": {  
            "url": "http://localhost",  
            "browser": "chrome"  
        }  
    },  
    "include": {},  
    "bootstrap": false,  
    "mocha": {},  
    "name": "06_e2e_codeceptjs"  
}
```

如此一來就完成 codeceptjs 環境設定，記得開始測試之前一定要先安裝 selenium 以及模擬環境。

## 開始進行 codeceptjs

在環境完成安裝設定之後，我們就可以開始編寫第一個 e2e 測試腳本。

透過 codeceptjs 指令進行新增腳本，

```
codeceptjs gt
```

透過 gt 參數，代表 generator test 的意思，開始進行測試的編輯，並且同時設定檔案名稱為 `first_test.js`，因為我們前面已經設定 test 執行路徑將會讀取 `*_test.js`，所以命名規則必須要符合才有辦法進如執行環境中。

```
Test root is assumed to be $PATH_ENV
Creating a new test...
-----
? Filename of a test first_test.js
? Feature which is being tested First test.js
Test for first_test.js was created in $PATH_ENV/first_test.js
```

如上述設定完成後，就會看到新增一個基本的測試環境可以提供使用，

## 第一個測試腳本

完成上述描述後，我們就可以開始進行第一個測試腳本，

```
Feature('First test.js');

Scenario('test something', (I) => {
    I.amOnPage('https://www.google.com.tw/');
    I.see('Google');
});
```

`I.amOnPage` 表示瀏覽器即將前往的網址。

I.see 為最後確認 assertion，確定驗證資訊是否與預測相同，目前會直接找到 page title 進行驗證。

我們可以透過底下指令，就會看到立即開啟一個 google page，並且進行驗證的動作。

```
codeceptjs run --steps
```

# github 程式體驗

透過底下描述，我們可以快速瞭解 github 頁面流程，以及透過快速進行頁面測試，檔名設定為 `github_test.js`，

```
Scenario('search', (I) => {
  I.amOnPage('https://github.com/search');
  I.fillField('Search GitHub', 'mokayo');
  I.pressKey('Enter');
  I.see('miiixr/mokayo', 'a');
});

Scenario('signin', (I) => {
  I.click('Sign in');
  I.see('Sign in to GitHub');
  I.fillField('Username or email address', 'user@user.com');
  I.fillField('Password', '123456');
  I.click('Sign in');
  I.see('Incorrect username or password.', '.flash-error');
});

Scenario('register', (I) => {
  within('.js-signup-form', function () {
    I.fillField('user[login]', 'User');
    I.fillField('user[email]', 'user@user.com');
    I.fillField('user[password]', 'user@user.com');
    I.fillField('q', 'aaa');
    I.click('button');
  });
  I.see('There were problems creating your account.');
  I.click('Explore');
  I.seeInCurrentUrl('/explore');
});
```

檔案名稱設定完後，一樣進行執行，將會看到瀏覽器成果。

```
codeceptjs run --steps
```