

程式人《十分鐘系列》



用 JavaScript 實踐《軟體工程》的那些事兒！

陳鍾誠

2017 年 3 月 10 日

四年前

- 我透過 node. js 開始重新接觸 JavaScript

然後

- 我發現 ...

JavaScript

- 總是給我一次又一次的驚奇

四年來

- 我就像劉姥姥二進大觀園一樣
還是沒能把大觀園給看遍！

但是即使沒看遍

- 也已經有非常多的收穫了！

今天

- 我想透過 JavaScript
- 來介紹一下我所知道的...

關於《軟體工程》

- 還有那些軟體開發的工具...

首先我要介紹的

- 是 JavaScript 的 IDE 開發工具

雖然 JavaScript 的 IDE 有很多

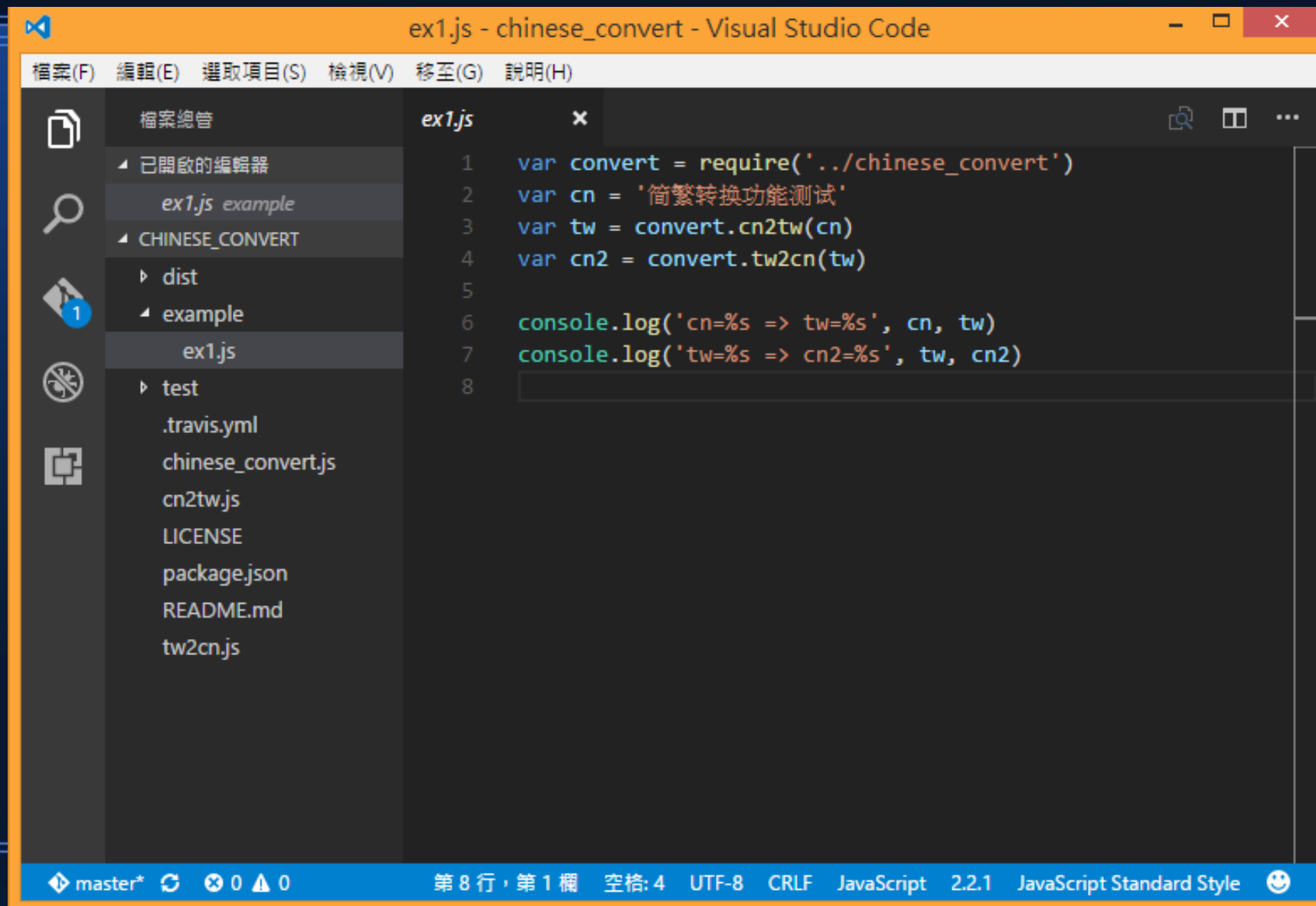
- 像是 WebStorm、Eclipse、Netbeans 等等
- 但是我目前採用的，是微軟免費的 Visual Studio Code (VS Code)

必須注意的是 Visual Studio Code \neq Visual Studio

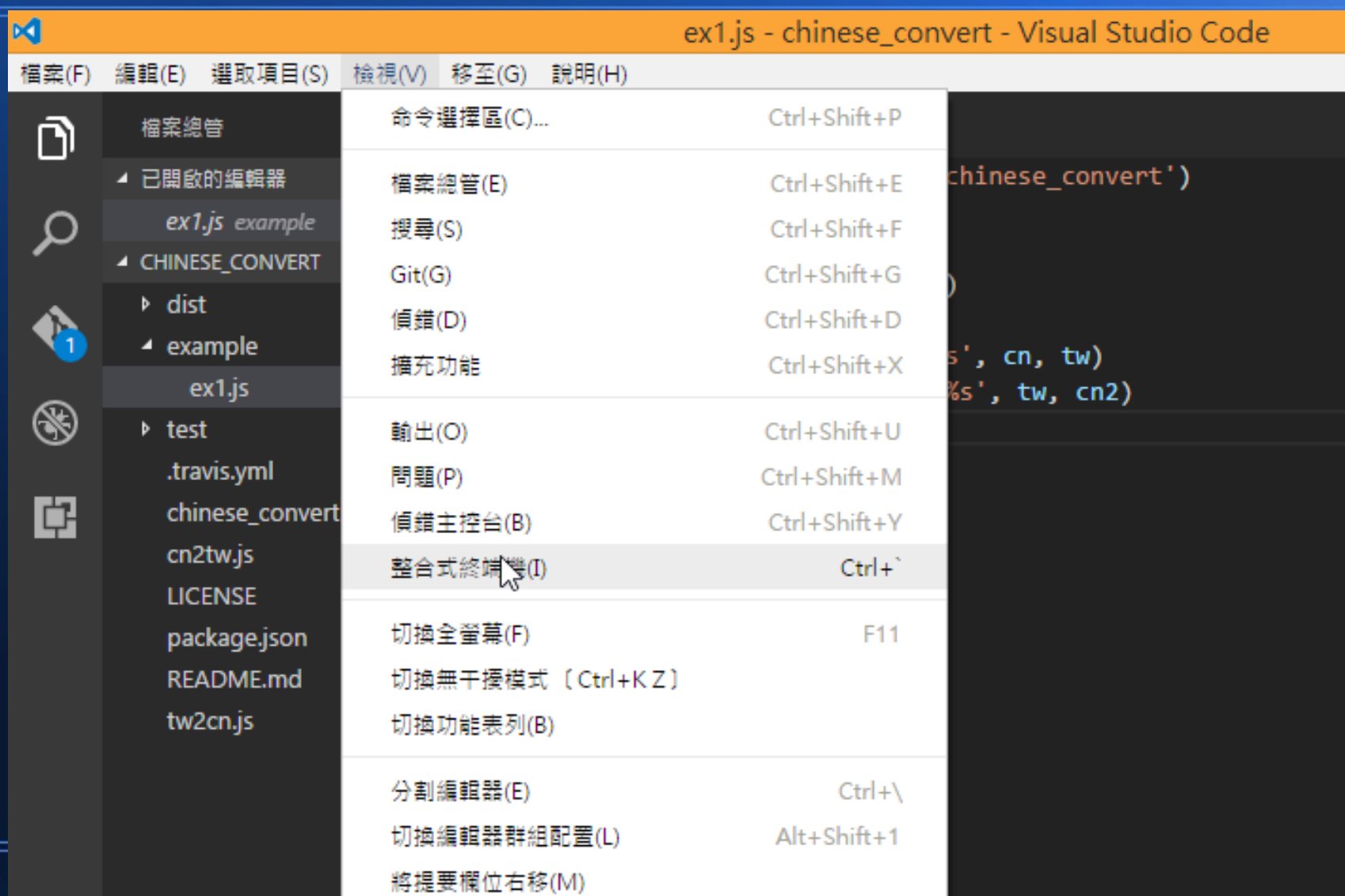
雖然之前

- 我都是用 Notepad++ 或 Sublime 這樣的編輯器開發 JavaScript
- 但是最近愈來愈覺得 VS Code 不只是個編輯器，反而比較像是 IDE

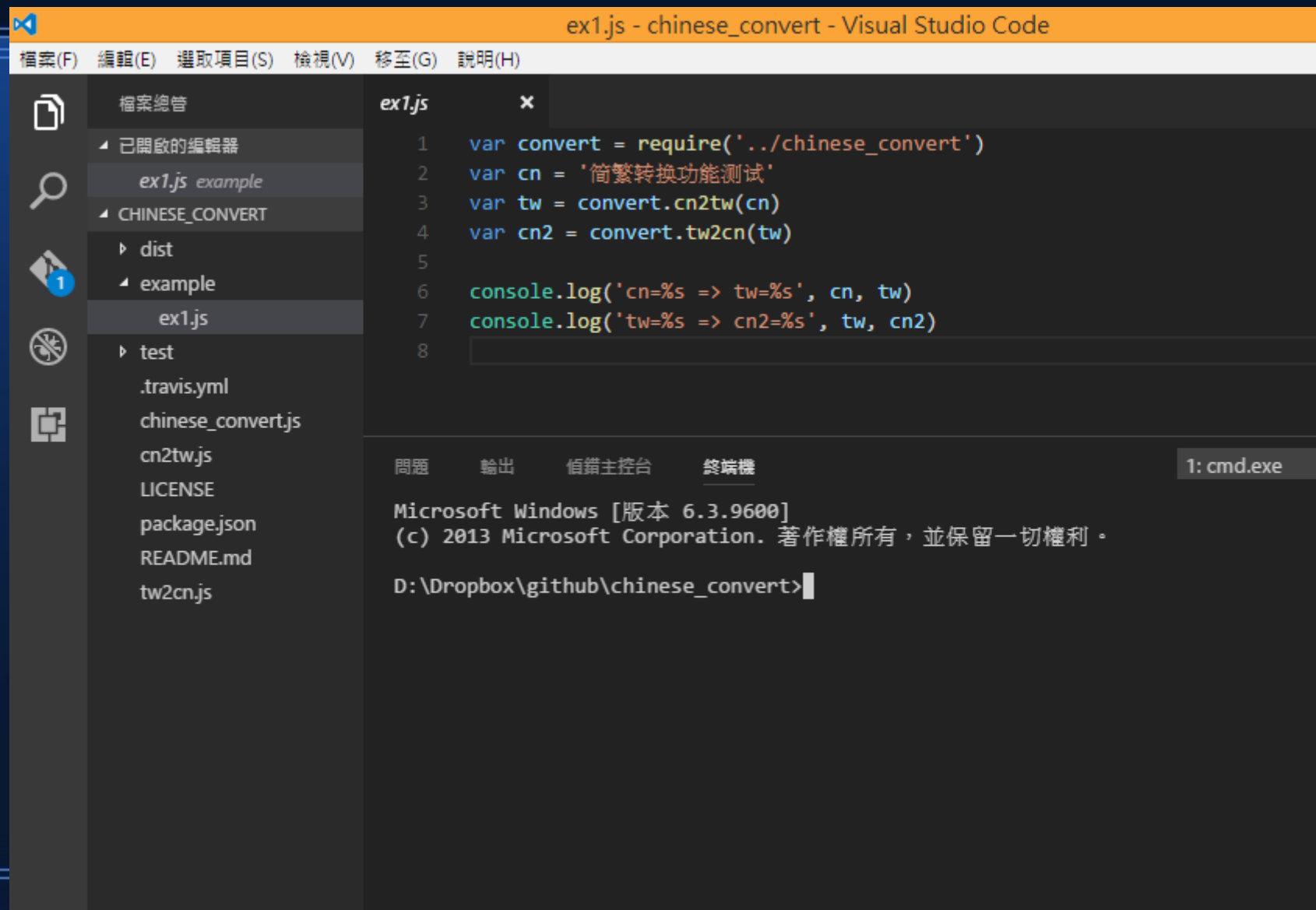
以下是 VS Code 的畫面



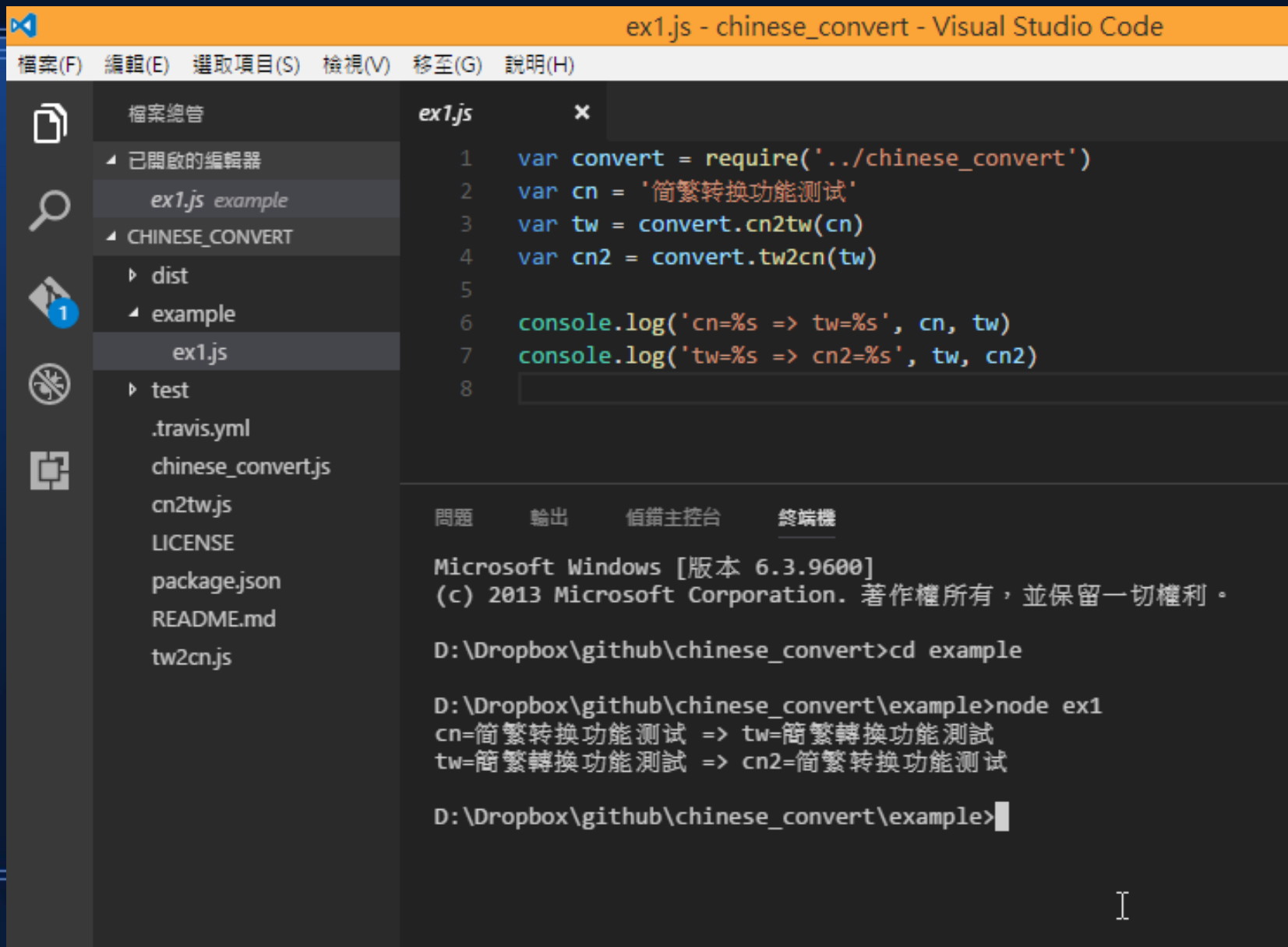
如果您選擇《檢視 / 整合式終端機》



就會有《命令列》出現



您可以邊寫程式邊執行



The screenshot shows the Visual Studio Code interface with a file named `ex1.js` open in the editor. The file contains JavaScript code for a simple Chinese-to-Taiwanese character conversion. The left sidebar shows the file explorer with the project structure. The bottom panel shows the terminal output of running the script.

```
ex1.js - chinese_convert - Visual Studio Code
檔案(F) 編輯(E) 選取項目(S) 檢視(V) 移至(G) 說明(H)

檔案總管
├─ 已開啟的編輯器
│   └─ ex1.js example
├─ CHINESE_CONVERT
│   └─ dist
│       └─ example
│           └─ ex1.js
├─ test
├─ .travis.yml
├─ chinese_convert.js
├─ cn2tw.js
├─ LICENSE
├─ package.json
├─ README.md
└─ tw2cn.js

ex1.js
1  var convert = require('../chinese_convert')
2  var cn = '簡繁轉換功能测试'
3  var tw = convert.cn2tw(cn)
4  var cn2 = convert.tw2cn(tw)
5
6  console.log('cn=%s => tw=%s', cn, tw)
7  console.log('tw=%s => cn2=%s', tw, cn2)
8

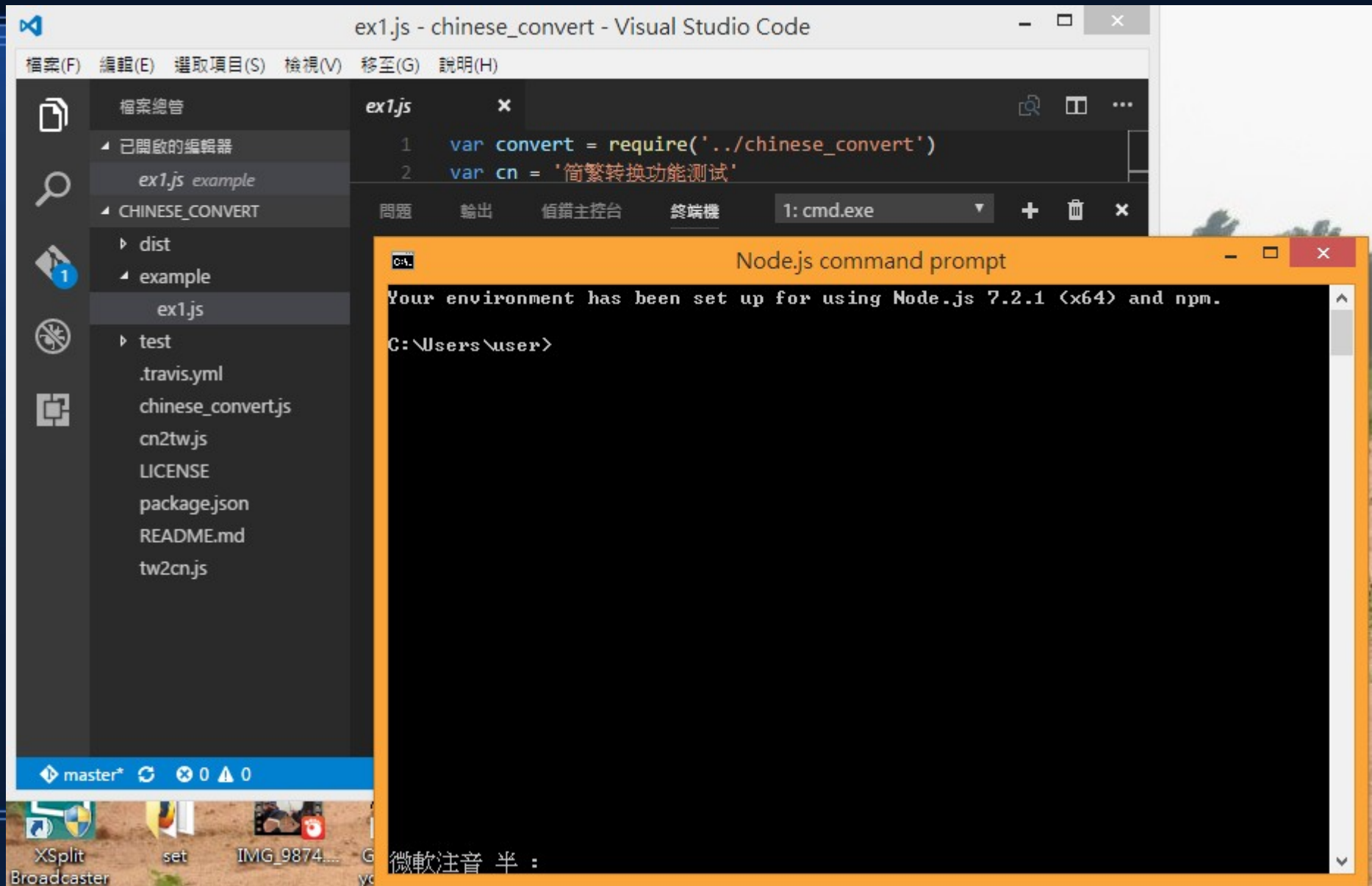
問題 輸出 偵錯主控台 終端機
Microsoft Windows [版本 6.3.9600]
(c) 2013 Microsoft Corporation. 著作權所有，並保留一切權利。

D:\Dropbox\github\chinese_convert>cd example

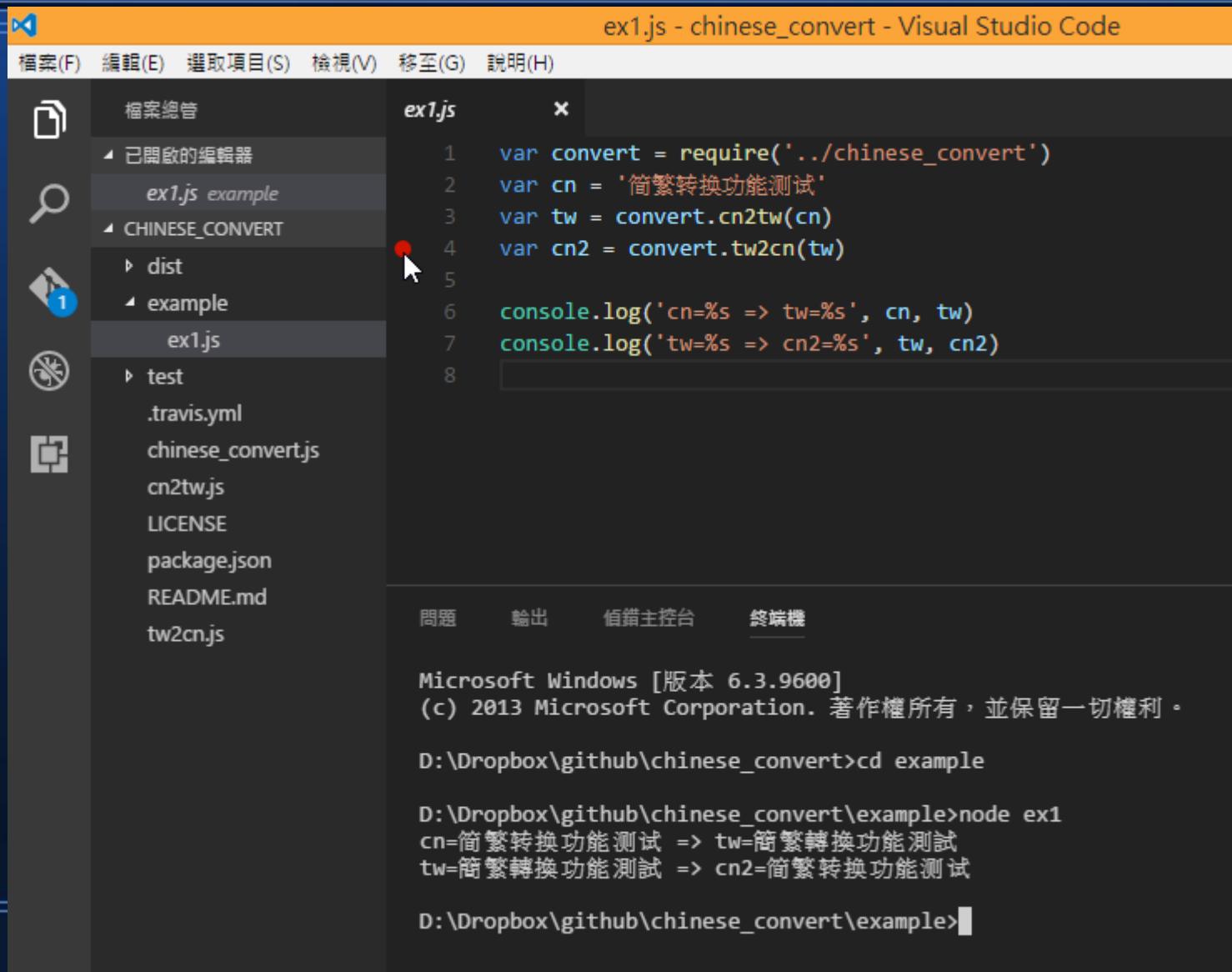
D:\Dropbox\github\chinese_convert\example>node ex1
cn=簡繁轉換功能测试 => tw=簡繁轉換功能測試
tw=簡繁轉換功能測試 => cn2=簡繁轉換功能测试

D:\Dropbox\github\chinese_convert\example>
```

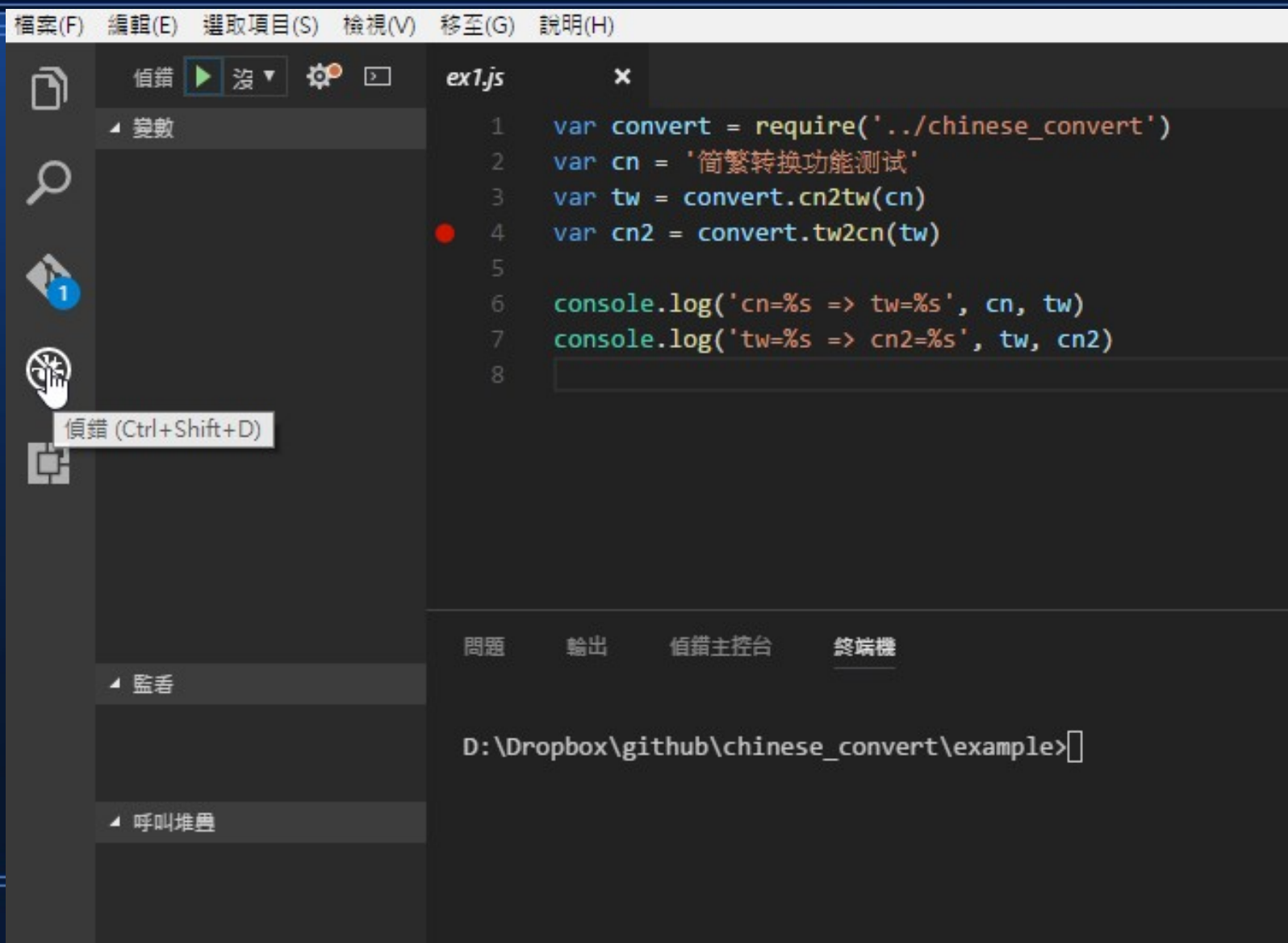
不需要一直切換視窗



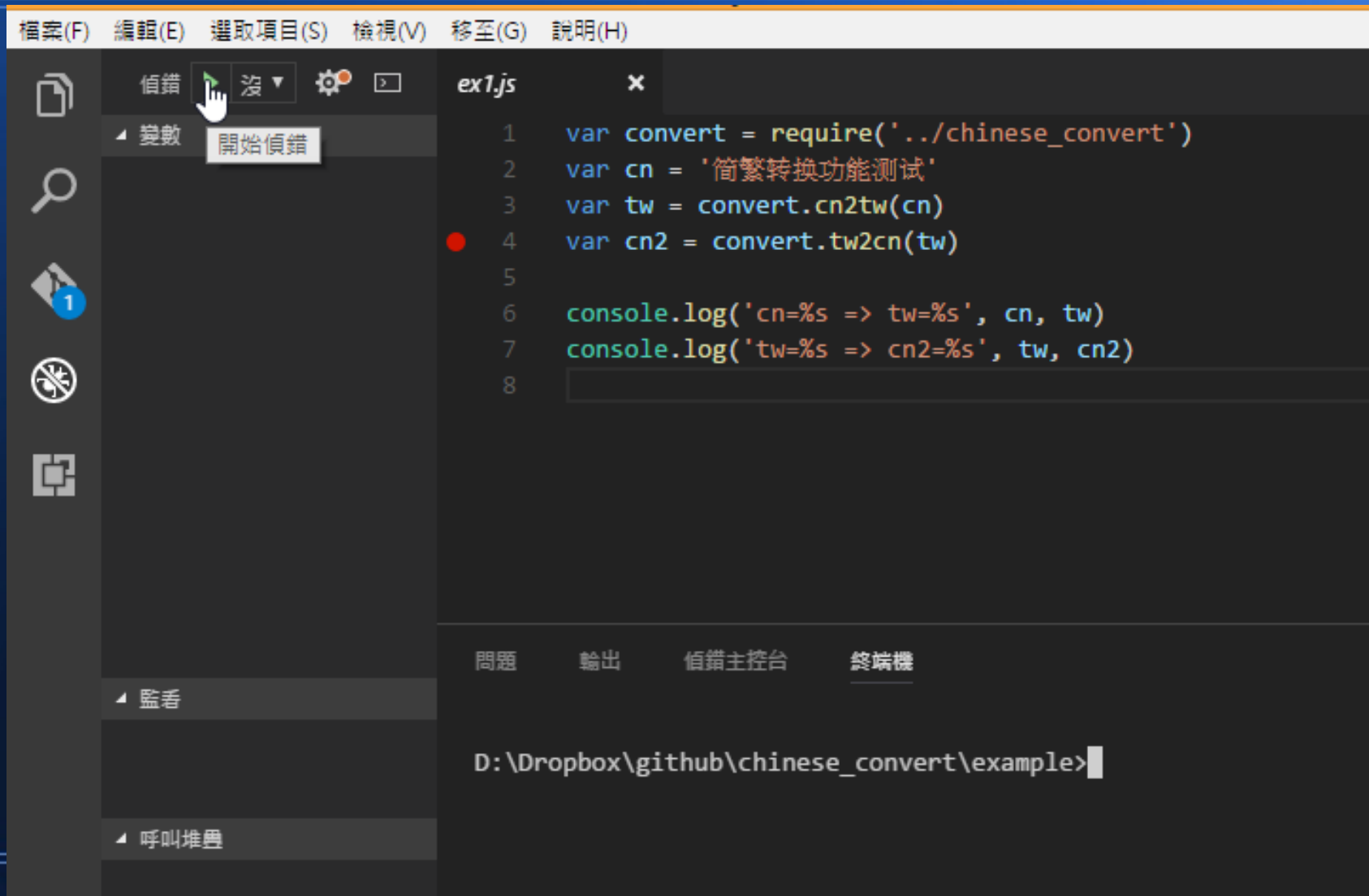
您可以在程式碼前點一下 (就能設定中斷點)



接著按下那個甲蟲符號 (會出現偵錯環境)



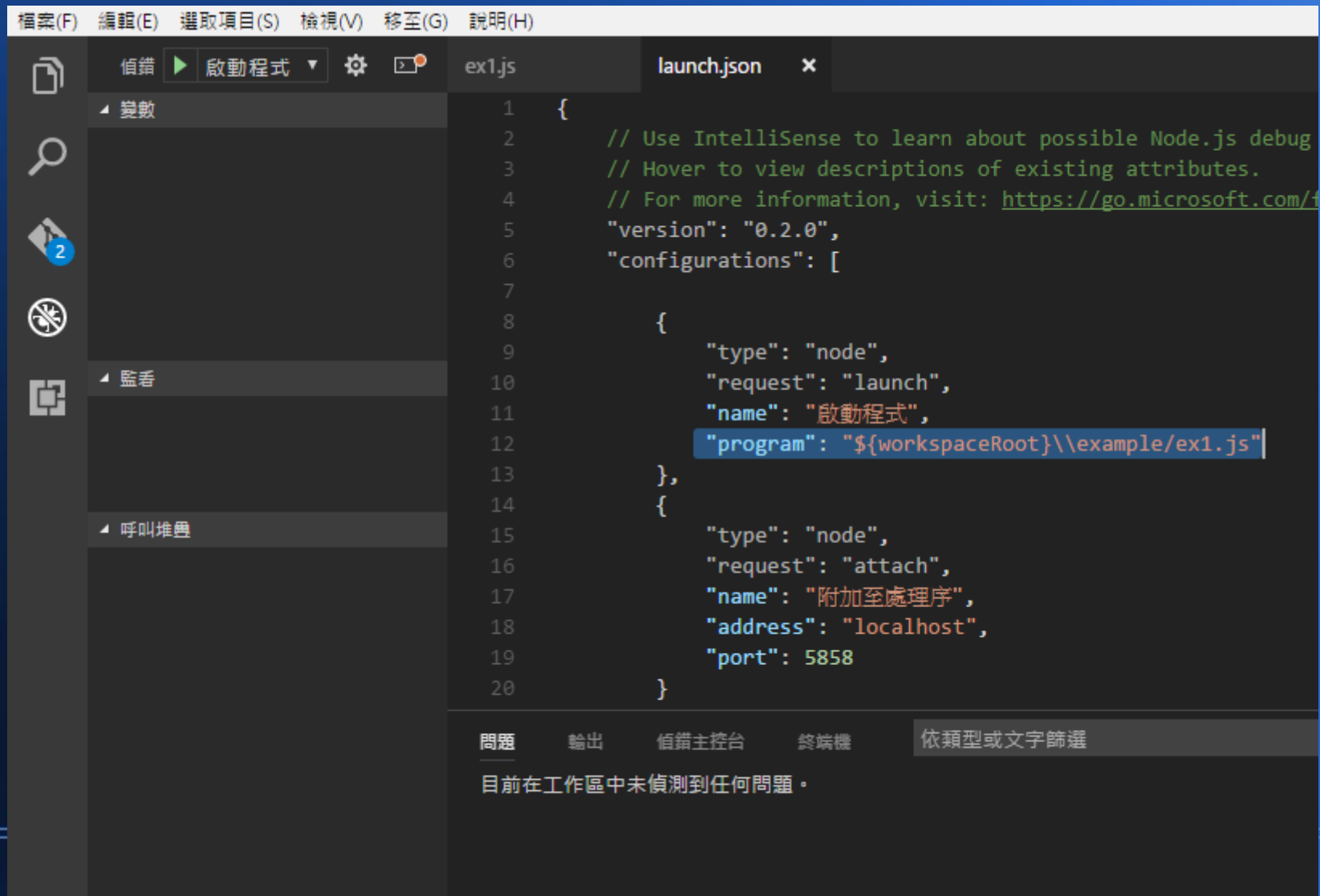
然後按下綠色箭頭
(就可以開始除錯了)



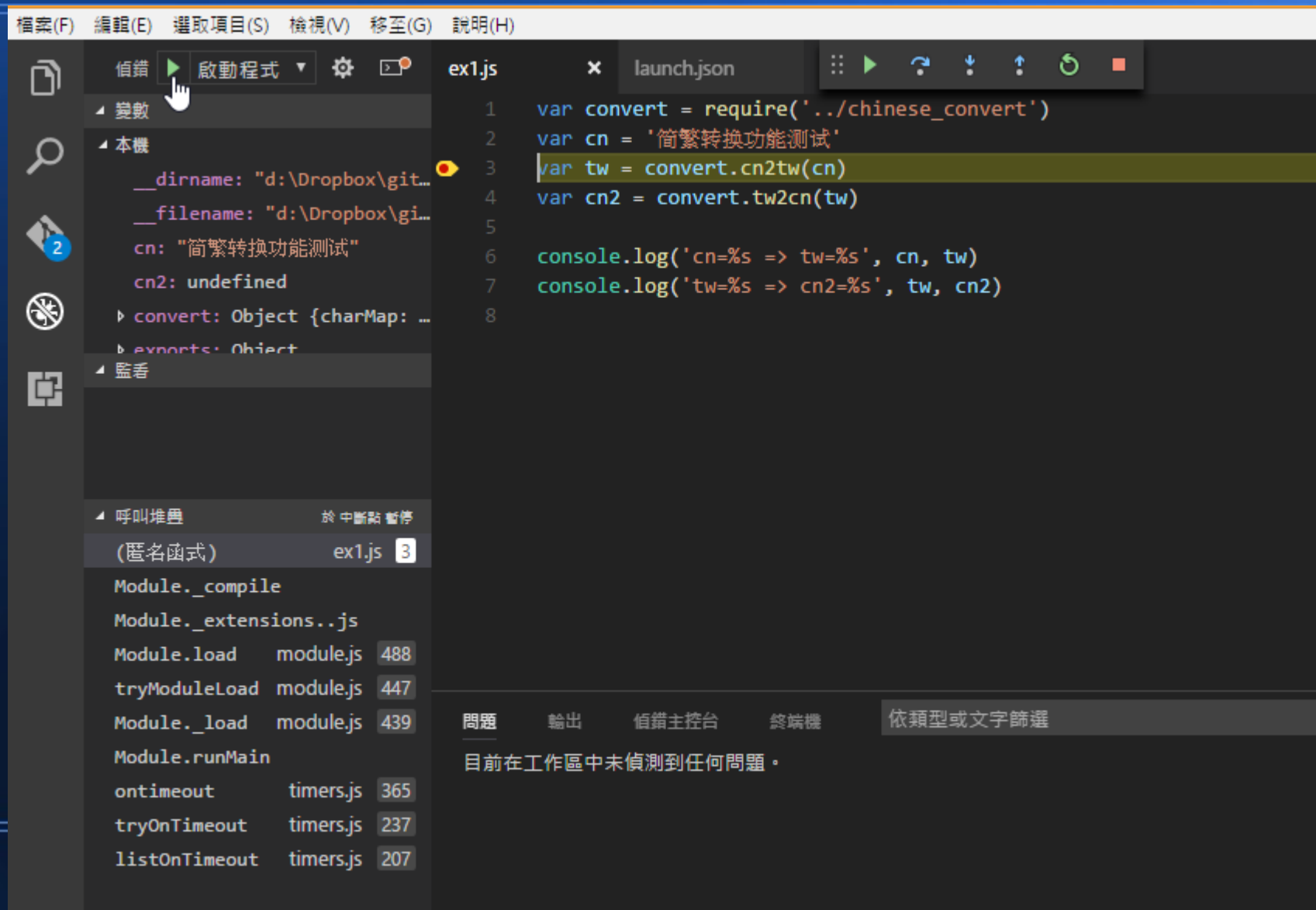
不過、當你開啟整個資料夾當專案時

- VS Code 不知道你的哪一個程式是待測試的主程式，所以會亂猜一個並寫入 launch.json 當中
- 如果 VS Code 猜錯的話，你應該編輯主程式（待測試程式）的名稱。

例如以下我們設定 example/ex1.js 為主程式



然後設定好中斷點
(按下綠色箭號就可以開始除錯了)

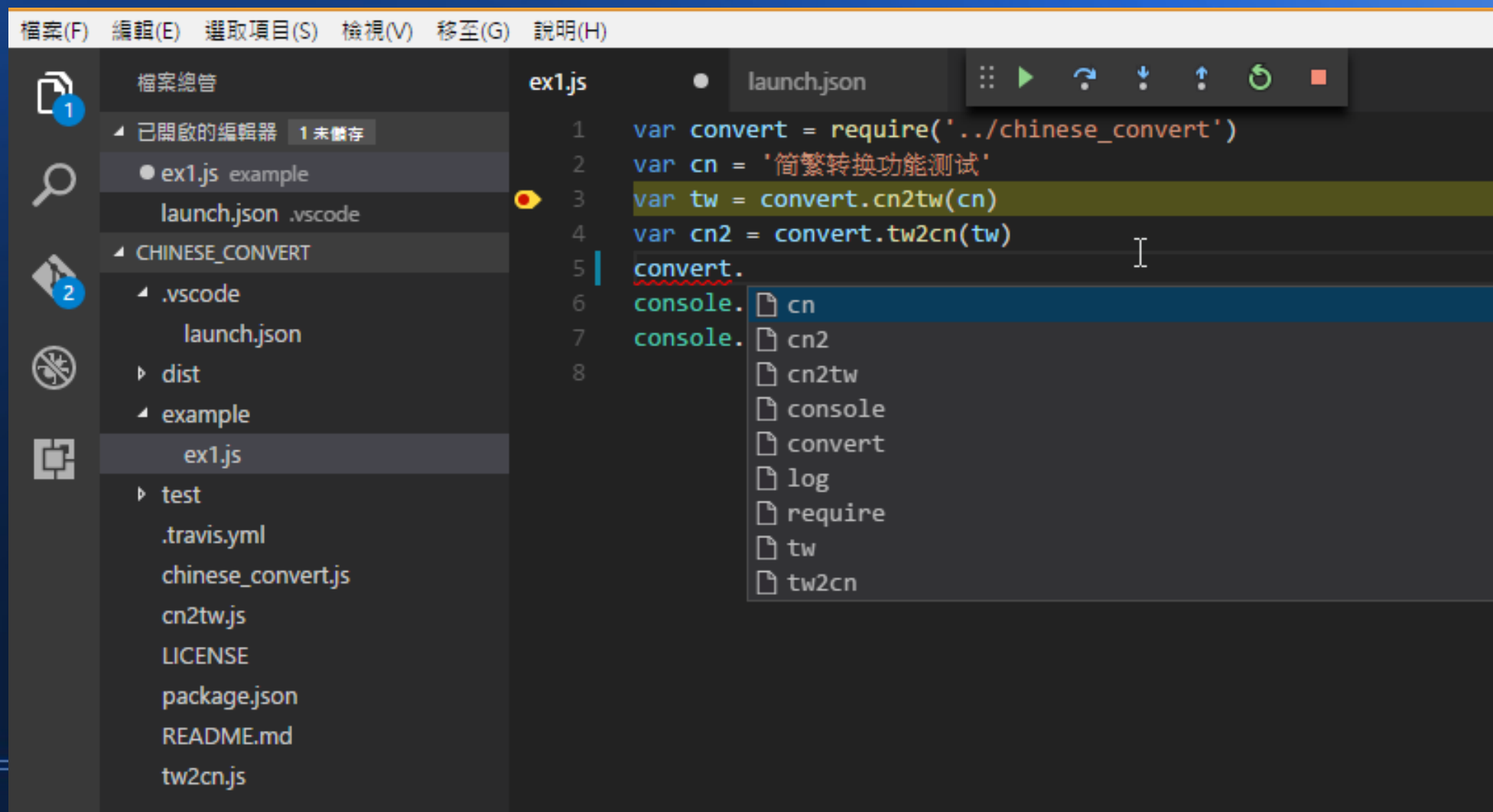


這種方式

- 有點像在使用 Visual Studio 的感覺
- 或許這也是為何叫做 Visual Studio Code 的原因

（這兩個根本就是不同的產品阿！）

更厲害的是 VS Code 連 Intellisense 這樣的成員提示功能都有



根本就是

- Visual Studio 的輕量版

現在

- 您應該已經瞭解，如何用 VS Code 當作 JavaScript 開發的 IDE 了

但是、這樣還是不夠厲害的

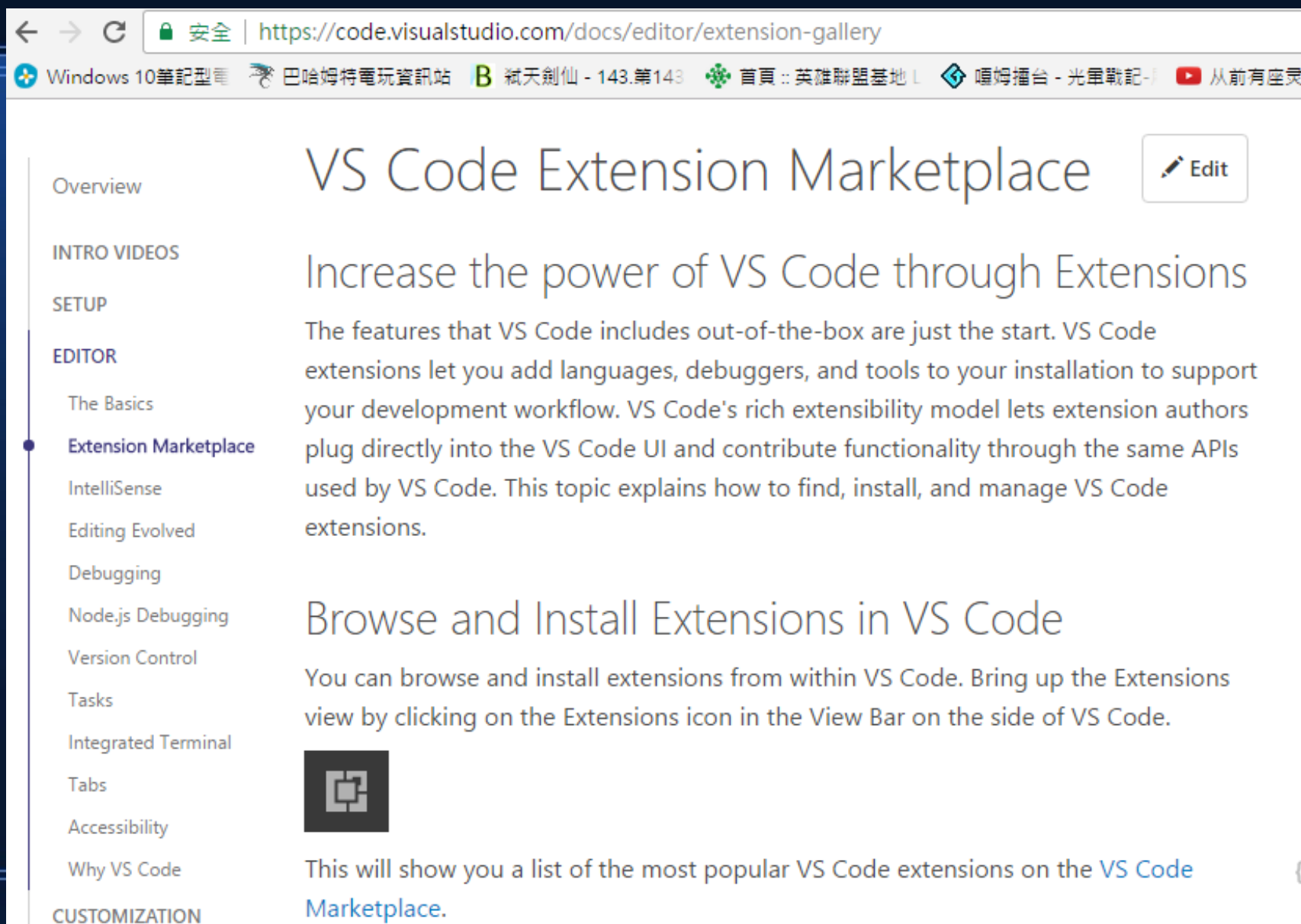
因為自從微軟被蘋果 K 了之後

- 終於開始理解了一件事...

只有產品好是不夠的

- 必須要注重社群生態 (Ecology)
- 所以要能讓大家能輕易地撰寫插件
Plugin

於是 VS Code 創造了 Marketplace



The screenshot shows the VS Code Extension Marketplace documentation page. The browser address bar displays the URL <https://code.visualstudio.com/docs/editor/extension-gallery>. The page features a left-hand navigation menu with the following items: Overview, INTRO VIDEOS, SETUP, EDITOR (highlighted with a blue dot), The Basics, Extension Marketplace (highlighted with a blue dot), IntelliSense, Editing Evolved, Debugging, Node.js Debugging, Version Control, Tasks, Integrated Terminal, Tabs, Accessibility, Why VS Code, and CUSTOMIZATION. The main content area is titled "VS Code Extension Marketplace" with an "Edit" button. Below the title, the text reads: "Increase the power of VS Code through Extensions". A paragraph follows: "The features that VS Code includes out-of-the-box are just the start. VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow. VS Code's rich extensibility model lets extension authors plug directly into the VS Code UI and contribute functionality through the same APIs used by VS Code. This topic explains how to find, install, and manage VS Code extensions." Below this, the section "Browse and Install Extensions in VS Code" is introduced with the text: "You can browse and install extensions from within VS Code. Bring up the Extensions view by clicking on the Extensions icon in the View Bar on the side of VS Code." This is accompanied by a small icon of a square with a smaller square inside. The text concludes: "This will show you a list of the most popular VS Code extensions on the [VS Code Marketplace](#)."

← → ↻ 安全 | <https://code.visualstudio.com/docs/editor/extension-gallery>

Windows 10筆記型電腦 巴哈姆特電玩資訊站 B 弑天劍仙 - 143.第143 首頁 :: 英雄聯盟基地 L 噶姆播台 - 光暈戰記- 从前有座灵

VS Code Extension Marketplace [Edit](#)

INTRO VIDEOS

SETUP

EDITOR

- The Basics
- Extension Marketplace**
- IntelliSense
- Editing Evolved
- Debugging
- Node.js Debugging
- Version Control
- Tasks
- Integrated Terminal
- Tabs
- Accessibility
- Why VS Code


CUSTOMIZATION

Increase the power of VS Code through Extensions

The features that VS Code includes out-of-the-box are just the start. VS Code extensions let you add languages, debuggers, and tools to your installation to support your development workflow. VS Code's rich extensibility model lets extension authors plug directly into the VS Code UI and contribute functionality through the same APIs used by VS Code. This topic explains how to find, install, and manage VS Code extensions.

Browse and Install Extensions in VS Code

You can browse and install extensions from within VS Code. Bring up the Extensions view by clicking on the Extensions icon in the View Bar on the side of VS Code.



This will show you a list of the most popular VS Code extensions on the [VS Code Marketplace](#).

目前我所使用的 VS Code 插件只有一個

那個插件稱為 StandardJS



寫過 JavaScript 的人都知道

- JavaScript 最大的缺點，
就是寫法太多又太亂 ...

以下是各種允許的寫法

- 字串可用單引號或雙引號
- 行尾可以加分號，但也可以不加
- `===` 與 `==` 很像但又有點不同
- 函數可用 `function xxx()` 或 `xxx=function()` ...
- 物件導向的寫法更是不計其數 ...
- ES6 才有的 `yield/generator` 竟然又有了新的 `async/await`

所以很多公司

- 都為 JavaScript 雜亂的語法感到傷腦筋
- 因為不只太方便，而且是太隨便了 ...

所以比較在乎程式碼統一性的公司

- 會使用像 JSLint 或 JSHint 這樣的工具來統一限制語法



StandardJS

- 其實就是利用 JsHint 所建構出來的一套工具，讓你可以完全不用自己規定，而是採用完全標準的統一語法

當然付出的代價是

- 你的程式碼要受更多規範
- 原本沒有問題的程式碼，在檢查之後
會出現一堆語法不夠好的警告訊息

不過我覺得

- 習慣 StandardJS 的風格之後，
JavaScript 程式碼看起來好多了！

所以前天開始

- 我寫程式都會採用 StandardJS 檢查語法

以下是 StandardJS 的語法限制

① standardjs.com/readme-zhtw

語法規則

- 兩個空白 – 當作縮排
- 字串用單引號 – 除非要避免跳脫字元
- 沒有不必要的變數 – 這可以解決 *超多的 Bug* !
- 不要加分號 – 這真的很 OK，真的！
- 絕對不要用 (、[或 ` 當開頭
 - 這是不用分號 *唯一* 可能遇到的問題 – *那就自動幫你檢查吧！*
 - 更多解釋
- 關鍵字後加空白 `if (condition) { ... }`
- 函數名稱後加空白 `function name (arg) { ... }`
- 統一用 `===` 取代 `==` – 但是 `obj == null` 可以用來檢查 `null || undefined`。
- 一定要例外處理 node.js 中的 `err` 參數
- 一定要對瀏覽器中的全域變數加上 `window` 前綴 – 除了 `document` 和 `navigator` 可以不用
 - 避免使用那些命名得很爛的全域變數，像是 `open`、`length`、`event` 和 `name`。
- 還有 更多更多的好處 – 今天就來試試 *standard* 吧！

看看一些 用 JavaScript Standard Style 寫的範例 來了解更多，或查看其他 數以千計使用 standard 的專案。

您可以用下列指令安裝 StandardJS 的檢查工具

安裝

使用 JavaScript Standard Style 最簡單的方法就是安裝在全域下，變成一個 Node 指令列程式。在 Terminal 中執行以下指令來安裝：

```
$ npm install standard --global
```

或者，你也可以在單一專案下局部的安裝 standard：

```
$ npm install standard --save-dev
```

注意 為了執行前面的指令，請確保你已經安裝了 [Node.js](#) 和 [npm](#)。

然後用下列指令檢查語法

用法

在你安裝 standard 之後，你就可以使用 standard 這支程式了。最簡單的用法就是在當前目錄下檢查所有 JavaScript 檔案的樣式：

```
$ standard
Error: Use JavaScript Standard Style
  lib/torrent.js:950:11: Expected '===' and instead saw '=='.
```

你也可以選擇性的檢查部分目錄們（請確保路徑前後有引號，避免出錯）。

```
$ standard "src/util/**/*.js" "test/**/*.js"
```

注意：standard 預設會檢查所有符合名為 `**/*.js` 和 `**/*.jsx` 的檔案。

但這些都還不夠方便

- 我們需要在打程式的時候
立刻看到語法是否有錯 ...

好消息是

StandardJS

提供了各種編輯器

的插件

<http://standardjs.com/readme-zhtw>

有文字編輯器的插件嗎？

首先，安裝 `standard`。接下來，就可以依據你使用的編輯器安裝對應的插件了：

Sublime Text

使用 [Package Control](#) 安裝 [SublimeLinter](#) 和 [SublimeLinter-contrib-standard](#)。

如果想要在儲存時自動修改樣式，可以安裝 [StandardFormat](#)。

Atom

安裝 [linter-js-standard](#)。

如果想要在儲存時自動修改樣式，可以安裝 [standard-formatter](#)。或是安裝 [standardjs-snippets](#) 可以使用自動補完。

Visual Studio Code

安裝 [vscode-standardjs](#)。(包含自動修改樣式的支援。)

需要 JS 自動補完，可以安裝：[vscode-standardjs-snippets](#)。需要 React 自動補完，可以安裝：[vscode-react-standard](#)。

Vim

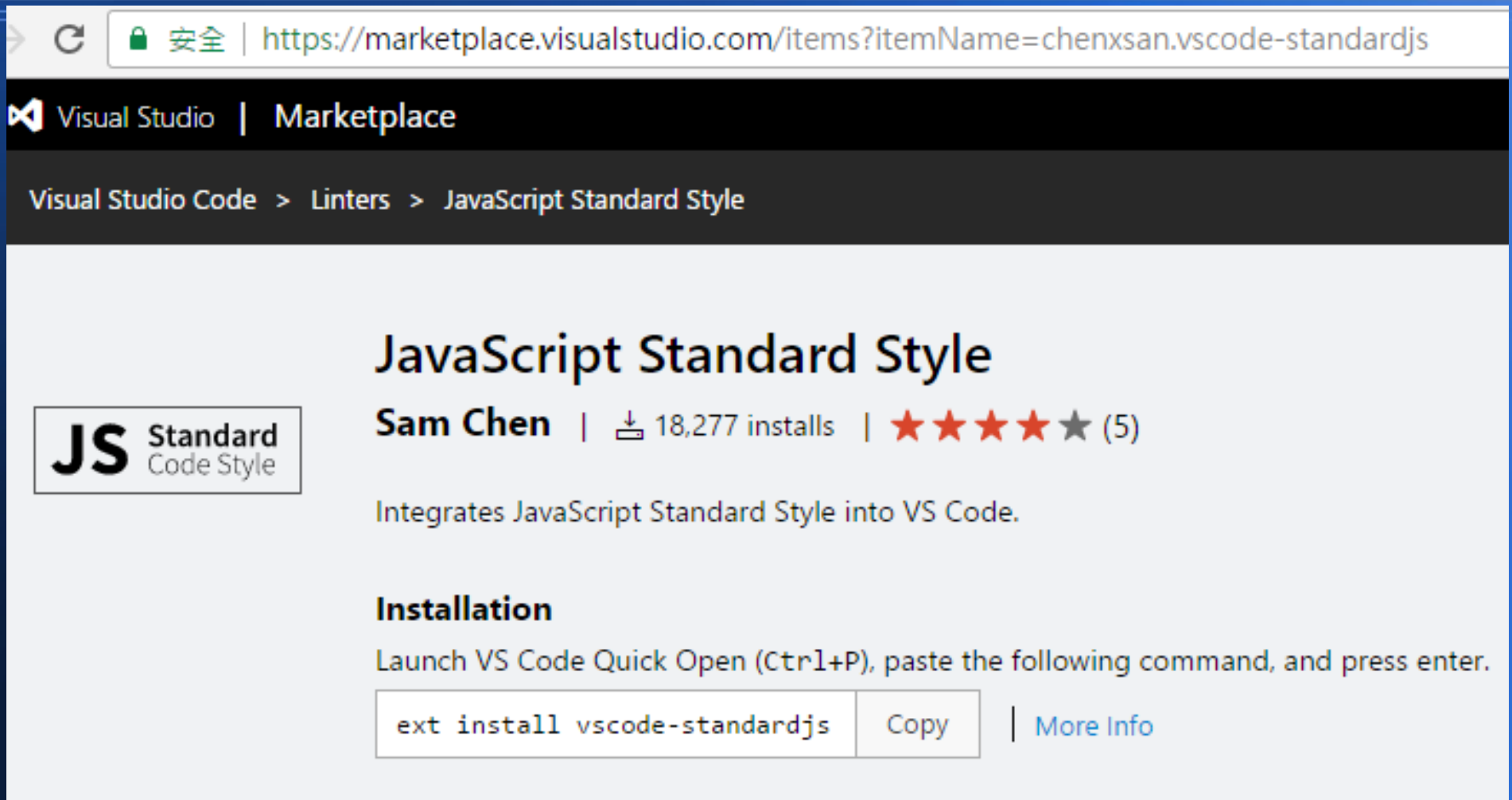
安裝 [Syntastic](#)，然後把以下加到 `.vimrc` 中：

```
let g:syntastic_javascript_checkers = ['standard']
```

如果想要在儲存時自動修改樣式，加入以下到 `.vimrc` 中：

```
autocmd bufwritepost *.js silent !standard --fix %  
set autoread
```

我們只要按照指示 在 VS Code 裡安裝插件就行了



The screenshot shows the Visual Studio Marketplace page for the 'JavaScript Standard Style' extension. The browser address bar shows the URL: <https://marketplace.visualstudio.com/items?itemName=chenxsan.vscode-standardjs>. The page header includes 'Visual Studio | Marketplace' and a breadcrumb trail: 'Visual Studio Code > Linters > JavaScript Standard Style'. The extension's title is 'JavaScript Standard Style' by 'Sam Chen'. It has 18,277 installs and a 4.5-star rating (5 reviews). The description states: 'Integrates JavaScript Standard Style into VS Code.' Under the 'Installation' section, it instructs users to launch VS Code Quick Open (Ctrl+P), paste the command 'ext install vscode-standardjs', and press enter. There are buttons for 'Copy' and 'More Info'.

Visual Studio Code > Linters > JavaScript Standard Style

JavaScript Standard Style

Sam Chen | 18,277 installs | ★★★★★ (5)

Integrates JavaScript Standard Style into VS Code.

Installation

Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter.

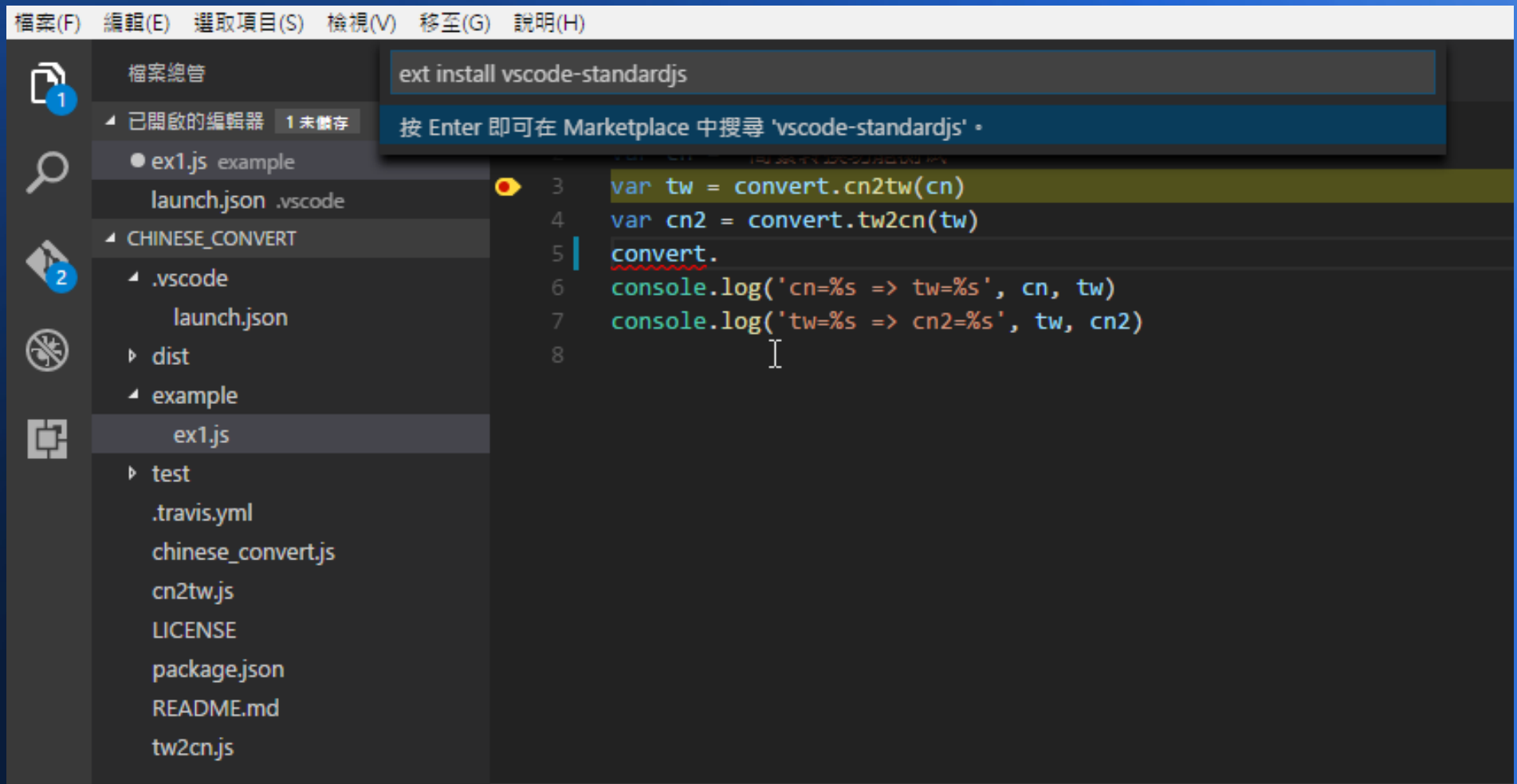
```
ext install vscode-standardjs
```

[Copy](#) | [More Info](#)

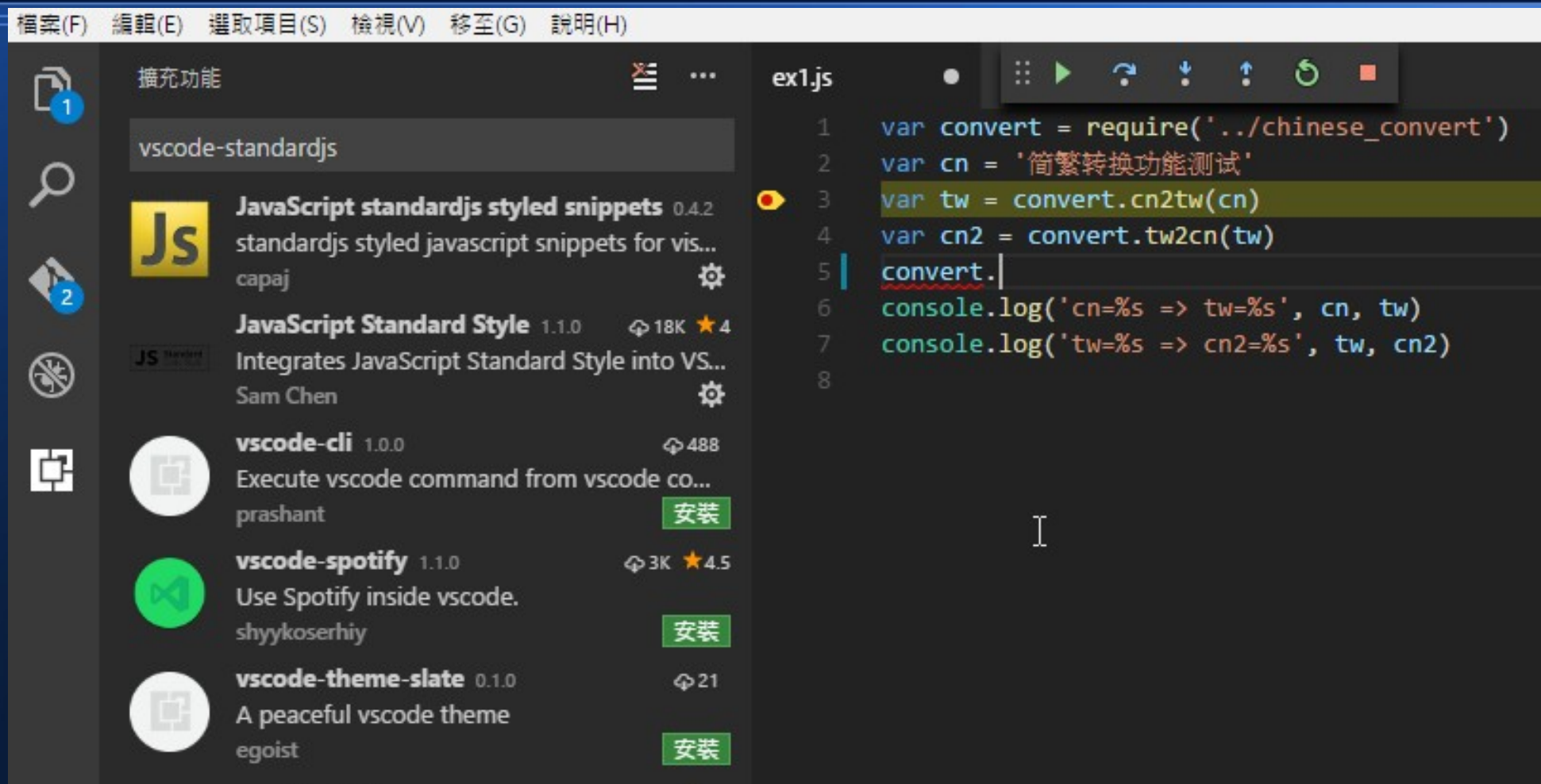
<https://marketplace.visualstudio.com/items?itemName=chenxsan.vscode-standardjs>

您只要在 VS Code 裏按下 Ctrl-P

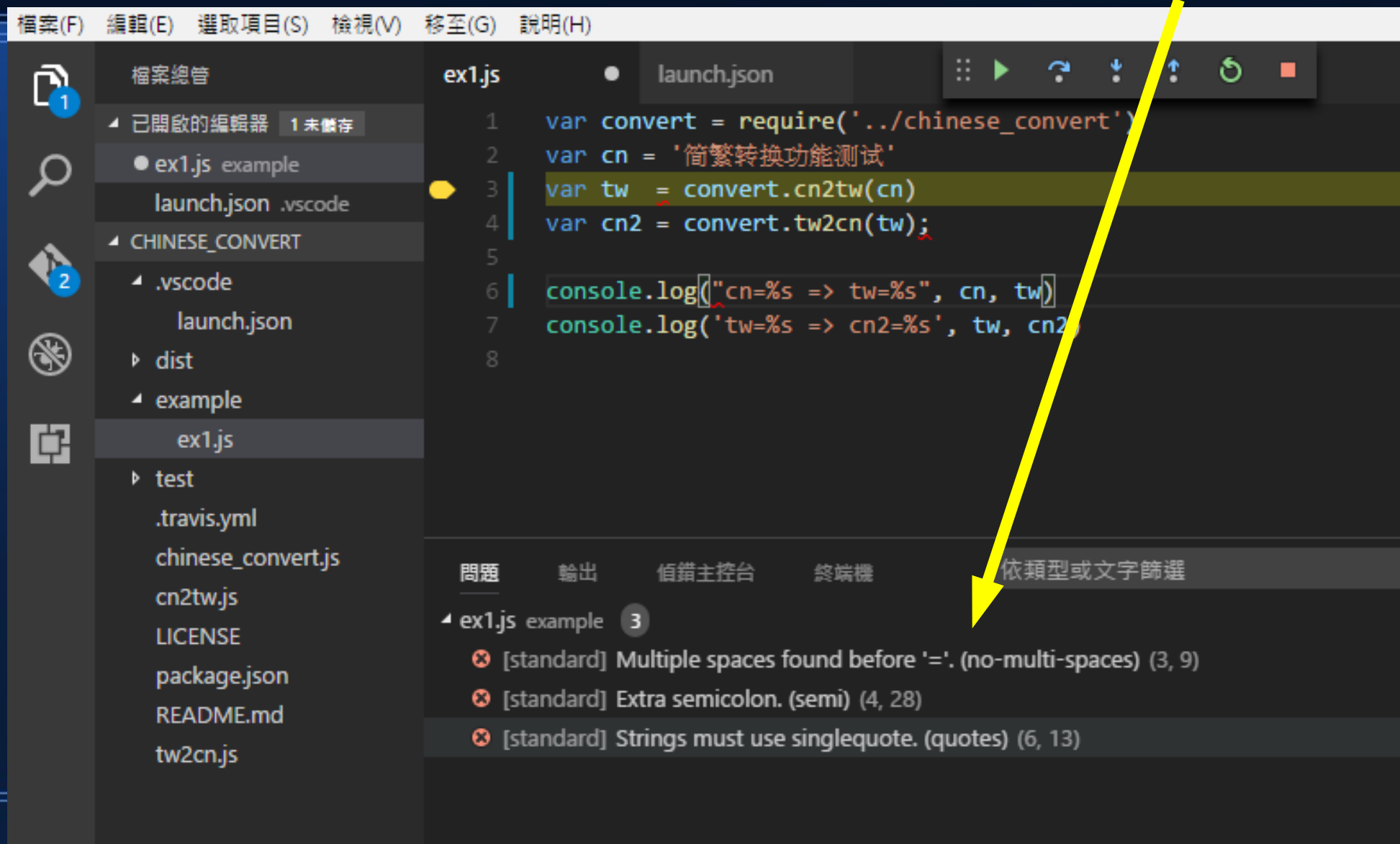
然後在輸入框中貼上 `ext install vscode-standardjs`



就可以找到 JavaScript Standard Style 這個插件，按下安裝就行了

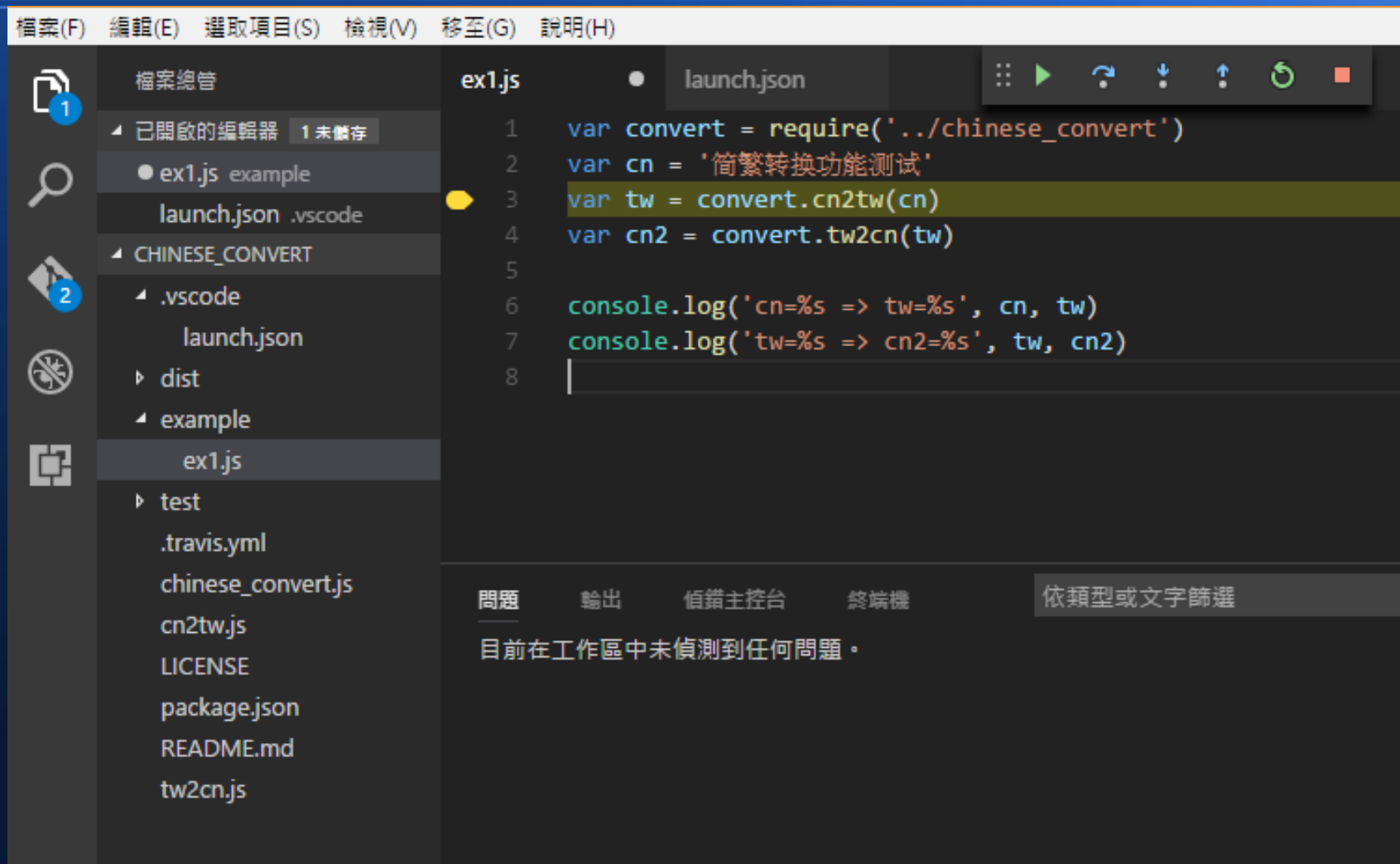


安裝完畢後，切到《問題》視窗
就可以看到程式碼是否有語法錯誤



只要把所有的語法錯誤都修掉

- 您的程式就符合了 StandardJS 的規格



這樣

- JavaScript 的程式，就有了一套標準語法可循
- 比較不會那麼混亂了！

但是、語法統一了

- 不代表程式就沒有問題！

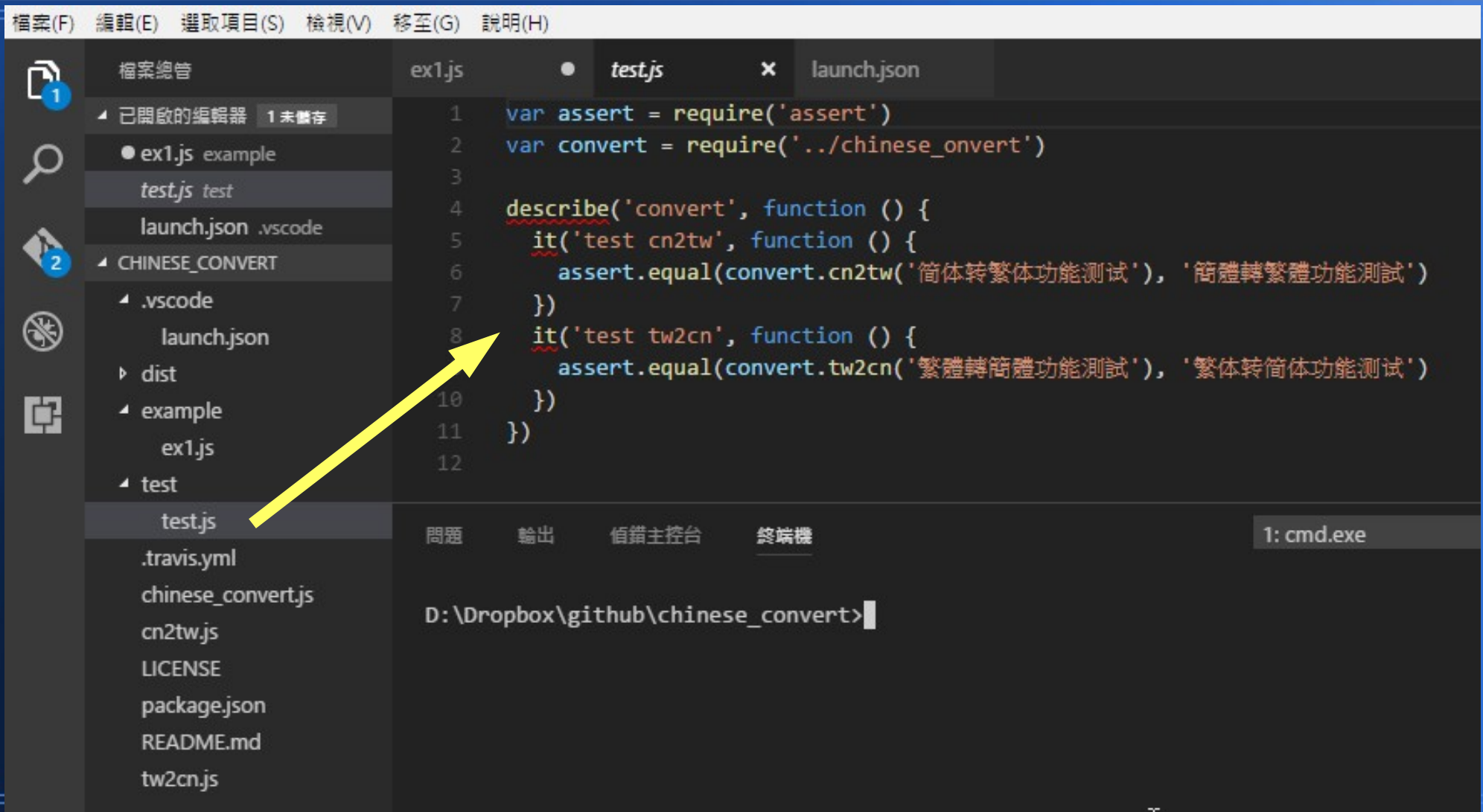
所以我們需要寫測試程式

- 測試程式是否有錯誤！

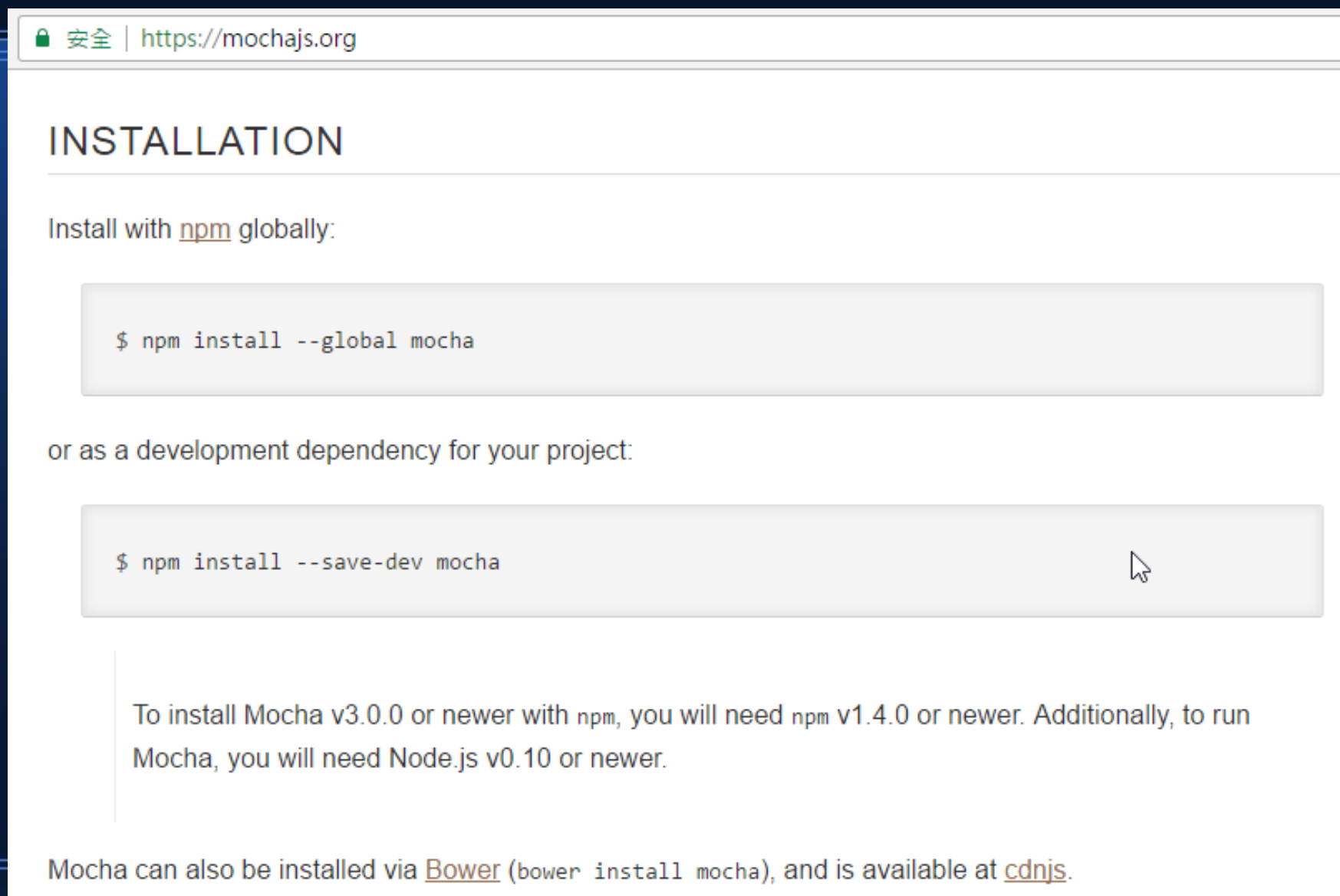
在 node. js 裡面

- Mocha 是一套最常用的測試框架

我們只要在 test 資料夾中 寫好測試程式



然後安裝好 mocha 測試框架



The screenshot shows the Mocha.js website with the URL <https://mochajs.org> in the browser's address bar. The page has a white background with a dark blue header. The main heading is "INSTALLATION". Below it, the text "Install with [npm](#) globally:" is followed by a code block containing the command `$ npm install --global mocha`. Below this, the text "or as a development dependency for your project:" is followed by another code block containing the command `$ npm install --save-dev mocha`. A mouse cursor is visible over the second code block. At the bottom, a note states: "To install Mocha v3.0.0 or newer with `npm`, you will need `npm` v1.4.0 or newer. Additionally, to run Mocha, you will need Node.js v0.10 or newer." The footer text says: "Mocha can also be installed via [Bower](#) (`bower install mocha`), and is available at [cdnjs](#)."

安全 | <https://mochajs.org>

INSTALLATION

Install with [npm](#) globally:

```
$ npm install --global mocha
```

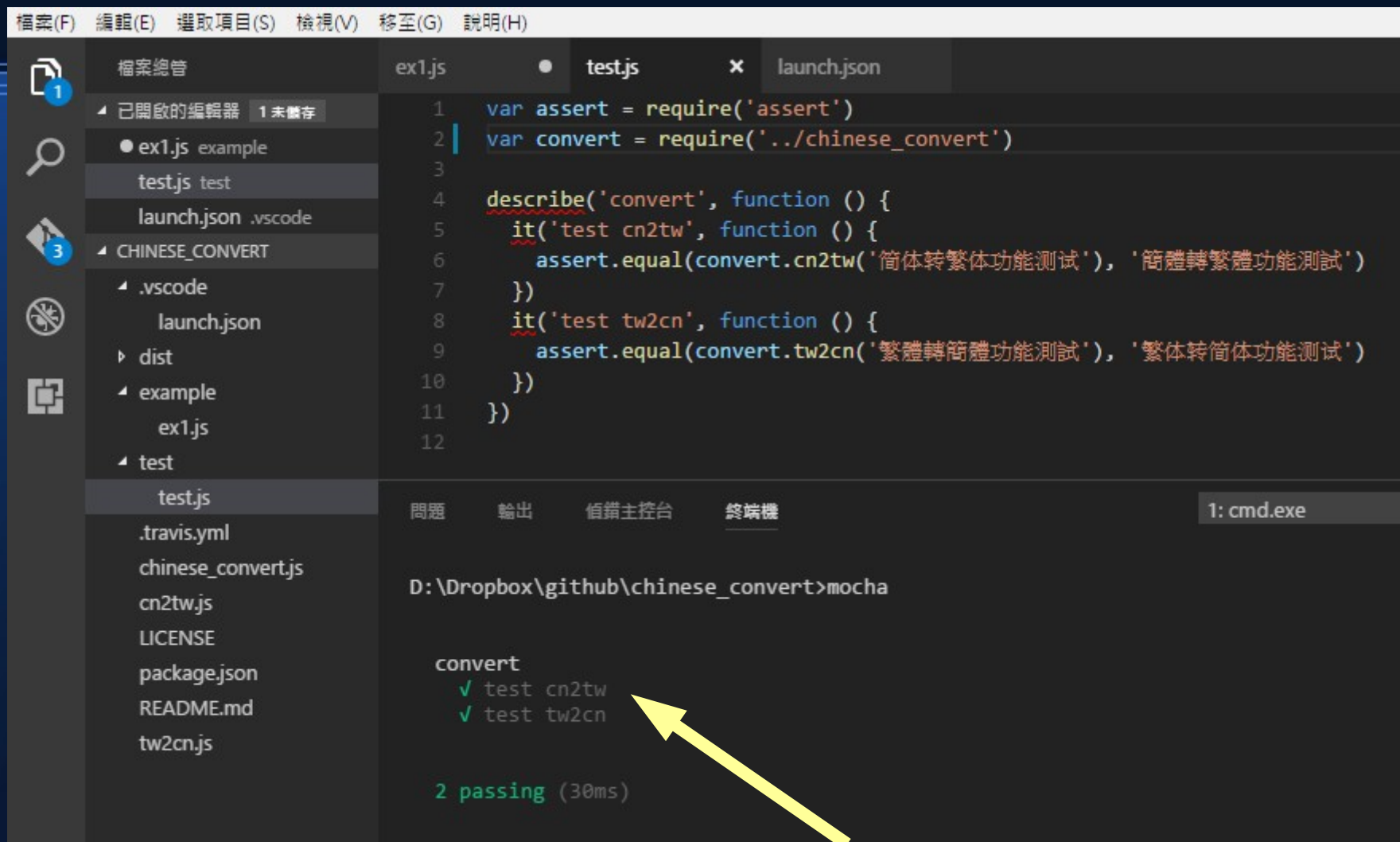
or as a development dependency for your project:

```
$ npm install --save-dev mocha
```

To install Mocha v3.0.0 or newer with `npm`, you will need `npm` v1.4.0 or newer. Additionally, to run Mocha, you will need Node.js v0.10 or newer.

Mocha can also be installed via [Bower](#) (`bower install mocha`), and is available at [cdnjs](#).

接著打上 mocha 指令



The screenshot shows the Visual Studio Code interface. The left sidebar displays the file explorer with a project structure including 'example', 'test', and various configuration files. The main editor area shows the 'test.js' file with Mocha test code. The bottom panel shows the 'Terminal' tab with the command 'mocha' executed in the directory 'D:\Dropbox\github\chinese_convert'. The output shows two passing tests: 'test cn2tw' and 'test tw2cn', with a total of '2 passing (30ms)'. A yellow arrow points to the test results in the terminal output.

```
var assert = require('assert')
var convert = require('../chinese_convert')

describe('convert', function () {
  it('test cn2tw', function () {
    assert.equal(convert.cn2tw('简体转繁体功能测试'), '簡體轉繁體功能測試')
  })
  it('test tw2cn', function () {
    assert.equal(convert.tw2cn('繁體轉簡體功能測試'), '繁体转简体功能测试')
  })
})
```

問題 輸出 偵錯主控台 終端機 1: cmd.exe

D:\Dropbox\github\chinese_convert>mocha

convert

- ✓ test cn2tw
- ✓ test tw2cn

2 passing (30ms)

就可以看到測試結果了！

在測試程式裏

- describe 是測試群， it 則是單一測試，
assert.equal 這類的函數可用來檢定結果正確與否

```
1  var assert = require('assert')
2  var convert = require('../chinese_convert')
3
4  describe('convert', function () {
5    it('test cn2tw', function () {
6      assert.equal(convert.cn2tw('简体转繁体功能测试'), '簡體轉繁體功能測試')
7    })
8    it('test tw2cn', function () {
9      assert.equal(convert.tw2cn('繁體轉簡體功能測試'), '繁体转简体功能测试')
10   })
11 })
12
```

但是

- Node.js 官方只支援簡易的 assert 判斷句
- 這種語法是給工程師看的，稱為 TDD (Test Driven Development) 語法

如果你直接接觸客戶

- 是從需求分析開始作的程式人
- 那麼客戶可能會看不懂你的測試案例
- 這將不利於和客戶溝通 ...

此時您可能就會想用客戶容易看懂的 BDD 語法

- 像是 should 或 expect 就是這類 BDD 語法

Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.lengthOf(3);
tea.should.have.property('flavors')
  .with.lengthOf(3);
```

[Visit Should Guide](#) ➔

Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.lengthOf(3);
expect(tea).to.have.property('flavors')
  .with.lengthOf(3);
```

[Visit Expect Guide](#) ➔

有個 **chai** 套件能支援 TDD/BDD 的三類語法



The screenshot shows the homepage of the Chai Assertion Library. At the top, there's a navigation bar with the Chai logo, the text "Chai Assertion Library", and links for "Guide", "API", and "Plugins". Below the navigation bar, a paragraph explains that Chai offers different interfaces for developers to choose from, highlighting BDD styles for expressive language and TDD for a classical feel. The main content area is divided into three columns, each representing a different assertion style: "Should", "Expect", and "Assert". Each column contains sample code snippets and a link to visit the corresponding guide.

chaijs.com

 Chai Assertion Library

[Guide](#) [API](#) [Plugins](#)

Chai has several interfaces that allow the developer to choose the most comfortable. The chain-capable BDD styles provide an expressive language & readable style, while the TDD assert style provides a more classical feel.

Should

```
chai.should();  
  
foo.should.be.a('string');  
foo.should.equal('bar');  
foo.should.have.lengthOf(3);  
tea.should.have.property('flavors')  
  .with.lengthOf(3);
```

[Visit Should Guide](#) ➔

Expect

```
var expect = chai.expect;  
  
expect(foo).to.be.a('string');  
expect(foo).to.equal('bar');  
expect(foo).to.have.lengthOf(3);  
expect(tea).to.have.property('flavors')  
  .with.lengthOf(3);
```

[Visit Expect Guide](#) ➔

Assert

```
var assert = chai.assert;  
  
assert.typeOf(foo, 'string');  
assert.equal(foo, 'bar');  
assert.lengthOf(foo, 3);  
assert.property(tea, 'flavors');  
assert.lengthOf(tea.flavors, 3);
```

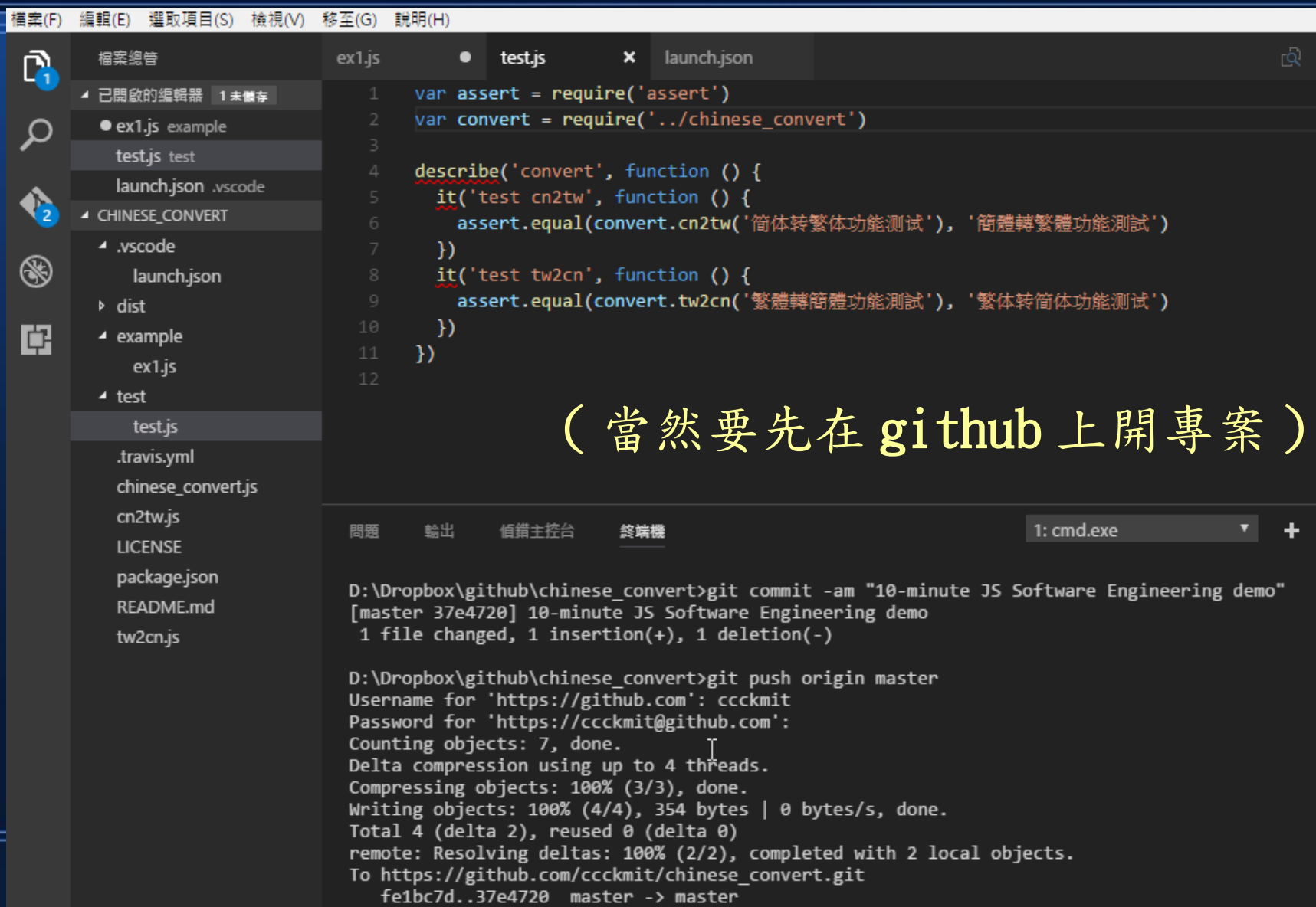
[Visit Assert Guide](#) ➔

您可以視需要

- 選擇到底要使用 TDD 還是 BDD 語法
- 到底要用 `assert/expect` 還是 `should`

只要用得順手就行了！

測試好之後，您就可以 用 git 指令上傳到 github 上發布



The screenshot displays the Visual Studio Code interface. The left sidebar shows the Explorer view with a project structure including files like `ex1.js`, `test.js`, `launch.json`, and a `CHINESE_CONVERT` directory. The main editor area shows the content of `test.js`, which includes imports for `assert` and `convert`, and two Mocha test cases: `it('test cn2tw', ...)` and `it('test tw2cn', ...)`. The bottom terminal panel shows the following commands and output:

```
D:\Dropbox\github\chinese_convert>git commit -am "10-minute JS Software Engineering demo"
[master 37e4720] 10-minute JS Software Engineering demo
1 file changed, 1 insertion(+), 1 deletion(-)

D:\Dropbox\github\chinese_convert>git push origin master
Username for 'https://github.com': ccckmit
Password for 'https://ccckmit@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 354 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/ccckmit/chinese_convert.git
fe1bc7d..37e4720 master -> master
```

(當然要先在 github 上開專案)

甚至使用 npm 發佈 node.js 套件

問題

輸出

偵錯主控台

終端機

```
D:\Dropbox\github\chinese_convert>npm version patch  
v1.0.3
```

```
D:\Dropbox\github\chinese_convert>npm publish ./  
+ chinese_convert@1.0.3
```

```
D:\Dropbox\github\chinese_convert>█
```

讓人家可以輕易地安裝



https://github.com/ccckmit/chinese_convert

這樣差不多就完成
套件開發與布署的任務了！

不過

- 我們還可以做得更好！

在今日的軟體工程上

- 通常會採用《持續整合》
(Continuous Integration) 的方式，
讓您的程式一上傳就能進行測試。

Travis-CI 是一個提供《持續整合》的網站

- 而且可以和 github 搭配

https://zh.wikipedia.org/wiki/Travis_CI

沒有登入 對話 貢獻 建立帳號 登入

條目 討論 台灣正體 閱讀 編輯 檢視歷史 搜尋維基百科

維基台北寫作聚於每月第二個禮拜六舉行，歡迎報名參與 [關閉]

Travis CI [編輯]

維基百科，自由的百科全書



本條目存在以下問題，請協助改善本條目或在討論頁針對議題發表看法。

- 本條目的語調或風格可能不適合百科全書的寫作方式。(2012年11月22日)
- 沒有或很少條目連入本條目。(2012年11月22日)

Travis CI是在軟體開發領域中的一個在線的，分布式的^[1]持續集成服務，用來構建及測試在GitHub^[2]託管的代碼。這個軟體的代碼同時也是開源的，可以在GitHub上下載到^[3]，儘管開發者當前並不推薦在開源項目中單獨使用它。^[4]

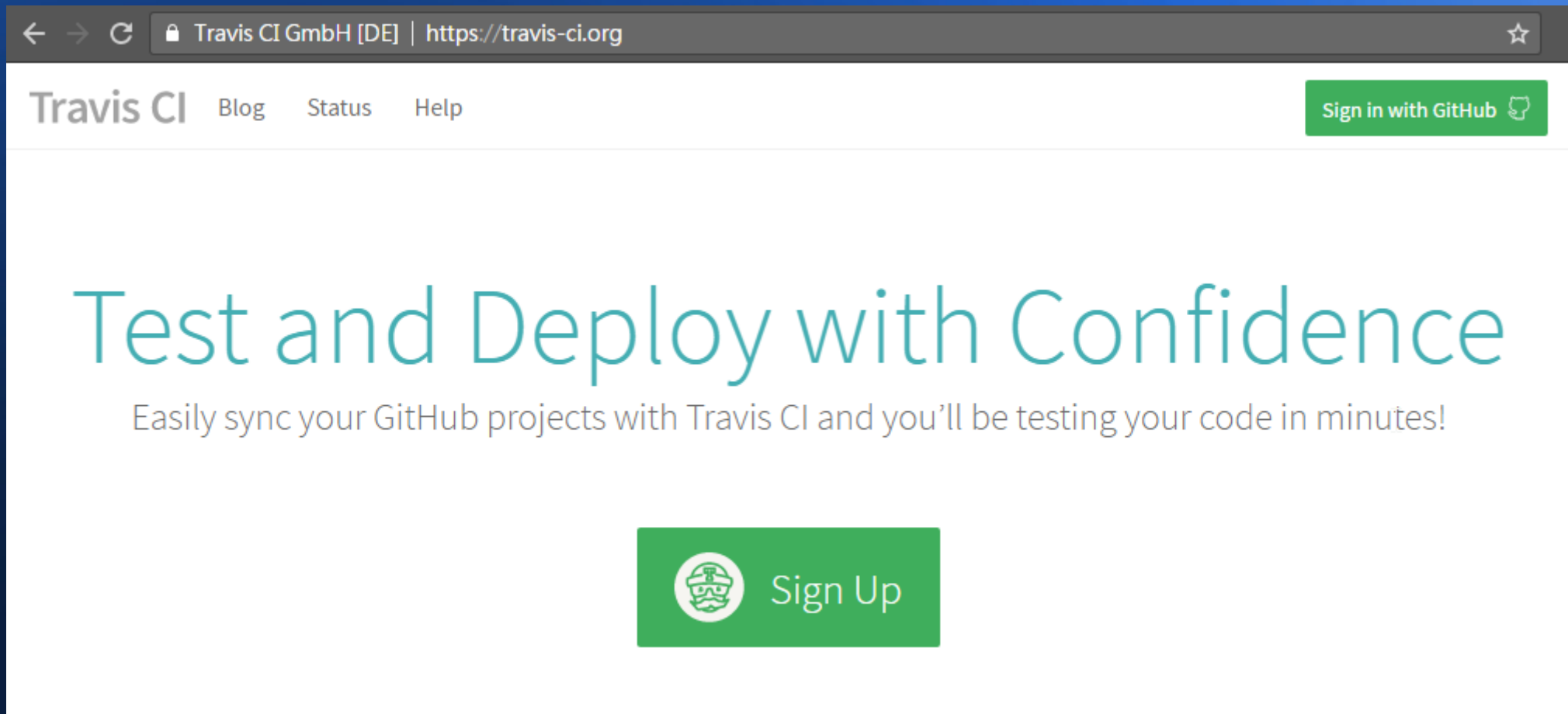
它提供了多種程式語言的支持，包括 Ruby，JavaScript，Java，Scala，PHP，Haskell和Erlang在內的多種語言。^[5]許多知名的開源項目使用它來在每次提交的時候進行構建測試，比如Ruby on Rails，Ruby和Node.js。^{[5][6]}

2012年，Travis CI 決定進行募資以支持後續的開發^[7]，在這次募資活動中，許多重量級的科技公司給予了資助。^[8]

Travis CI	
開發者	Travis CI 社區
程式語言	Ruby
系統平台	Web
類型	持續集成
網站	travis-ci.org

您只要用 `github` 的帳號註冊啟用

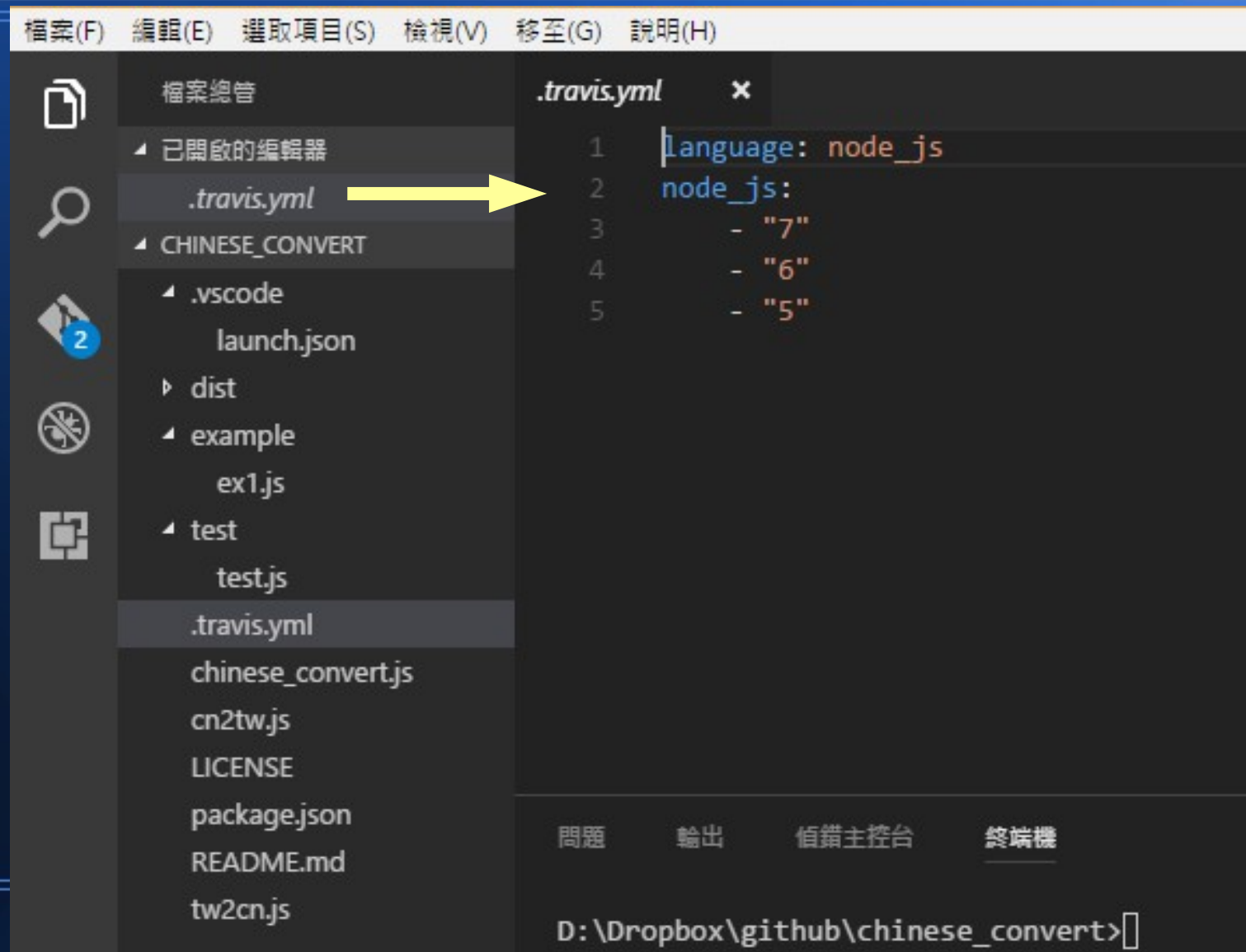
- 就可以從中選擇哪些專案要《持續整合測試》



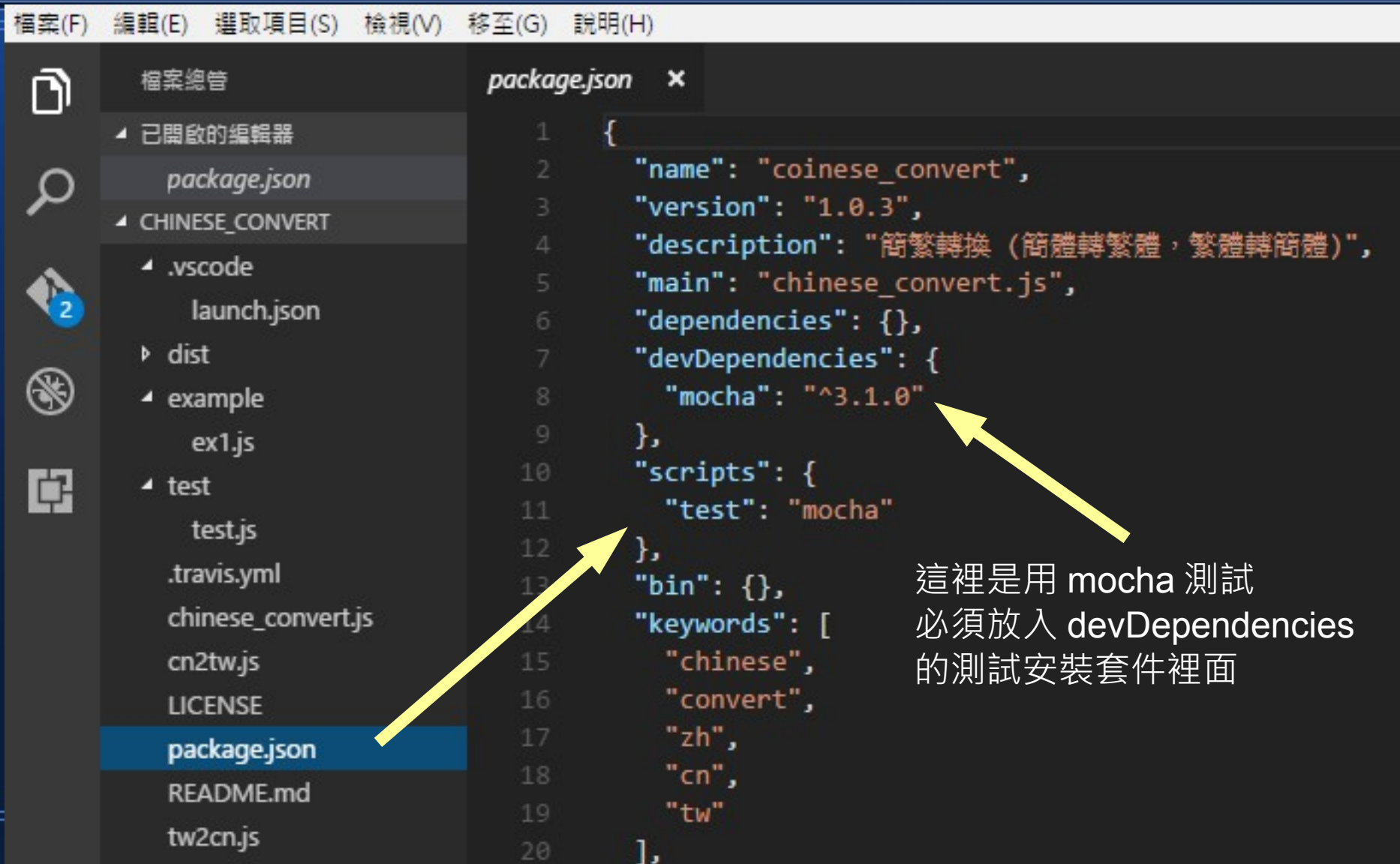
<https://travis-ci.org/>

然後你必須在專案裏 寫一個 `.travis.yml` 的檔案

- 說明
測試環境
與版本



並在 package.json 裏 描述 script/test 的測試方式



若還不清楚可以參考下列文章

- [Node.js] 用 mocha 做單元測試並整合 Travis-CI

<http://larry850806.github.io/2016/10/02/mocha-travis-ci/>

這樣

- 當你每次將專案推向 github 時，
travis-ci 就會自動幫你進行測試！

測試的結果，除了在 travis-ci 網站上可以看到之外

The screenshot displays the Travis CI web interface for the repository `ccckmit / chinese_convert`. The page shows a successful build status with a green checkmark and the text "build passing". The build is identified as "master 10-minute JS Software Engineering demo" and "#6 passed". Key details include the commit hash `37e4720`, the branch `master`, and the build duration of 59 seconds. The build was completed 24 minutes ago. Below the main build summary, a table lists the build jobs, showing three jobs (#6.1, #6.2, #6.3) all passing, with durations of 56, 34, and 40 seconds respectively.

Travis CI | Blog | Status | Help | 陳鍾誠

ccckmit / chinese_convert

build passing

Current | Branches | Build History | Pull Requests | More options

✓ **master** 10-minute JS Software Engineering demo

Commit 37e4720
Compare fe1bc7d..37e4720
Branch master

ccckmit authored and committed

🔄 #6 passed

⌚ Ran for 59 sec
⌚ Total time 2 min 10 sec
📅 24 minutes ago

🔄 Restart build

Build Jobs

✓ # 6.1	🐙 </> Node.js: 7	📦 no environment variables set	⌚ 56 sec	🔄
✓ # 6.2	🐙 </> Node.js: 6	📦 no environment variables set	⌚ 34 sec	🔄
✓ # 6.3	🐙 </> Node.js: 5	📦 no environment variables set	⌚ 40 sec	🔄

也可以在您的 e-mail 裏看到

Gmail ▾

←

📁

⚠

🗑

📁 ▾

🏷 ▾

更多 ▾

第 1 個，共 31,154

撰寫

收件匣 (226)

重要郵件

寄件備份

草稿 (6)

社交圈

活動

研討會

寫書

購買

Fixed: ccckmit/chinese_convert#6 (master - 37e4720) 📄 收件匣 x 🖨 🔍

 **Travis CI** <builds@travis-ci.org> 16:28 (26 分鐘前) ☆ ↩ ⌵

寄給我 ▾

🌐 英文 ▾ > 中文 (繁體) ▾ [翻譯郵件](#) [關閉下列語言的翻譯功能：英文 x](#)

 **ccckmit / chinese_convert (master)**

✓ **Build #6 was fixed.** 🕒 59 seconds

 ccckmit 37e4720 Changeset →

10-minute JS Software Engineering demo

Want to know about upcoming build environment updates?

Would you like to stay up-to-date with the upcoming Travis CI build environment updates? We set up a mailing list for you! Sign up [here](#).

現在

- 您應該已經大致瞭解 JavaScript 從
除錯、TDD/BDD 測試、發布到
持續整合的那些事情了！

剩下的

- 就是自己真正去寫個程式，
把上述的事情通通做一遍！

透過這樣的實戰訓練

相信您應該會有很多收穫才對！

這就是我最近幾年

- 透過 JavaScript/Node. js
對軟體工程的新認識！

也是我們今天的

- 十分鐘系列！

希望您會喜歡！

我們下回見！

Bye Bye!