



# Angular 自動化測試實戰

---

使用 Protractor 實現 E2E 測試



多奇數位創意有限公司

技術總監 黃保翕 ( Will 保哥 )

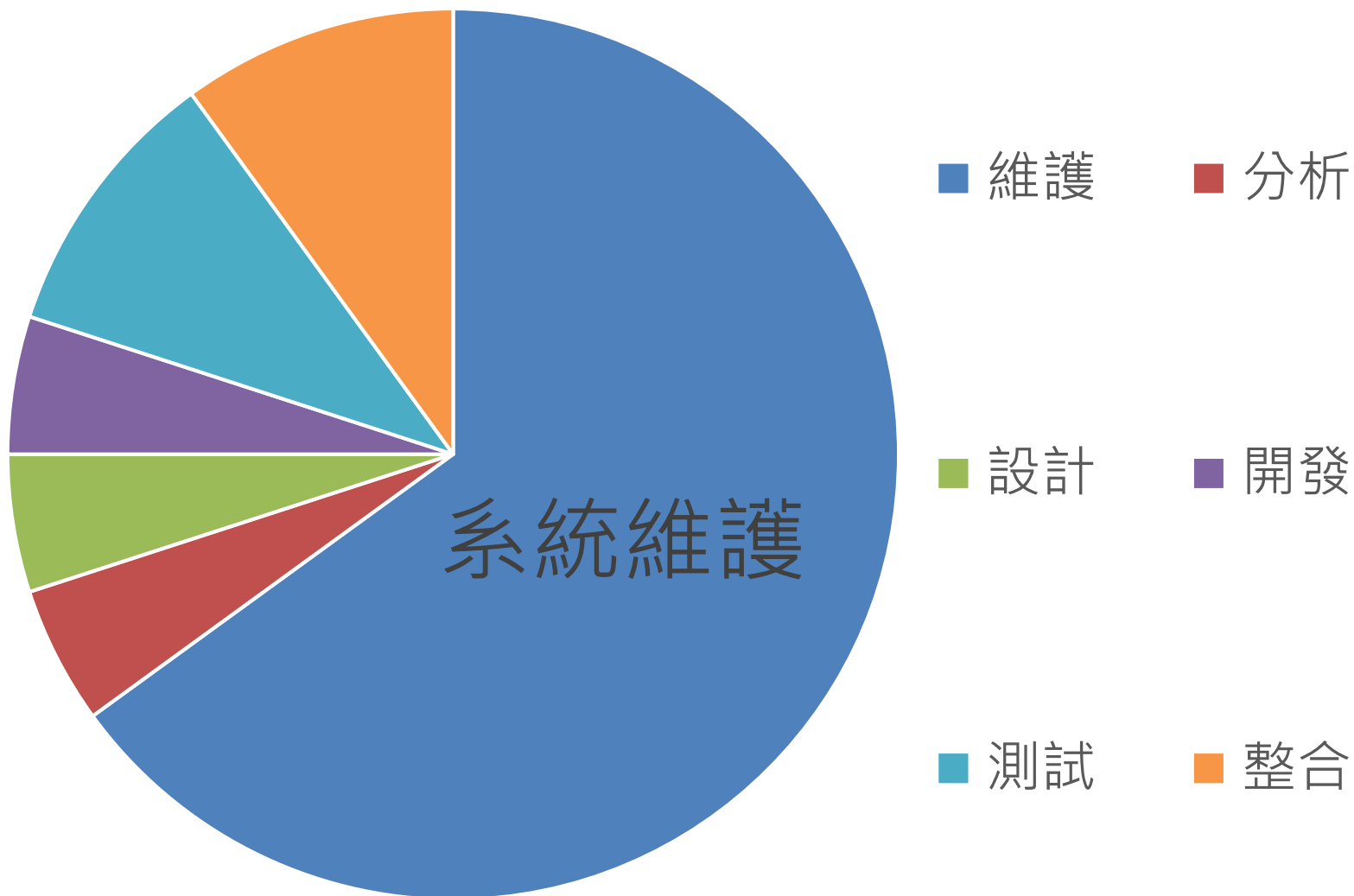
部落格：<http://blog.miniasp.com/>

一次搞懂單元測試、整合測試、端對端測試之間的差異

# 建立正確的測試觀念



# 軟體開發生命週期的占比



# 維護網站時可能會遇到的問題

- 維護程式碼的過程讓網站越來越容易出現問題
- 新增需求時會造成舊的功能壞掉的情況
- 上版後才發現不小心把其他地方改壞了
  - 被使用者發現 Bugs
- 修正 Bugs 的風險越來越高
  - 寧願把 Bug 當作 Feature 看待
- 負責維護網站的人通常不是建置該網站的人
  - 所有 Domain Knowledge 都在原始開發者身上
  - 原始開發者通常很理解需求，造成 Bugs 的機率較小
  - 接手的人無法短時間了解專案的全貌

# 自動測試 & 人工測試

自動測試	人工測試
不易犯錯	容易犯錯
花時間寫 <b>測試程式</b> 與維護	花時間寫 <b>測試腳本</b> 與維護
隨時隨地執行 <b>成本低廉</b>	每次測試都需花 <b>人力成本</b>
<b>更快速的找到問題</b>	較花時間找到問題
測試程式碼會與版控一起簽入	測試腳本通常不會與版控一起簽入

# 單元測試 & 整合測試

- 單元測試

- 物件之間沒有相依
- 測試粒度非常小，通常是一個方法或類別
- 每次執行結果都會一致

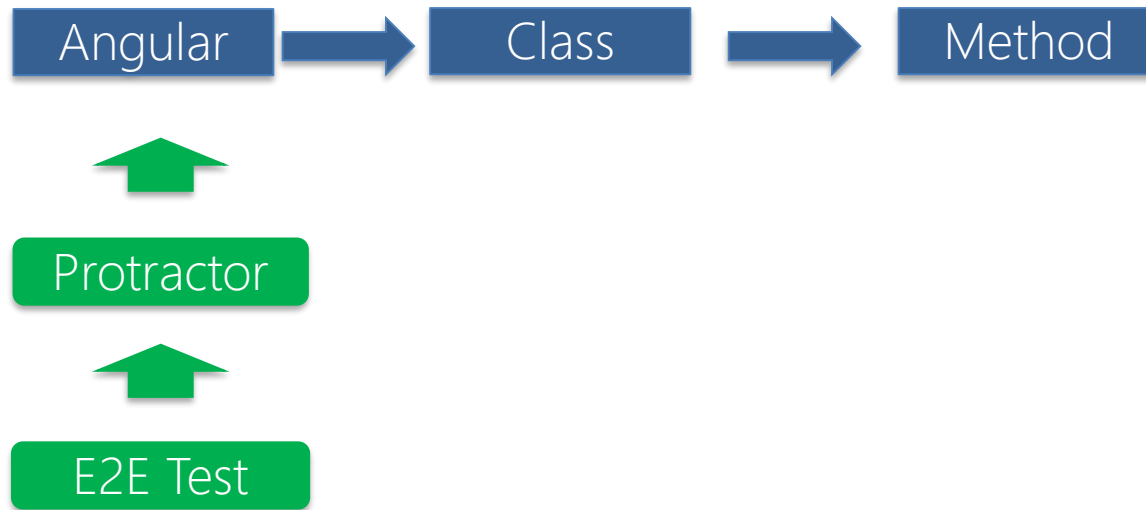


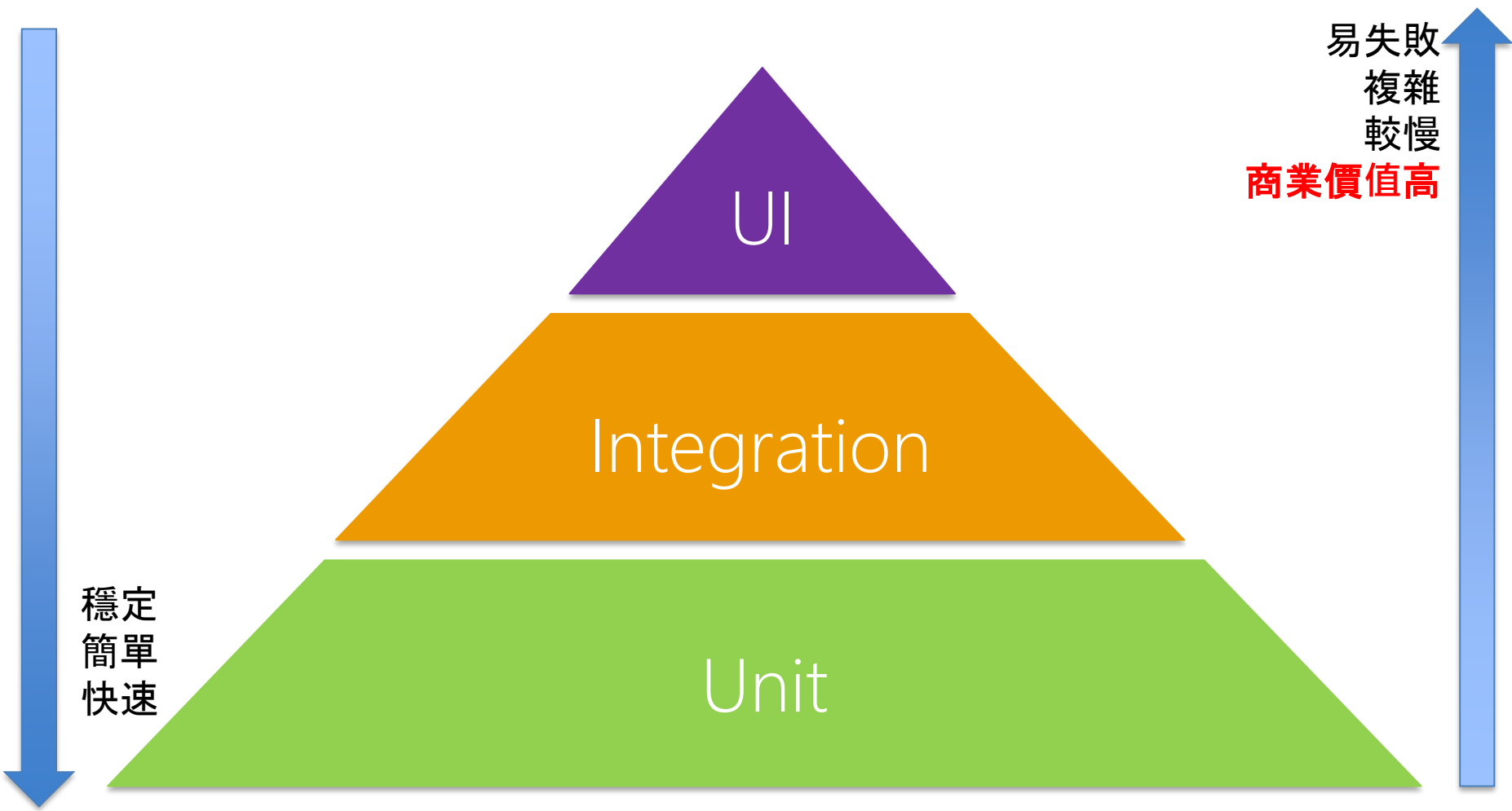
- 整合測試

- 不同模組之間的溝通測試，例如元件與元件之間的溝通

# E2E 測試 / UI 測試

- 模擬終端使用者的各種操作情境





Google Suggestions: 70%/20%/10%



Test Framework & Test Runner

# 認識 Jasmine 與 Karma



# 在 Angular 中執行測試

- 以 [Karma](#) 作為測試執行器 (Test Runner)
  - 允許使用不同的裝置 (如瀏覽器、行動裝置等) 執行測試
  - 支援多數常見的測試框架 (如 [Jasmine](#)、[Mocha](#) 等)
  - 提供各種不同的**測試報表範本**
  - 提供 CLI 介面，與 CI 輕鬆整合
- 以 [Jasmine](#) 作為測試框架 (Test Framework)
  - 行為驅動 (Behavior-Driven) 的測試框架
  - 包含大量**語意化的斷言 (assertion) API**
  - 能夠清楚描述測試行為，容易寫出**好閱讀的測試案例**
  - 具有模擬 (mock) 功能，隔離與外部類別的相依

# 使用 Jasmine 撰寫測試案例

用來描述一個測試情境

```
describe('Testing Calculator Service', () => {  
  it('should add two numbers', () => {
```

用來描述一個測試案例

(一組測試情境中可以包含多個測試案例)

```
    // Arrange  
    const num1 = 1;  
    const num2 = 2;  
    const expected = 3;  
    const service = new CalculatorService();  
  
    // Act  
    const actual = service.addTwoNumbers(num1, num2);  
  
    // Assert  
    expect(actual).toBe(expected);  
  });  
});
```

# 使用 Jasmine 撰寫測試案例

```
describe('Testing Calculator Service', () => {  
  it('should add two numbers', () => {
```

```
    // Arrange
```

```
    const num1 = 1;
```

```
    const num2 = 2;
```

```
    const expected = 3;
```

```
    const service = new CalculatorService();
```

準備執行測試單元  
所需的資料

```
    // Act
```

```
    const actual = service.addTwoNumbers(num1, num2);
```

```
    // Assert
```

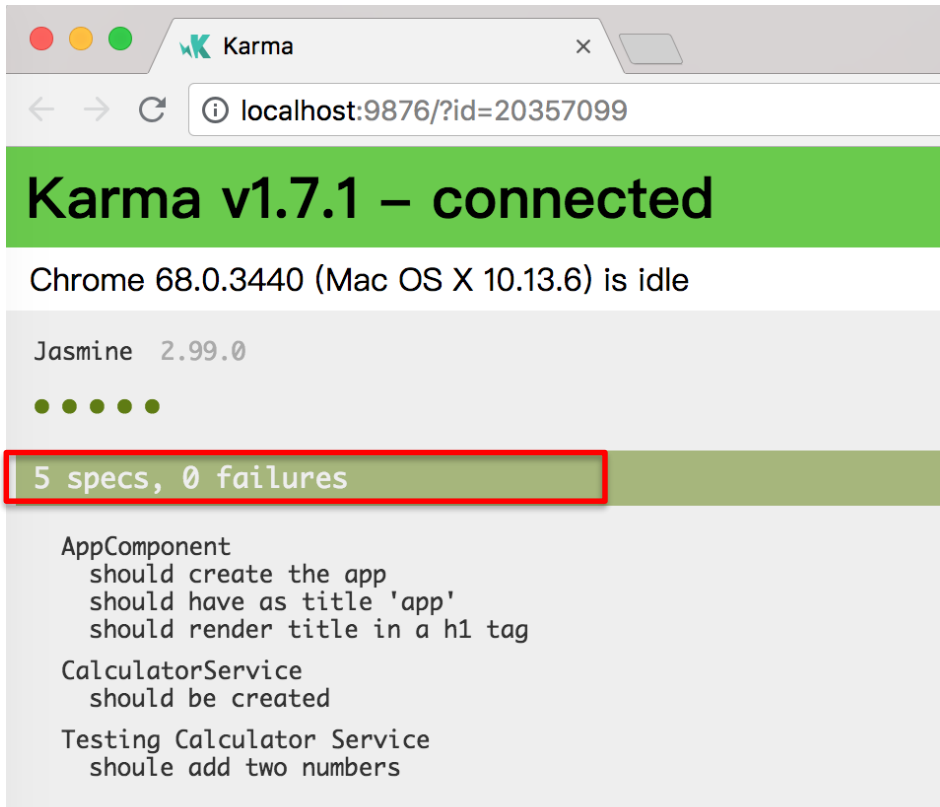
```
    expect(actual).toBe(expected);
```

實際執行測試單元

```
  });  
});
```

驗證執行結果是否符合預期

# 使用 **ng test** 指令執行測試



```
26 08 2018 17:28:23.054:WARN [karma]: No captured browser, open http://localhost:9876/
26 08 2018 17:28:23.062:INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876/
26 08 2018 17:28:23.062:INFO [launcher]: Launching browser Chrome with unlimited concurrency
26 08 2018 17:28:23.068:INFO [launcher]: Starting browser Chrome
26 08 2018 17:28:27.623:WARN [karma]: No captured browser, open http://localhost:9876/
26 08 2018 17:28:27.750:INFO [Chrome 68.0.3440 (Mac OS X 10.13.6)]: Connected on socket 1234567890
Chrome 68.0.3440 (Mac OS X 10.13.6): Executed 5 of 5 SUCCESS (0.16 secs / 0.139 secs)
```

# 當測試發生錯誤時的測試結果

Chrome 68.0.3440 (Mac OS X 10.13.6) Testing Calculator Service shoule add two numbers FAILED

Expected 3 to be 5.

at UserContext.<anonymous> src/app/calculator.service.spec.ts:32:20  
at ZoneDelegate.push../node\_modules/zone.js/dist/zone.js.ZoneDelegate.invoke node\_  
at ProxyZoneSpec.push../node\_modules/zone.js/dist/zone-testing.js.ProxyZoneSpec.or  
at ZoneDelegate.push../node\_modules/zone.js/dist/zone.js.ZoneDelegate.invoke node\_

Chrome 68.0.3440 (Mac OS X 10.13.6): Executed 5 of 5 (1 FAILED) (0 secs / 0.203 secs)

**Karma v1.7.1 – connected**

Chrome 68.0.3440 (Mac OS X 10.13.6) is idle

Jasmine 2.99.0

● ● ● ● X

**5 specs, 1 failure**

Spec List | Failures

**Testing Calculator Service shoule add two numbers**

Expected 3 to be 5.

# 測試案例執行順序

```
describe('測試案例群組', () => {  
  beforeAll(() => { /* 在整個 describe() 一開始執行 */ });  
  
  beforeEach(() => { /* 在 describe() 內的每個 it() 前執行 */ });  
  
  // 舊版 Jasmine 2.x 預設 it 會按照順序執行 (Angular CLI 預設用 2.x 版)  
  // 新版 Jasmine 3.x 預設 it 並不會依序執行!  
  it('測試案例內容 1', () => {  
    expect(true).toBe(true);  
  });  
  
  it('測試案例內容 2', () => {  
    expect(true).toBe(true);  
  });  
  
  afterEach(() => { /* 在 describe() 內的每個 it() 結束後執行 */ });  
  
  afterAll(() => { /* 在整個 describe() 將要結束前執行 */ });  
});
```

# 指定執行或不執行測試案例及群組

- **fdescribe()**：所有測試中僅執行這個測試群組
  - 這裡的 **f** 代表著 **focus** (專注) 的意思！
  - 整份測試計畫中有使用 **fdescribe()** 時，只有 **fdescribe()** 內的測試案例會被執行，任何其他 **describe()** 群組內的測試案例不會被執行。
  - **fdescribe()** 允許有兩個以上。
- **xdescribe()**：所有測試中不執行這個測試群組
  - 這裡的 **x** 代表著 **exclude** (排除) 的意思！
  - 暫時停止特定測試群組執行
  - **xdescribe()** 允許有兩個以上。
- **fit()**：在特定測試群組中僅執行這個測試案例
- **xit()**：在特定測試群組中不執行這個測試案例



# 認識 Matchers API

- 所謂 Matchers 的用途
  - 撰寫測試必須經常 **斷言** (Assert) 執行結果是否符合預期
  - Matchers 用來匹配「實際結果」與「預期結果」之用
  - 常見的語句：I expect YOU **to be** SOMETHING.

`expect(actual).toBe(expected);`

  
真實的你      Matcher 可用的 API      父母期待的你

- 執行 `expect(actual)` 會得到一個 `Matchers<T>` 物件
- 所有 Matchers 物件提供的斷言方法如下：
  - <https://jasmine.github.io/api/edge/matchers.html>
- 若要自訂 Matchers 斷言方法，可以參考[這篇文章](#)。

# Matchers 常用的斷言方法 (1)

- `expect(actual).toBe(realThing)`
- `expect(actual).not.toBe(true)`
- `expect(actual).toBeDefined()`
- `expect(actual).toBeFalsy()`
- `expect(actual).toBeTruthy()`
- `expect(actual).toBeNull()`
- `expect(actual).toBeUndefined()`
- `expect(array).toContain(element)`
- `expect(string).toContain(substring)`
- `expect(string).toMatch(/string$/)`
- `expect(object).toEqual({"foo": ['bar', 'baz']})`

# Matchers 常用的斷言方法 (2)

- `expect(number).toBeNaN()`
- `expect(number).toBeCloseTo(42.12345, 3)` [[原始碼](#)]
  - 取小數點三位 (0.001) 的一半為最大可容許誤差！
  - `Math.round(兩數差 * (10**(3+1))) / (10**(3+1)) <= (10**-3)/2`
- `expect(number).toBeGreaterThan(3)`
- `expect(number).toBeGreaterThanOrEqual(25)`
- `expect(number).toBeLessThan(0)`
- `expect(number).toBeLessThanOrEqual(123)`
- `expect(number).toBeNegativeInfinity()`
- `expect(number).toBePositiveInfinity()`
- `expect().nothing();`

# 外掛其他 Matchers 套件

- karma-jasmine-matchers 擴充許多語意 Matchers
  - <https://github.com/JamieMason/karma-jasmine-matchers>
- 擴充 Matchers 部分範例

```
expect(array).toBeArray();  
expect(array).toBeArrayOfBooleans();  
expect(array).toBeArrayOfNumbers();  
expect(array).toBeArrayOfObjects();  
expect(array).toBeArrayOfSize(number);
```

# 實戰演練：使用 Jasmine 撰寫單元測試

- 服務元件：`"jasmine-intro.service.ts"`
- 單元測試：`"jasmine-intro.service.spec.ts"`
- 練習目標
  - 編輯 **JasmineIntroService** 服務元件
    - 新增**乘法與除法**的 Methods
  - 請在單元測試的 `.spec.ts` 檔案中**新增**兩個測試案例
    - 新增**乘法與除法** Methods 的單元測試程式碼
- 執行測試 (預設會進入監視模式)
  - `npm test`
  - `Ctrl+R T` (VSCode keyboard shortcuts)

Protractor Quick Started

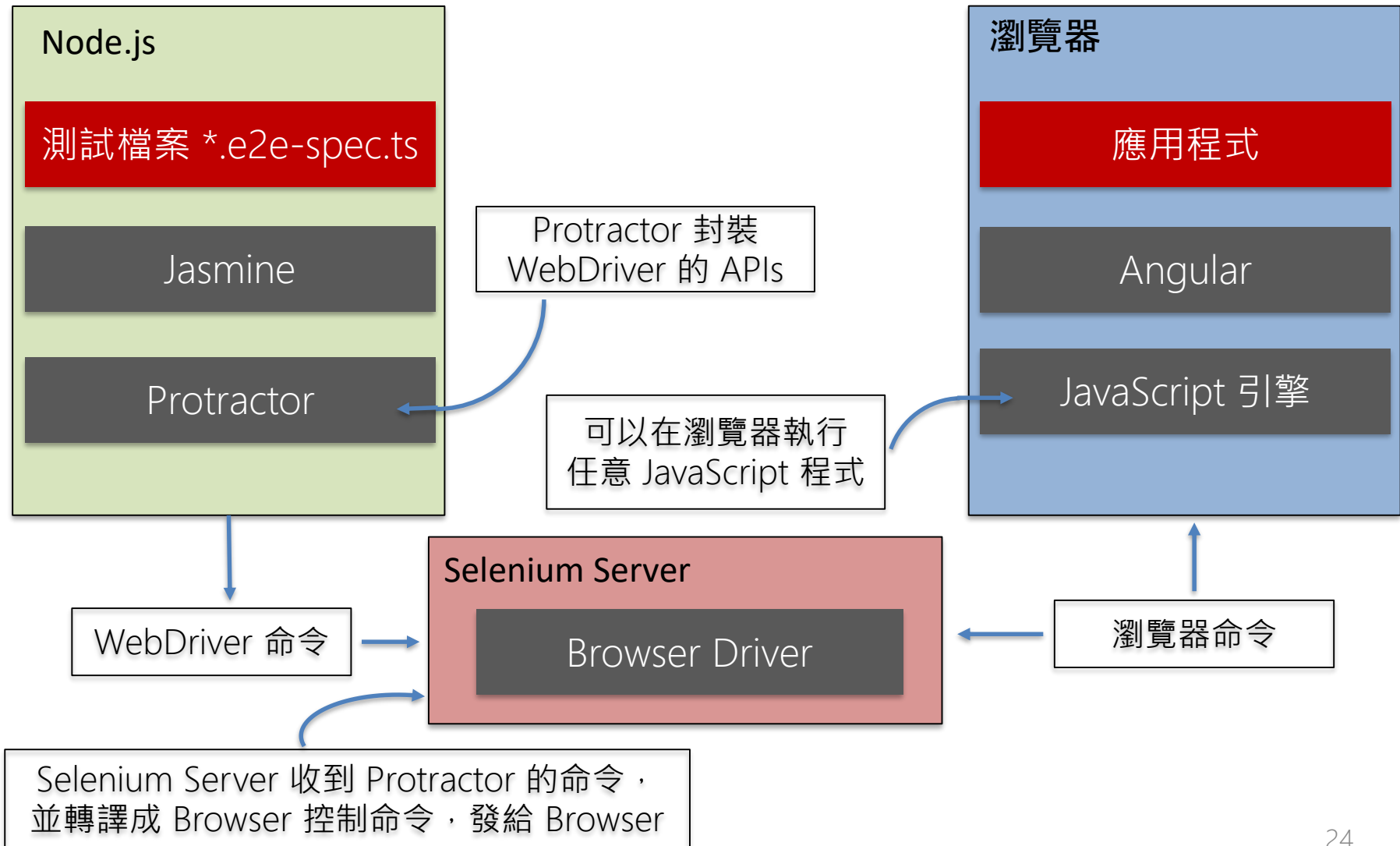
# Protractor 快速上手



# Protractor 環境要求與安裝

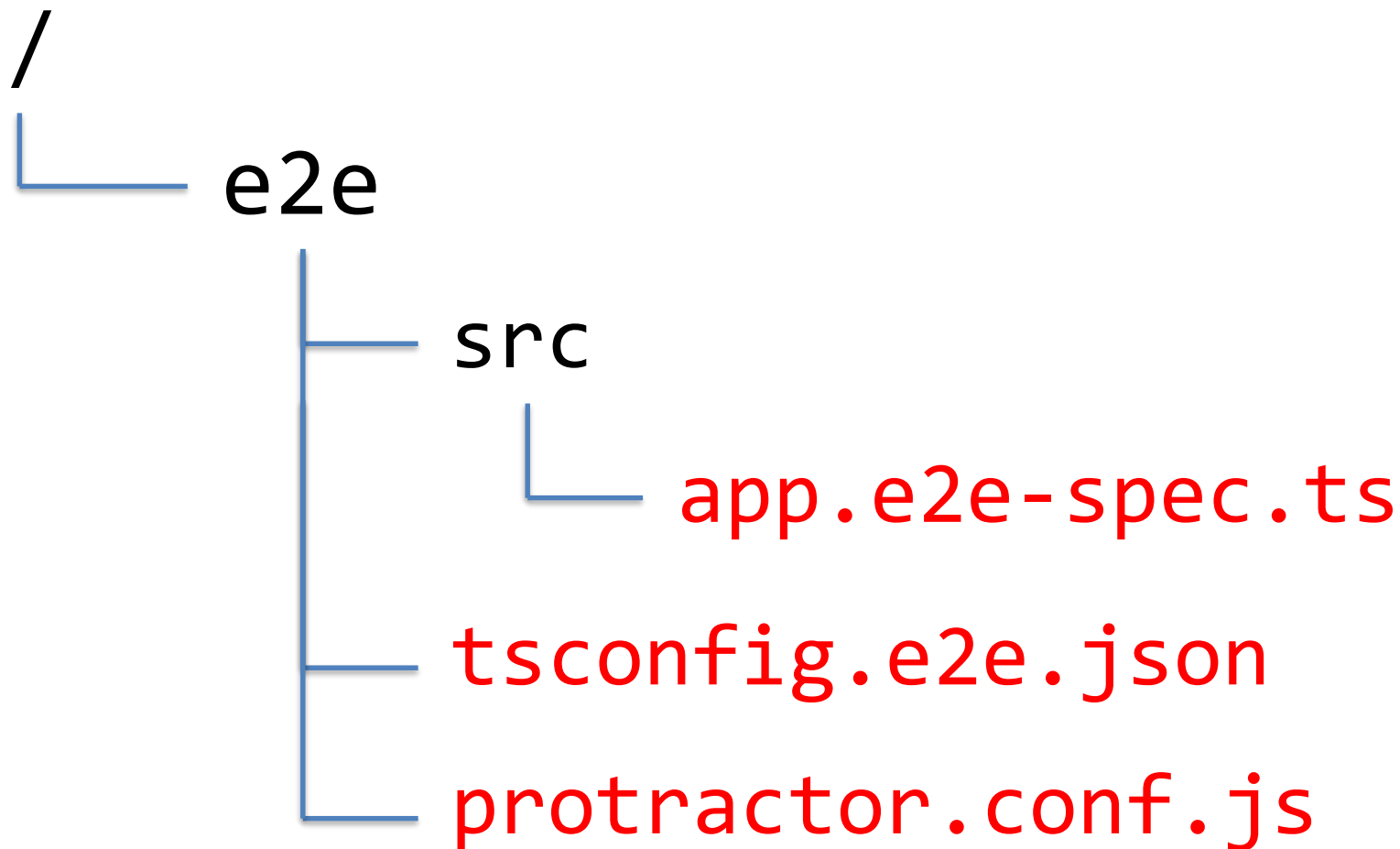
- 環境要求
  - [Node.js](#)
  - [Java Runtime \(JRE\)](#)
    - 執行 [Selenium Server](#) 的必要條件 (為了跑 IE 自動化測試)
- Protractor 套件內容
  - [Protractor API \(Library\)](#)
  - Protractor command line tools (**protractor --help**)
  - [Webdriver-manager command line tools](#)
  - [Selenium Server Jar](#)
  - [Web Driver](#)

# Protractor 運作機制





# 認識 Protractor 檔案結構



# package.json

- npm 用於管理 protractor 與其相依套件

```
{  
  "name": "protractor-tests",  
  "scripts": {  
    "e2e": "protractor"  
  },  
  "devDependencies": {  
    "@types/jasmine": "^2.53.43",  
    "@types/jasminewd2": "^2.0.1",  
    "@types/selenium-webdriver": "^3.0.0",  
    "typescript": "^2.2.1",  
    "protractor": "^5.1.1",  
    "ts-node": "^2.1.0"  
  }  
}
```

執行 Protractor 測試的指令

# e2e/tsconfig.e2e.json

```
{  
  "extends": "../tsconfig.json",  
  "compilerOptions": {  
    "outDir": "../out-tsc/app",  
    "module": "commonjs",  
    "target": "es5",  
    "types": [  
      "jasmine",  
      "jasminewd2",  
      "node"  
    ]  
  }  
}
```

# e2e/src/app.e2e-spec.ts

- 你撰寫的測試腳本

```
import { browser } from 'protractor';

describe('your first protractor test', () => {

  it('should load a page and verify the url', () => {

    browser.get('/');

    expect(browser.getCurrentUrl()).toEqual(browser.baseUrl);

  });

});
```

# protractor.conf.js (主要 protractor 設定檔)

```
exports.config = {  
  capabilities: {  
    browserName: 'chrome',  
  },  
  directConnect: true,  
  baseUrl: 'http://localhost:4200/',  
  framework: 'jasmine',  
  specs: [  
    './e2e/first-test.e2e-spec.ts'  
  ],  
  onPrepare: () => { ... }  
}
```

要執行的瀏覽器

設定直接控制瀏覽器

要執行的測試檔案

執行測試前的準備工作

# 執行 Protractor 的方法

- 透過全域 npm 套件執行
  - `protractor e2e/protractor.conf.js`
- 透過本地 npm 套件執行
  - `npx protractor e2e/protractor.conf.js`
  - `npm run protractor e2e/protractor.conf.js`
- 透過 Angular CLI 執行
  - `ng e2e` (預設會啟動開發伺服器)
  - `ng e2e --dev-server-target=` (預設不啟動開發伺服器)

# 範例專案的 package.json 說明

```
"scripts": {
```

```
  "protractor": "protractor",
```

下載 Chrome, Firefox  
WebDriver 到本地路徑

```
  "postinstall": "webdriver-manager update",
```

下載 IE WebDriver 到本地路徑

```
  "wdmupdateie": "webdriver-manager update --ie",
```

執行 Selenium Server

```
  "wdmstart": "webdriver-manager start",
```

等同執行 `protractor e2e/protractor.conf.js`

```
  "e2e": "ng e2e --dev-server-target= --webdriver-update=false",
```

```
  "e2e-golden": "ng e2e --dev-server-target= --webdriver-  
update=false --protractor-config=e2e/protractor-goldens.conf.js"
```

```
}
```

等同執行 `protractor e2e/protractor-goldens.conf.js`

# 寫測試之前要先懂的事

- 操控瀏覽器的物件 [browser](#)
  - 開啟網址
  - 取得網址
- 操控 DOM 的物件 [element](#)
  - 執行元件的互動
  - 輸入文字
  - 點擊事件
- 定位 DOM 的位置 [locator](#)
  - 提供各種 API 定位 DOM 物件
  - `by.id`, `by.className` 等等...



# 常用 Protractor APIs

執行動作	APIs
開啟網址	<code>browser.get('/user/login')</code>
取得網址	<code>browser.getCurrentUrl()</code>
取得基底網址	<code>browser.baseUrl</code>
取得 DOM 物件	<code>element(by.id('name'))</code> <code>element(by.css('.email'))</code> <code>element(by.className('table'))</code> <code>element(by.name('name'))</code> <code>element(by.buttonText('儲存'))</code>
操作 DOM 物件	<code>element(locator).click()</code> <code>element(locator).sendKeys('abc')</code> <code>element(locator).sendKeys(Key.ENTER)</code>

# 第一個 E2E 測試

- 測試案例 (login.e2e-spec.ts)
  1. 進入登入頁面
  2. 輸入帳號密碼
  3. 驗證登入成功後的網址是否正確



登入

使用者名稱:

密碼:

登入 取消

```
import { browser, by, element } from 'protractor';

describe('the user try to login', () => {
  it('should login to event page', () => {
    browser.get('/user/login');
    element(by.id('userName')).sendKeys('John');
    element(by.id('password')).sendKeys('123456');
    element(by.buttonText('登入')).click();
    expect(browser.getCurrentUrl()).toContain('events');
  });
});
```

# 實戰演練：完成登入功能的 E2E 測試

- <http://localhost:4200/user/login>
- 主要程式：**login.e2e-spec.ts**
- 測試案例
  - 輸入帳號 **John** 與密碼 **123456**
  - 按下「登入」按鈕執行登入 (登入成功)
  - 檢查瀏覽器網址是否進入 **/events** 頁面
- 練習目標
  - 練習開發基本 E2E 測試情境

# 認識 Control Flow 運作機制 (即將棄用)

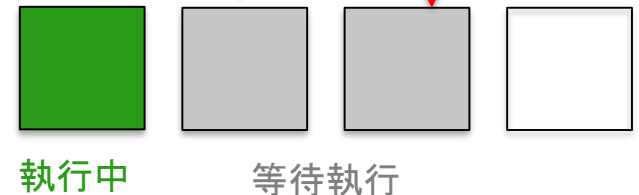
- WebDriverJS API (Protractor) 完全是**非同步**的，所有函式的回傳值都是 Promise 物件。
- WebDriverJS 維護一份 Pending Promise 佇列 (Queue)，確保所有要執行的命令都能**依序執行**，這樣的機制稱為 Control Flow 機制。
- 優點：簡化程式撰寫，不用撰寫 Promise API
- 缺點：偵錯不易，因為 Promise 物件結構複雜！

# 講解 Control Flow 運作機制

```
import { browser, by, element } from 'protractor';

describe('the user try to login', () => {
  it('should login to event page', () => {
    browser.get('/');
    element(by.id('userName')).sendKeys('John');
    element(by.id('password')).sendKeys('123456');
    element(by.buttonText('Login')).click();
    expect(browser.getCurrentUrl()).toContain('events');
  });
});
```

Control Flow 確保程式按照順序執行！

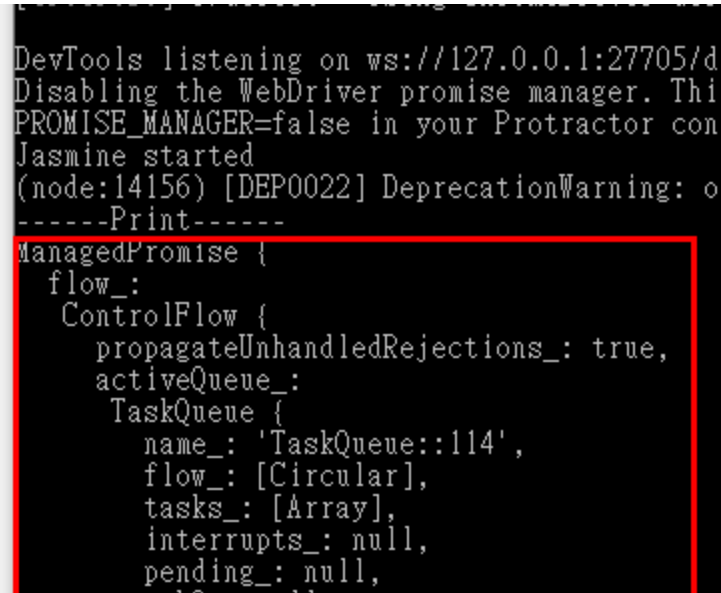


# 關於 Control Flow 的缺點

- 偵錯是一件麻煩的事情，無法輕易取得想得到的內容

```
it('should login to event page', () => {  
  browser.get('/');  
  element(by.id('userName')).sendKeys('John');  
  element(by.id('password')).sendKeys('123456');  
  element(by.buttonText('Login')).click();  
  const url = browser.getCurrentUrl();  
  console.log('-----Print-----');  
  console.log(url);  
  expect(url).toContain('events');  
});
```

```
it('should login to event page', () => {  
  page.navigateTo();
```



```
DevTools listening on ws://127.0.0.1:27705/d  
Disabling the WebDriver promise manager. This  
PROMISE_MANAGER=false in your Protractor con  
Jasmine started  
(node:14156) [DEP0022] DeprecationWarning: o  
-----Print-----  
ManagedPromise {  
  flow_:  
    ControlFlow {  
      propagateUnhandledRejections_: true,  
      activeQueue_:  
        TaskQueue {  
          name_: 'TaskQueue::114',  
          flow_: [Circular],  
          tasks_: [Array],  
          interrupts_: null,  
          pending_: null,
```

- Protractor/Selenium 未來將不支援 Control Flow
  - [Deprecate and remove the WebDriverJS promise manager](#)
  - protractor 變更紀錄：[CHANGELOG.md](#)

# 停用 Control Flow 的方法 (必須設定)

- 請到 [protractor.conf.js](#) 加入以下設定

```
exports.config = {  
  specs: [  
    './src/**/*.e2e-spec.ts'  
  ],  
  SELENIUM_PROMISE_MANAGER: false,  
  capabilities: {  
    browserName: 'chrome',  
  },  
};
```

# 停用 Control Flow 的設定調整

- 請到 [protractor.conf.js](#) 加入 **async** 關鍵字

```
async onPrepare() {  
  require('ts-node').register({  
    project: require('path').join(  
      __dirname, './tsconfig.e2e.json')  
    });  
  jasmine.getEnv().addReporter(new SpecReporter({  
    spec: {  
      displayStacktrace: true  
    })  
  }));  
}
```



# Node.js 7.6.0+ 完整支援 async/await

- 使用 async/await 改寫測試

```
import { browser, by, element } from 'protractor';

describe('the user try to login', () => {
  it('should login to event page', async () => {
    await browser.get('/');
    await element(by.id('userName')).sendKeys('John');
    await element(by.id('password')).sendKeys('123456');
    await element(by.buttonText('Login')).click();
    const url = await browser.getCurrentUrl();
    expect(url).toContain('events');
  });
});
```

# 使用 async/await 偵錯更容易

- 直接取得回傳值

```
it('should login to event page', async () => {  
  await browser.get('/');  
  await element(by.id('userName')).sendKeys('John');  
  await element(by.id('password')).sendKeys('123456');  
  await element(by.buttonText('Login')).click();  
  const url = await browser.getCurrentUrl();  
  console.log('-----Print-----');  
  console.log(url);  
  expect(url).toContain('events');  
});
```

```
D:\Solutions\Laboratory\Protract  
[22:00:20] I/launcher - Running  
[22:00:20] I/direct - Using Chrome  
DevTools listening on ws://127.0.0.1:4200/...  
Disabling the WebDriver promise  
IUM_PROMISE_MANAGER=false in you  
Jasmine started  
(node:8924) [DEP0022] Deprecati  
-----Print-----  
http://localhost:4200/events  
  
the user try to login  
✓ should login to event page
```

- 透過 async/await 才能使用 [chrome inspector](#) 偵錯

# 快速取得 CSS Selector 方法

- 安裝 Chrome Extension
  - [Copy Css Selector](#)
    - 右鍵點選任意元素 → 點擊 Copy CSS Selector 複製
  - [WebDriver Scripting Assistant](#)
    - 右鍵點選任意元素 → 點擊 Pick CSS Selector 複製
- 透過 F12 開發者工具
  - Elements → 滑鼠右鍵 → Copy → Copy selector

# 實戰演練：完成登入功能的 E2E 測試

- <http://localhost:4200/user/login>
- 主要程式：**login.e2e-spec.ts**
- 測試案例
  - 輸入帳號 **John** 與密碼 **123456**
  - 按下「登入」按鈕執行登入 (登入成功)
  - 檢查瀏覽器網址是否進入 **/events** 頁面
- 練習目標
  - 停用 Control Flow 機制
  - 使用 async/await 撰寫 E2E 測試

# 實戰演練：登入失敗的 E2E 測試

- <http://localhost:4200/user/login>
- 主要程式：**login.e2e-spec.ts**
- 測試案例
  - 輸入帳號 **John** 與密碼 **abc**
  - 按下「登入」按鈕執行登入 (登入失敗)
  - 檢查是否出現「**錯誤的帳號密碼**」訊息
- 練習目標
  - 使用 `async/await` 撰寫 E2E 測試
  - 定位 DOM 物件練習
  - 可使用 Visual Studio Code 進行偵錯 ([第 124 頁](#))

Angular and Protractor integration: NgZone

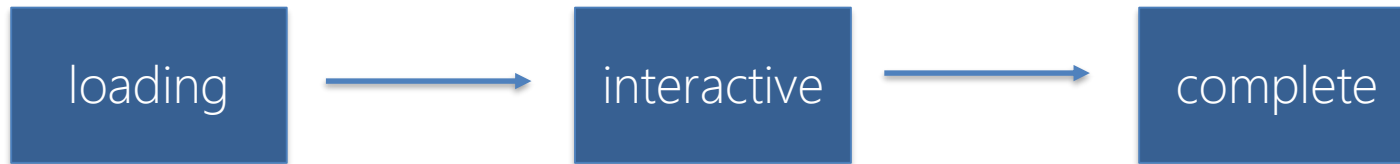
# Angular 與 Protractor 整合細節



# 原生 WebDriver API 等待頁面的機制 (1)

- 非 SPA 網頁每一次 換頁 (送出表單) 都會重新渲染網頁，Protractor 底層的 WebDriver API 會等待 document 的 readyState 狀態變成 `complete` 時，才會繼續執行下一步！

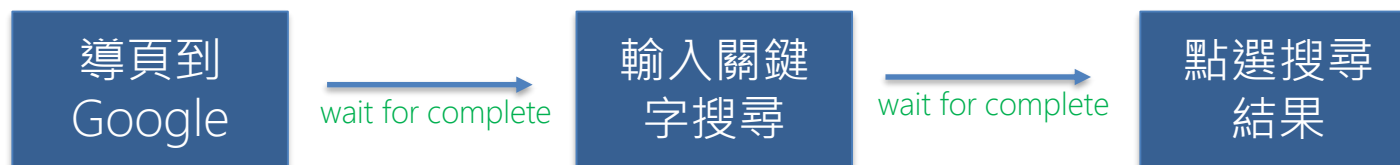
document.readyState 總共有三種狀態



- 官方文件
  - [void get\(\)](#)
  - [void submit\(\)](#)

# 原生 WebDriver API 等待頁面的機制 ( 2 )

- 以 Google 搜尋頁面為例 (非 SPA 網頁)



- 範例程式碼

```
it('should do search on google', async () => {  
  await browser.driver.get('https://www.google.com.tw/');  
  // 等待 document.readyState == 'complete'  
  const q = browser.driver.findElement(by.name('q'));  
  await q.sendKeys('protractor');  
  await q.sendKeys(Key.ENTER);  
  // 等待 document.readyState == 'complete'  
  const link = browser.driver.findElement(by.partialLinkText(  
    'end-to-end testing'));  
  await link.click();  
});
```



# Angular 網站使用原生 WebDriver API 的問題

- SPA 網站在 "**換頁**" 的時候 `document.readyState` 並不會發生改變 (因為並沒有真的換頁)
  - 當 SPA 網站載入完畢後，因為在導覽不同頁面時並沒有真正換頁，因此 `document.readyState` 永遠是 `complete` 狀態！
  - 特定網頁在送出表單資料時，大多透過 `XMLHttpRequest` 非同步方法更新部分 DOM 物件，過程中 `document.readyState` 也永遠是 `complete` 狀態！
- 所以原生 WebDriver API 並不會等待網頁更新完畢，就立即執行下一步驟！

# 簡介 Angular 的 [NgZone](#) 物件

- 主要用途
  - 管理**非同步事件**的**執行與錯誤追蹤**
  - 可以精確掌握非同步事件的運作時機
  - 應用程式生命週期中所有非同步事件都由 [NgZone](#) 管理
- 常見的非同步 API
  - setTimeout
  - setInterval
  - XMLHttpRequest
  - 所有 DOM 事件 (click, change, keyup, ...)

# Protractor API 背後的運作機制

```
element(by.buttonText('Login')).click();
```

## 1. 檢查 Angular NgZone 是否處於**穩定狀態**

如果 NgZone 一直處於不穩定狀態，Protractor 就會一直等待網頁進入穩定狀態，直到 Timeout 錯誤發生！

## 2. 透過 ElementFinder 尋找 DOM 物件

## 3. 檢查 Angular NgZone 是否處於**穩定狀態**

## 4. 對選中的 DOM 執行 click() 動作

原理說明：

<https://www.protractortest.org/#/infrastructure>

# 會等待 NgZone 穩定的 APIs

- 取得網頁資訊的 APIs

- `browser.getCurrentUrl()`
- `browser.getPageSource()`
- `browser.getTitle()`

- 操作 DOM 物件的 APIs

- `browser.findElement(by.id('xx'))`
- `element(by.id('xx')).click()`
- `element(by.id('xx')).sendKeys()`
- `element(by.id('xx')).getTagName()`
- `element(by.id('xx')).isPresent()`
- `element(by.id('xx')).takeScreenshot()`

# 不會等待 NgZone 穩定的 APIs

- 瀏覽器操作的 APIs 不會等待 NgZone 穩定
  - `browser.get()`
  - `browser.restart()`
  - `browser.executeScript()`
  - `browser.takeScreenshot()`
  - `browser.refresh()`
  - `browser.navigate().back();`
  - `browser.actions().mouseMove(element).click().perform();`
  - `browser.switchTo()`
  - `browser.manage().window().setSize(1366, 1024)`

# 實戰演練：體驗卡關的感覺

- <http://localhost:4200/labs/questionnaire>
- 主要程式：
  - `questionnaire.e2e-spec.ts`
  - `questionnaire.component.ts`
- 測試案例
  - 輸入姓名 **John** 與 最喜歡的語言 **C#**
  - 按下「送出」按鈕
  - 驗證是否「送出成功」
- 練習目標
  - 感受 Protractor 為何會因為特定程式碼而導致無限等待

# 正確使用連續的非同步事件

```
export class AppComponent implements OnInit {  
  constructor(private zone: NgZone) { }  
  
  ngOnInit() {  
    this.zone.runOutsideAngular(() => {  
      setInterval(() => {  
        // 執行 Protractor 不需要檢測的程式碼  
      }, 1000);  
    });  
  }  
}
```

# 認識 NgZone 基本用法

- 跳脫 NgZone 變更偵測
  - runOutsideAngular()

```
this.zone.runOutsideAngular(() => {  
    // 這裡的程式碼不會參與 NgZone 變更偵測  
});
```

- 返回 NgZone 變更偵測
  - run()

```
this.zone.run(() => {  
    // 將跳脫原本 NgZone 偵測範圍的程式碼通知 NgZone 有變更發生  
})
```



# 實戰演練：使用 NgZone 避開地雷

- <http://localhost:4200/labs/questionnaire>
- 主要程式：
  - `questionnaire.e2e-spec.ts`
  - `questionnaire.component.ts`
- 測試案例
  - 輸入姓名 **John** 與 最喜歡的語言 **C#**
  - 按下「送出」按鈕
  - 驗證是否「送出成功」
- 練習目標
  - 練習使用 NgZone 避免不容易測試的情境

ElementFinder & Locator

# 認識 ElementFinder 與 Locator



# 認識 ElementFinder 與 WebElement

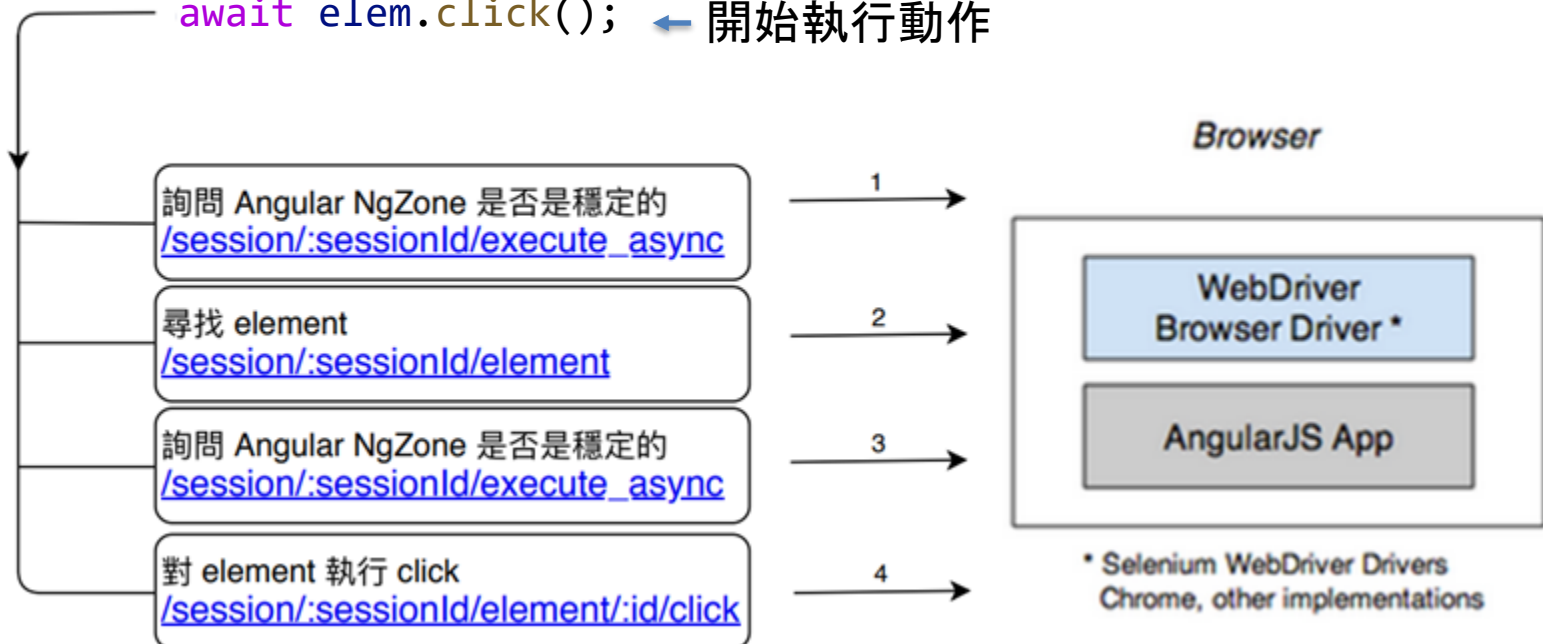
- WebElement
  - 來自 selenium-webdriver 套件中的類別
  - 主要用來控制瀏覽器中如何操作 DOM 物件
- ElementFinder
  - 來自 protractor 套件中的類別
  - 主要繼承自 WebElement 類別，並提供額外的功能
    - 例如 element(locator).isPresent() 就只有 ElementFinder 才有
  - 宣告的時候並不會立刻呼叫 WebElement 的 API，而是在有 DOM 操作行為的時候才會執行程式
    - `element(locator).sendKeys()`
    - `element(locator).click()`

# 三種尋找 DOM 物件的方式 (1)

- `element()` 或 `browser.element()`
  - 回傳型別 `ElementFinder` (繼承自 `WebElement`)

`const elem = element(by.id('btn'));` ← 此時還不會執行任何動作

`await elem.click();` ← 開始執行動作



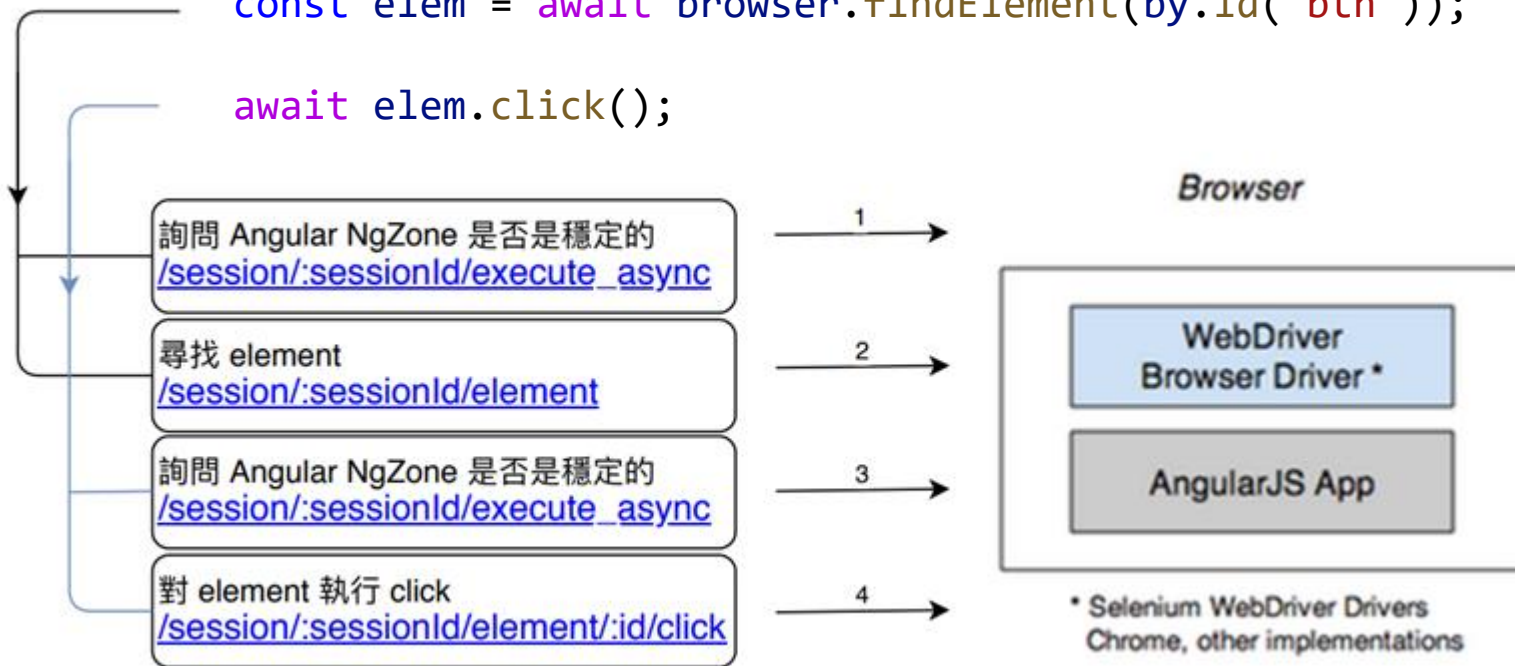
# 三種尋找 DOM 物件的方式 (2)

- `browser.findElement()`
  - 回傳型別 [WebElement](#)

此步驟會直接搜尋 DOM

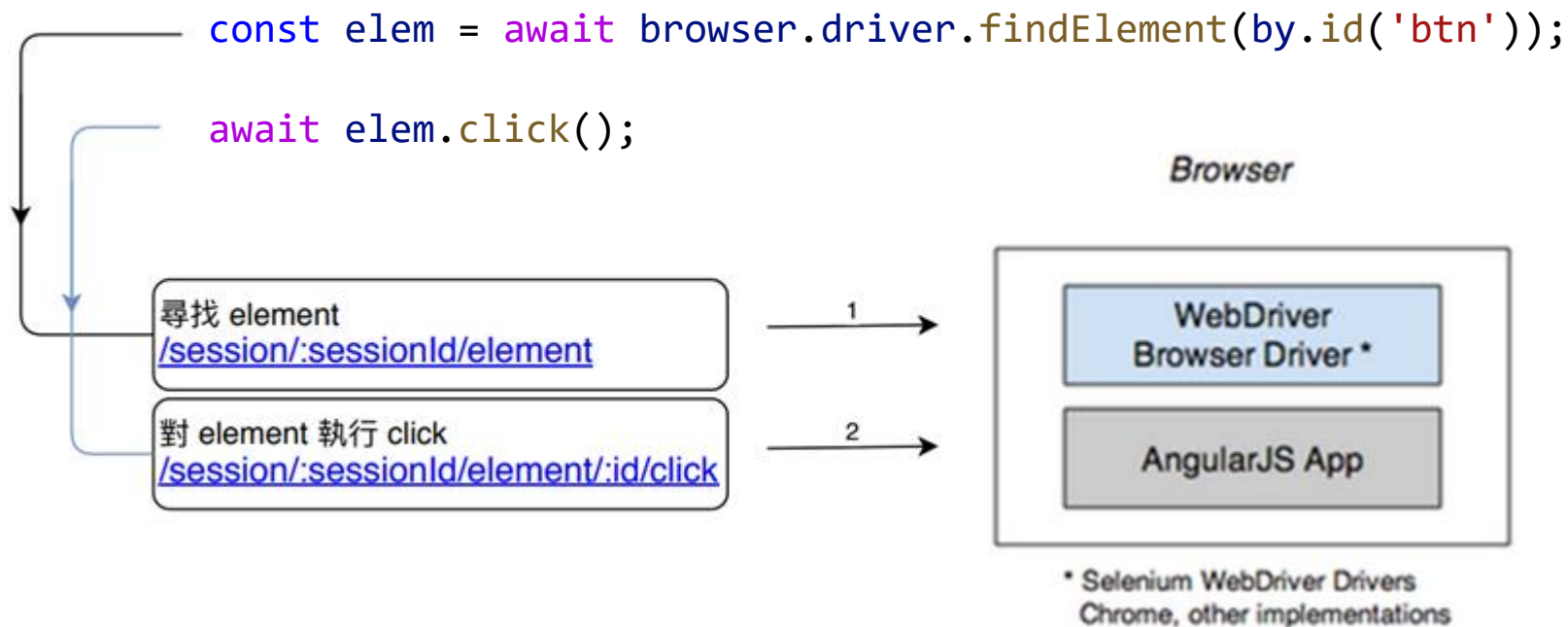
```
const elem = await browser.findElement(by.id('btn'));
```

```
await elem.click();
```



# 三種尋找 DOM 物件的方式 ( 3 )

- `browser.driver.findElement()`
  - 回傳型別 [WebElement](#)
  - 直接透過 WebDriver 底層 API 進行尋找 DOM 物件
  - 這種方法適用於**非 Angular 頁面**



# 認識 ElementFinder 常用 API (1)

ElementFinder	使用方法
<u><a href="#">sendKeys</a></u>	<pre>await element(by.id('name')).sendKeys('abc');  expect(await element(by.id('name')).getAttribute('value')).toBe('abc');</pre>
<u><a href="#">clear</a></u>	<pre>const name = element(by.id('name')); await name.sendKeys('abc'); await name.clear();  expect(await name.getAttribute('value')).toBe('');</pre>
<u><a href="#">click</a></u>	<pre>await element(by.id('btn')).click();</pre>
<u><a href="#">getText</a></u>	<pre>&lt;table&gt; &lt;tr&gt;&lt;td&gt;Hello&lt;/td&gt;&lt;/tr&gt; &lt;/table&gt; expect(await element(by.tagName('table')).getText())   .toBe('Hello');</pre>

# 認識 ElementFinder 常用 API (2)

ElementFinder	使用方法
<u>getTagName</u>	<pre>let body = element(by.tagName('body')); expect(await body.getTagName()).toBe('body');</pre>
<u>getCssValue</u>	<pre>let toolbar = element(by.tagName('mat-toolbar')); expect(await toolbar.getCssValue('background-color')). toBe('rgba(33, 150, 243, 1)');</pre>
<u>getAttribute</u>	<pre>let email = element(by.id('email')); await email.sendKeys('abc'); expect(await email.getAttribute('value')).toBe('abc');</pre>



# 認識 ElementFinder 常用 API (3)

ElementFinder	使用方法
<u>isPresent</u>	<pre>await element(by.css('#def')).isPresent();</pre> <p>※ 檢查是否有找到 DOM 物件 (回傳 bool 型別)</p>
<u>isElementPresent</u>	<pre>// 這兩段是一樣的事情 await element(by.css('#abc')).element(by.css('#def')).isPresent();  await element(by.css('#abc')).isElementPresent (by.css('#def'));</pre>
<u>isDisplayed</u>	<pre>&lt;div id="foo" style="visibility:hidden"&gt; &lt;div id="foo" style="display:none"&gt; // 判斷 DOM 是否顯示在網頁上 var foo = element(by.id('foo')); expect(await foo.isDisplayed()).toBe(false);</pre>

# 認識 ElementArrayFinder 類別

- 取得所有定位到的 DOM 物件
  - `element.all(locator)`
  - `$$('.xx')`
  - 可用 `get(index)` 取得指定元件

```
<ul>
  <li>abc</li>
  <li>123</li>
  <li>xyz</li>
</ul>
```

範例

```
let secondli = element.all(by.tagName('li')).get(1);

expect(secondli.getText()).toBe('123');
```

# 常用 Locator (用於定位 DOM 物件)

Locator	使用方法
<a href="#"><u>by.css</u></a>	<pre>&lt;input class="contact-email" type="email"&gt; let e = element(by.css('input[type=email]'));</pre>
<a href="#"><u>by.id</u></a>	<pre>&lt;input id="contact-email" type="email"&gt; let email = element(by.id('contact-email'));</pre>
<a href="#"><u>by.buttonText</u></a> <a href="#"><u>by.partialButtonText</u></a>	<pre>&lt;button&gt;Submit&lt;/button&gt; let f = element(by.buttonText( 'Submit')); let p = element(by.partialButtonText('Sub'));</pre>
<a href="#"><u>by.cssContainingText</u></a>	<pre>&lt;ul&gt;   &lt;li class="pet"&gt;Dog&lt;/li&gt;   &lt;li class="pet"&gt;Cat&lt;/li&gt; &lt;/ul&gt;  let dog = element(by.cssContainingText('.pet', 'Dog'));</pre>

# 常用 Locator (用於定位 DOM 物件)

Locator	使用方法
<u><a href="#">by.linkText</a></u> <u><a href="#">by.partialLinkText</a></u>	<pre>&lt;a href="/add"&gt;Add contact&lt;/a&gt;  let f = element(by.linkText('Add contact')); let p = element(by.partialLinkText('contact'));</pre>
<u><a href="#">by.tagName</a></u>	<pre>&lt;app-contact-detail&gt;...&lt;/app-contact-detail&gt;  let tag = element(by.tagName('app-contact- detail'));</pre>
<u><a href="#">by.name</a></u>	<pre>&lt;input name="contact-email" type="email"&gt;  let e = element(by.name('contact-email '));</pre>

# 其他 Locator

Locator	使用方法
<a href="#">by.xpath</a> 不建議使用	<pre>&lt;ul&gt;&lt;li&gt;&lt;a&gt;Foobar&lt;/a&gt;&lt;/li&gt;&lt;/ul&gt; let xpath = element(by.xpath('//ul/li/a'));</pre>
<a href="#">by.model</a> 只會在 AngularJS 1.x 使用	<pre>&lt;span ng-bind="contact.name"&gt;&lt;/span&gt; let binding = element(by.binding('contact.name'));</pre>
<a href="#">by.binding</a> 只會在 AngularJS 1.x 使用	<pre>&lt;input ng-model="contact.name"&gt; let model = element(by.model('contact.name'));</pre>
<a href="#">by.repeater</a> 只會在 AngularJS 1.x 使用	<pre>&lt;div ng-repeat="cat in pets"&gt;   &lt;span&gt;{{cat.name}}&lt;/span&gt;   &lt;span&gt;{{cat.age}}&lt;/span&gt; &lt;/div&gt;  var secondCat = element( by.repeater('cat in pets').row(1));</pre>

# 實戰演練：練習複雜 DOM 定位運用

- <http://localhost:4200/events>
- 主要程式：**search-event.e2e-spec.ts**
- 測試案例
  - 於搜尋文字方塊輸入 **Angular** 並點擊**搜尋**
  - 驗證畫面上是否出現搜尋結果
  - 驗證畫面上是否出現 3 個搜尋結果
  - 點擊 **Angular 實戰開發** 並進入下一頁
  - 驗證活動標題是否為「**ANGULAR 7 開發實戰：新手入門篇**」
- 練習目標
  - 熟悉 Locator 的應用
  - 練習使用 [ElementArrayFinder](#) 搜尋物件

Forms

# 常見的表單 E2E 測試



# 送出表單

- 直接透過**按鈕文字**搜尋 DOM 物件

```
<button type="submit">儲存</button>
```

```
const btn = element(by.buttonText('儲存'));
```

- 常見互動方式
  - `await btn.click()`



# 文字輸入框

- 常見定位方式

```
<input name='presenter' id="presenter" type="text"/>
```

```
const p = element(by.name('presenter'));
```

```
const p = element(by.id('presenter'));
```

```
const p = $('input[name=presenter]');
```

- 常見互動方式

- `await p.sendKeys('John')`

- 驗證方式

- `expect(await p.getAttribute('value')).toBe('John')`

# 下拉式選單

- 點擊到下拉式選單的選項

```
<select name="mylist">  
  <option value="1">選項 A</option>  
  <option value="2">選項 B</option>  
</select>
```

```
const mylist = element(by.name('mylist'));
```

```
// 方法一：用 Index 選擇
```

```
await mylist.all(by.tagName('option')).get(0).click();
```

```
// 方法二：用選項文字選擇
```

```
await mylist.element(by.cssContainingText('option', '選項 A'))  
.click();
```

- 驗證方式

- expect(await mylist.getAttribute('value')).toBe('2');

# 核取方塊 (Checkbox)

- 常見選擇方式

```
<input name='level' type="checkbox" value='初級' />  
<input name='level' type="checkbox" value='中級' />  
  
const chk =  
  element(by.css(`input[type=checkbox][name=level][value=初級]`));  
const chk2 =  
  element(by.css(`input[type=checkbox][name=level][value=中級]`));
```

- 互動方式

- `await chk.click();`

- 驗證是否勾選

- `expect(await chk.getAttribute('checked')).toBeTruthy();`

# 單選按鈕 (Radio Button)

- 常見選擇方式

```
<input name="gender" type="radio" value="男生"/>
<input name="gender" type="radio" value="女生"/>

const male =
  element(by.css(`input[type=radio][name=gender][value=男生]`));
const female =
  element(by.css(`input[type=radio][name=gender][value=女生]`));
```

- 互動方式

- `await female.click();`

- 驗證是否選擇

```
expect(await female.getAttribute('checked')).toBeTruthy();
```

# 檔案上傳

- 使用 Node.js 的 [path](#) 套件將路徑轉成絕對路徑
  - `npm install path --save-dev`
- 互動方式
  - 使用相對路徑選擇檔案並透過 `path` 轉成絕對路徑
  - 透過 `sendKeys` 將要上傳的檔案路徑輸入到 input 中

```
<input name="imageFile" type="file" />

import * as path from 'path';

const imgPath = path.resolve('./e2e/src/assets/Angular.png');
await element(by.name('imageFile')).sendKeys(imgPath);
```

- 驗證方式

```
expect(await imageFile.getAttribute('value')).toBeTruthy();
```

# 多重檔案上傳

- 檔案上傳需用絕對路徑

```
<input name="imageFile" type="file" multiple />

let files = [];
files.push('./e2e/src/assets/Angular1.png');
files.push('./e2e/src/assets/Angular2.png');
files.push('./e2e/src/assets/Angular3.png');

const imgPath = files.map(x => path.resolve(x)).join('\n');

await element(by.name('imageFile')).sendKeys(imgPath);
```

- 驗證方式

```
expect(await imageFile.getAttribute('value')).toBeTruthy();
```

# 實戰演練：練習表單操作

- <http://localhost:4200/events/new>
- 主要程式：**create-event.e2e-spec.ts**
- 測試案例
  - 輸入活動名稱 **Protractor 實戰**
  - 使用 **Datepicker** 選擇活動日期 **2019/3/16**
    - 請勿使用 `sendKeys` 測試
    - **進階練習**：請嘗試設定 **1997/12/31** 這個日期！
  - 輸入活動時間 **早上**；輸入活動票價 **500**
  - 輸入活動地址 **中正路100號**；城市 **台北市**；國家 **台灣**
  - 輸入活動網址 **http://example.com**
  - 上傳活動圖片 **e2e\src\assets\Protractor.png**
  - 點擊**儲存**按鈕，驗證活動列表顯示 **Protractor 實戰** 活動

# 實戰演練：練習表單操作

- <http://localhost:4200/events/1>
- 主要程式：**create-session.e2e-spec.ts**
- 測試案例
  - 點擊 **建立議程**
  - 輸入議程名稱 **Protractor 表單練習**
  - 輸入主講人 **John**
  - 選擇演講時間 **一小時**
  - 選擇場次 **上午場**；選擇適合程度 **初級、中級**
  - 輸入演講內容 **自動化 Protractor 表單**
  - 點擊**儲存**按鈕，驗證議程列表顯示 **Protractor 表單練習**



ExpectedConditions

# 等待預期條件的設計方式



# 認識 ExpectedConditions 類別

- 讓瀏覽器等待特定預期條件成立才會繼續執行測試

```
import { browser, ExpectedConditions as EC } from 'protractor';  
await browser.wait(EC.alertIsPresent(), 5000);
```

- 適用情境
  - 非 Angular 應用程式
  - 等待使用者輸入 (例如輸入 CAPTCHA 要求的圖片文字)
  - 等待不在 NgZone 監視範圍內觸發的 DOM 事件
  - 等待 **alert()** 出現彈跳視窗出現

# browser.wait 用法

- 需搭配 [browser.wait\(arg1, arg2, arg3\)](#) 使用

參數	說明
arg1	等待 ExpectedConditions 條件
arg2	等待逾時時間，預設 <u><a href="#">defaultTimeoutInterval</a></u> 毫秒
arg3	逾時的錯誤訊息

# 單一條件與多重組合條件

- 單一等待條件

```
await browser.wait(EC.urlContains('foo'), 5000);
```

- 結合多個等待條件
  - [EC.or](#), [EC.and](#), [EC.not](#)

```
const titleContainsFoo = EC.titleContains('Foo');  
const titleIsNotFooBar = EC.not(EC.titleIs('FooBar')); // 反向條件  
await browser.wait(EC.and(titleContainsFoo, titleIsNotFooBar), 5000);
```

# ExpectedCondition APIs

ExpectedCondition	說明
<a href="#"><u>alertIsPresent</u></a>	判斷是否出現 alert 視窗  <code>await browser.wait(EC.alertIsPresent(), 5000);</code>
<a href="#"><u>elementToBeClickable</u></a>	元素是否 visible 和 enabled  <code>await browser.wait(EC.elementToBeClickable(\$('#abc')), 5000);</code>
<a href="#"><u>textToBePresentInElement</u></a>	元素內容包含指定文字  <code>await browser.wait(EC.textToBePresentInElement(\$('#abc'), 'foo'), 5000);</code>

# ExpectedCondition APIs

ExpectedCondition	說明
<u><a href="#">textToBePresentInElementValue</a></u>	頁面元素 (輸入欄位) 的 <b>value</b> 等於指定文字 (比對的字串必須完全相同)  <pre>await browser.wait(EC.textToBePresentInElementValue(\$('#myInput'), 'foo'), 5000);</pre>
<u><a href="#">titleContains</a></u>	document.title 包含指定文字 (部分比對)  <pre>await browser.wait(EC.titleContains('foo'), 5000);</pre>
<u><a href="#">titleIs</a></u>	document.title 完全等於指定文字  <pre>await browser.wait(EC.titleIs('foo'), 5000);</pre>

# ExpectedCondition APIs

ExpectedCondition	說明
<a href="#"><u>urlContains</u></a>	網址包含指定字串  <code>await browser.wait(EC.urlContains('foo'), 5000);</code>
<a href="#"><u>urlIs</u></a>	比對完整網址  <code>await browser.wait(EC.urlIs('http://foo'), 5000);</code>
<a href="#"><u>presenceOf</u></a>	頁面中可以找到 DOM 物件 ( 此 DOM 物件無論是否顯示在畫面都算成立 )  <code>await browser.wait(EC.presenceOf(\$('#abc')), 5000);</code>
<a href="#"><u>stalenessOf</u></a>	頁面中已經找不到 DOM 物件，與 <code>presenceOf</code> 為相反邏輯。  <code>await browser.wait(EC.stalenessOf(\$('#abc')), 5000);</code>

# ExpectedCondition APIs

ExpectedCondition	說明
<u><a href="#">visibilityOf</a></u>	<p>元件有顯示在頁面上 visible，且寬高大於 0</p> <pre>browser.wait(EC.visibilityOf(\$('#abc')), 5000);</pre>
<u><a href="#">invisibilityOf</a></u>	<p>其邏輯與 visibilityOf 相反 (判斷元素不可見)</p> <pre>browser.wait(EC.invisibilityOf(\$('#abc')), 5000);</pre>
<u><a href="#">elementToBeSelected</a></u>	<p>預期 select、checkbox、radio button 被選擇</p> <pre>browser.wait(EC.elementToBeSelected(\$('#chk')), 5000);</pre> <pre>browser.wait(EC.elementToBeSelected(element(by.name('duration')).element(by.cssContainingText('option', '半小時'))))</pre>



# 實戰演練：練習 ExpectedCondition

- <https://angular.io/>
- 主要程式：**angulario\_search.e2e-spec.ts**
- 測試案例
  - 搜尋條件輸入 **ngzone**
  - 等待搜尋畫面
  - 點擊 **NgZone** 連結
  - 檢查頁面標題出現 **NgZone** 字樣
- 練習目標
  - 學習使用 ExpectedCondition

Advanced Browser Control

# 進階瀏覽器控制



# browser 常用 APIs

APIs	說明
<a href="#"><u>waitForAngularEnabled</u></a>	設定關閉/啟用是否自動等待 NgZone 穩定  <code>await browser.waitForAngularEnabled(false)</code>
<a href="#"><u>waitForAngular</u></a>	執行等待 NgZone 穩定  <code>await browser.waitForAngular();</code>
<a href="#"><u>get</u></a>	開啟指定網址的網頁  <code>await browser.get('/user/login')</code> <code>expect(await browser.getCurrentUrl()).toBe('http://localhost:4200/user/login');</code>
<a href="#"><u>getCurrentUrl</u></a>	取得現在的網址  <code>const url = await browser.getCurrentUrl()</code>

# browser 常用 APIs

APIs	說明
<a href="#"><u>executeScript</u></a>	注入任意 JavaScript 執行  <code>await browser.executeScript('window.scrollTo(0, 200);');</code>
<a href="#"><u>sleep</u></a>	暫停測試執行，用於人工觀察自動化測試的畫面  <code>await browser.sleep(10000);</code>
<a href="#"><u>getPageSource</u></a>	取得頁面當下的 HTML 原始碼（網頁渲染過後的內容）  <code>const pageSource = await browser.getPageSource(); expect(pageSource).toContain('送出成功');</code>
<a href="#"><u>getTitle</u></a>	取得 document.title  <code>await browser.getTitle();</code>

# browser 常用 APIs

APIs	說明
<a href="#"><u>takeScreenshot</u></a>	頁面呈現的截圖，回傳 base-64 encoded PNG  <code>await browser.takeScreenshot();</code>
<a href="#"><u>switchTo</u></a>	視窗之間的切換  <code>await browser.switchTo().alert()</code> <code>await browser.switchTo().window(windowhandle)</code>
<code>manage().window().maximize()</code>	視窗最大化  <code>await browser.manage().window().maximize();</code>
<code>manage().window().setSize();</code>	設定視窗大小  <code>await browser.manage().window().setSize(1240, 736);</code>

# browser 常用 APIs

APIs	說明
<a href="#"><u>manage().timeouts().implicitlyWait()</u></a>	<p>Protractor 預設 <code>findElement</code> 時，如果找不到該 DOM 物件會<b>立即</b>拋出找不到 DOM 物件的例外。</p> <p>設定<b>隱含等待</b>可以等待 DOM 物件超過秒數後才會拋出找不到 DOM 物件的例外。</p> <pre>// onPrepare await browser.manage().timeouts().implicitlyWait(5000);  // spec await element(by.className('well')).click();</pre> <p>適用於非 <i>Angular</i> 網站。</p>

# browser 常用 APIs

APIs	說明
<u><a href="#">actions()</a></u>	<p>1. 操縱滑鼠的 API</p> <pre>await browser.actions()   .mouseMove(element)   .mouseMove({x: 50, y: 0})   .click().perform();</pre> <p>2. 滑鼠拖曳 API 雖有 <b>dragAndDrop</b> 但目前無法使用 (<a href="#">ISSUE#3604</a>)</p> <pre><del>await browser.actions().dragAndDrop(elem1, elem2)   .perform();</del></pre> <p>目前可用第三方套件 <a href="#">html-dnd</a> 模擬拖曳效果</p> <pre>import { code as dragAndDrop } from 'html-dnd'; // 一定要用 browser.findElement const source = await browser.findElement(by.id('div1')); const target = await browser.findElement(by.id('div2')); await browser.executeScript(dragAndDrop, source, target);</pre>

# 自訂 Promise 達成等待條件

APIs	說明
<u>wait</u>	<p>等待<b>某個條件</b>成立，才往下執行</p> <pre>await browser.wait(waitForSomething, 10000, '等待某條件成立');  function waitForSomething() {   return new Promise((resolve, reject) =&gt; {     const interval = setInterval(() =&gt; {       if(CONDITION){         clearInterval(interval);         resolve('條件成立');       }     }, 500);   }); }</pre>

- 應用範例
  - 等待人工輸入特定欄位並符合自訂條件才繼續



# 認識 browser.switchTo() 使用情境

- 切換至 window 彈出視窗
  - window.alert()
  - window.confirm()
  - window.prompt()
- 切換至其他 window 視窗
  - browser.switchTo().window(windowhandle)
- 切換至特定 frame/iframe 視窗
  - browser.switchTo().frame(number | WebElement)

# 控制 Window 彈出視窗

- 切換至 window 彈出視窗
  - [browser.switchTo\(\).alert\(\)](#)
- 控制彈出視窗的動作 ([AlertPromise](#))

動作	APIs
同意或OK	<code>accept()</code>
取消	<code>dismiss()</code>
輸入文字	<code>sendKeys()</code>
取得內容	<code>getText()</code>

- 範例程式
  - `await browser.switchTo().alert().accept();`

# 切換至其他 window 視窗

- 使用 [getAllWindowHandles](#) 取得 windowhandle

```
const handles = await browser.getAllWindowHandles();
```

- 切換至 window 彈出視窗

```
browser.switchTo().window(windowhandle)
```

- 切換到第二個視窗範例程式碼

```
const handles = await browser.getAllWindowHandles();  
await browser.switchTo().window(handles[1]);
```

# 注入任意 JavaScript 執行

- 使用方式
  - [browser.executeScript\(\)](#)
  - [browser.executeAsyncScript\(\)](#)
- 控制 Scrollbar 範例一 (將畫面捲動至最下方)

```
await browser.executeScript(() => {  
    return window.scrollTo(0, document.body.scrollHeight);  
});
```
- 控制 Scrollbar 範例二 (將畫面捲動至最下方)

```
await browser.executeScript(  
    'window.scrollTo(0, document.body.scrollHeight);');
```
- 實務上不建議使用 (除非真的有必要使用的理由)

# 實戰演練：練習 window 彈出視窗操作

- <http://localhost:4200/events/new>
- 主要程式：**leave-creating-event.e2e-spec.ts**
- 測試案例
  - 點選 **取消** 按鈕
  - 點選 window 彈出視窗 **確定**
  - 驗證導頁到 <http://localhost:4200/events>
- 練習目標
  - window 彈出視窗的操作

# 實戰演練：練習切換不同 window 操作

- <http://localhost:4200/user/new>
- 主要程式：**new-user.e2e-spec.ts**
- 測試案例
  - 輸入使用者名稱 **mike**；輸入密碼 **123**
  - 輸入名字 **bob**；輸入姓氏 **joe**
  - 點選 **會員權益** 會彈出一個全新視窗 (靜態網頁)
  - 將會員權益視窗捲動到最底部，並點選 **同意** 按鈕
  - 返回**新增會員表單**視窗，點選**新增**
  - 驗證網址導向 <http://localhost:4200/events>

# 實戰演練：browser.wait 進階應用

- <http://localhost:4200/labs/captcha>
- 主要程式：**captcha.e2e-spec.ts**
- 測試案例
  - 等待手動輸入**四碼驗證碼**
  - 輸入四碼驗證碼後，測試自動點選 **送出**
  - 驗證畫面上顯示 **驗證碼正確**
- 練習目標
  - 學習自訂等待方法

Screen comparison E2E testing

## 防止網頁跑版的 E2E 測試





# 認識 blue-harvest 套件

- 主要用途
  - 提供 E2E 測試過程的畫面截圖比較
  - 提供好用的輔助方法 (Action Helpers)

- 安裝 [blue-harvest](#)

```
npm install blue-harvest --save-dev
```

# Golden / Actual / Diff



Golden  
( 預期的畫面 )



Actual  
( 實際的畫面 )



Diff  
( 比對後的差異 )

※ 所謂 Golden 圖片，就是進行 E2E 測試時預期呈現的網頁畫面截圖！

# 調整 blue-harvest 的執行參數 (1)

- 新增 **e2e/protractor-goldens.conf.js** 設定檔
  - `process.env['UPDATE_GOLDENS'] = "true"`
    - "1" 或 "true" 會更新(建立) Golden 圖
    - 其他設定會比較 Golden 圖

```
let config = require('./protractor.conf.js').config;  
  
// 如果 UPDATE_GOLDENS 為 "1" 或 "true" 則會自動更新 Golden 圖片  
process.env['UPDATE_GOLDENS'] = "true";  
  
// 設定只有以下這些 e2e-spec.ts 才會執行畫面截圖比對  
config.specs = ['./src/labs/golden.e2e-spec.ts'];  
  
exports.config = config;
```

## 調整 blue-harvest 的執行參數 (2)

- 可以透過設定環境變數的方式改變執行參數
  - Windows
    - `set UPDATE_GOLDENS=true`
    - `set UPDATE_GOLDENS=`
  - macOS
    - `export UPDATE_GOLDENS=true`
    - `unset UPDATE_GOLDENS`
- 請將 **SELENIUM\_PROMISE\_MANAGER** 設為 **false**
  - Windows
    - `set SELENIUM_PROMISE_MANAGER=false`
  - macOS
    - `export SELENIUM_PROMISE_MANAGER=false`

# 撰寫畫面呈現的 E2E 測試

- 注意事項
  - 確保每次視窗大小一致
  - 圖片儲存資料夾要預先建立好

```
it('should compare pages', async () => {  
  await browser.get('/');  
  await browser.manage().window().setSize(1366, 1024);  
  const golden = 'e2e/goldens/home.png';  
  const diffDir = 'e2e/goldens/'; // 會產生 diff-home.png  
  await browser.waitForAngular(); // 截圖前一定要 wait  
  const actual = await browser.takeScreenshot();  
  const result = await compareScreenshot(actual, golden, diffDir);  
  expect(result).toBeTruthy();  
});
```

# 執行 Protractor 並載入指定設定檔

- 執行方式 1 - 透過 Angular CLI 的 ng e2e 執行
  - ng e2e `--protractor-config=e2e/protractor-goldens.conf.js`
  - 建議可調整 package.json 的 scripts 區段！
- 執行方式 2 - 透過 npm scripts 執行
  - npm run protractor `-- e2e/protractor-goldens.conf.js`
- 執行方式 3 - 透過 npm 的 [npx](#) 執行 scripts
  - npx protractor `e2e/protractor-goldens.conf.js`
- 執行方式 4 - 透過 protractor 全域工具執行
  - protractor `e2e/protractor-goldens.conf.js`

# 透過 blue-harvest 截圖的注意事項

- 第一次執行的注意事項
  - 先將 `UPDATE_GOLDENS` 設定為 `true`
  - 首次執行後要將產生的 Golden 圖片加入版控
  - 執行完後請移除 `UPDATE_GOLDENS` 環境變數
- 當畫面異動時的注意事項
  - 必須重新產生 Golden 圖片
    - 將 `UPDATE_GOLDENS` 設定為 `true`
    - 再執行一次 `protractor` 重新產生圖片
  - 將更新的 Golden 圖片加入版控
  - 移除 `UPDATE_GOLDENS` 環境變數

# 兩種截圖方式比較

```
element(by.tagName('body')).takeScreenshot()
```

```
browser.takeScreenshot()
```



含 Scrollbar

需額外執行 `brower.waitForAngular();`



無 Scrollbar

會自動等 Angular 穩定



# 動態物件的處理方式

- 遮罩動態物件

```
const e = element(by.id('myImg'));  
await addMask(e, 'gray');
```



# 比較動態物件範例程式

- 截圖之前要遮罩動態物件
  - [addMask](#) 有提供 xOffset, yOffset 調整遮罩位置

```
it('should compare gif lab page', async () => {  
  await browser.get('/labs/gif');  
  const golden = `e2e/goldens/giflab.png`;   
  const diffDir = 'e2e/goldens/';  
  const gif_img = element(by.id('gif-img'));  
  await addMask(gif_img, 'gray');  
  const actual = await browser.takeScreenshot();  
  const result = await  
    compareScreenshot(actual, golden, diffDir);  
  expect(result).toBeTruthy();  
});
```

遮罩動態物件

# 實戰演練：練習畫面呈現測試

- <http://localhost:4200/events>
- 主要程式：**golden.e2e-spec.ts**，**protractor-goldens.conf.js**
- 測試案例
  - 先更新 Golden 圖
  - 比較 <http://localhost:4200/events> 頁面圖
  - 打開 **events-list.component.ts**，故意調整版型
  - 再次比較 <http://localhost:4200/events> 頁面圖
- 練習目標
  - 更新 Golden 圖
  - 熟悉畫面呈現測試

# 實戰演練：練習遮罩動態圖的呈現測試

- <http://localhost:4200/labs/gif>
- 主要程式：**golden-gif.e2e-spec.ts**,  
**protractor-goldens.conf.js**
- 測試案例
  - 遮罩 gif 圖
  - 先更新 Golden 圖
  - 比較 <http://localhost:4200/labs/gif> 頁面圖
- 練習目標
  - 練習遮罩動態畫面

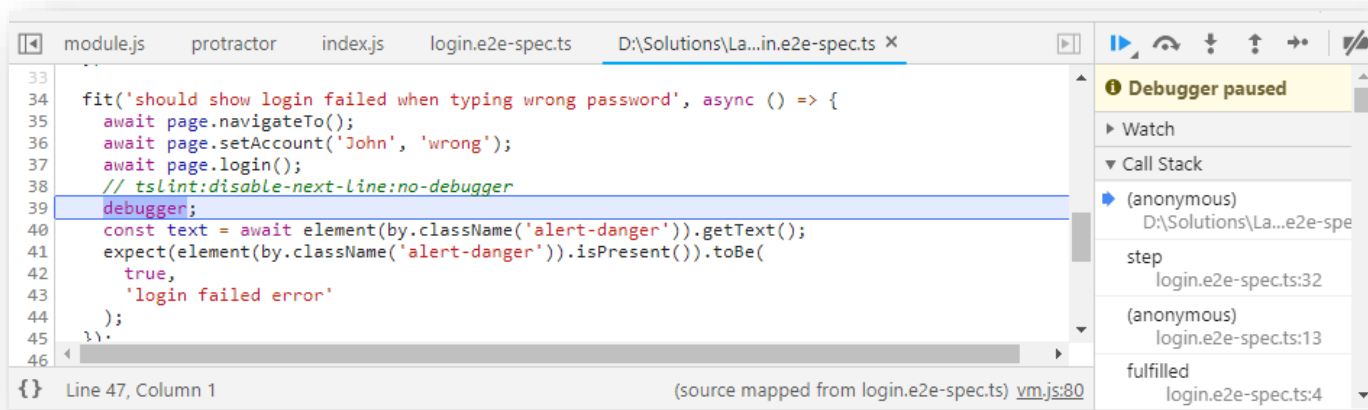
Debugging

# 偵錯技巧

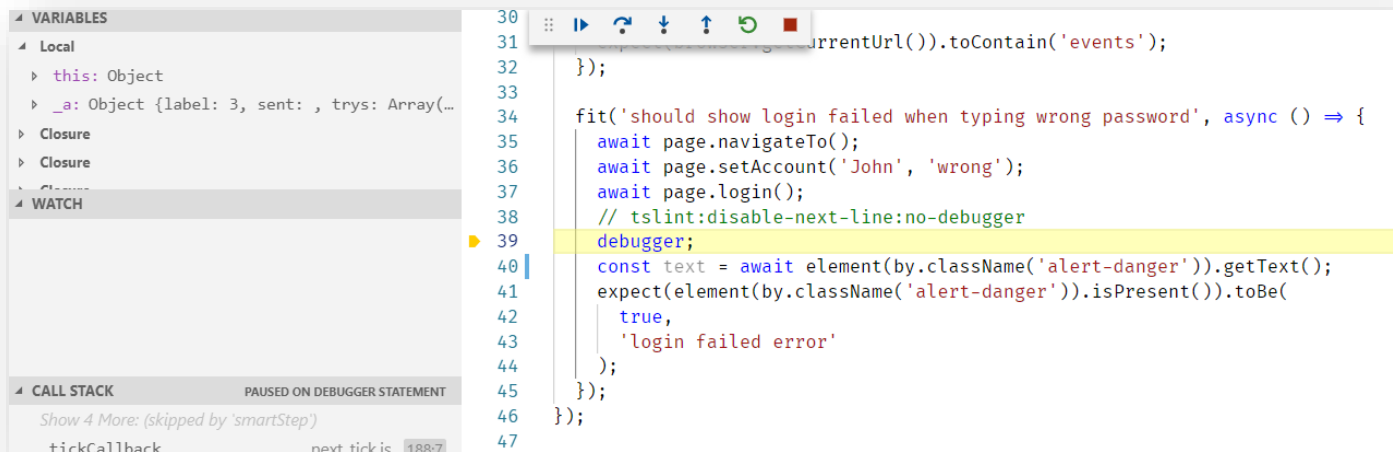


# 兩種偵錯模式

- 使用 Chrome Inspector 偵錯



- 使用 VS Code 偵錯



# 插入中斷點的方式

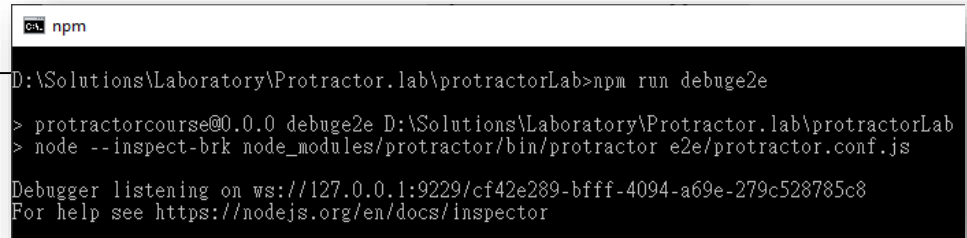
- 只能使用 async/await
  - protractor.conf.js 加入設定  
SELENIUM\_PROMISE\_MANAGER: false
- 於中斷的地方插入 debugger;

```
it('should type in user information', async () => {  
  await username.sendKeys('mike');  
  debugger;  
  await password.sendKeys('123');  
  expect(await username.getAttribute('value')).toBe('mike');  
  expect(await password.getAttribute('value')).toBe('123');  
});
```

# 使用 Chrome Inspector 偵錯 - 1

- 執行命令 ( [參考資料](#) )
  - node --inspect-brk <protractor> <config\_file>
- 設定 package.json 範例
  - npm run debuge2e

```
{  
  "version": "0.0.0",  
  "scripts": {  
    "start": "ng serve",  
    "e2e": "ng e2e",  
    "debuge2e": "node --inspect-brk node_modules/protractor  
/bin/protractor e2e/protractor.conf.js"  
  }  
}
```

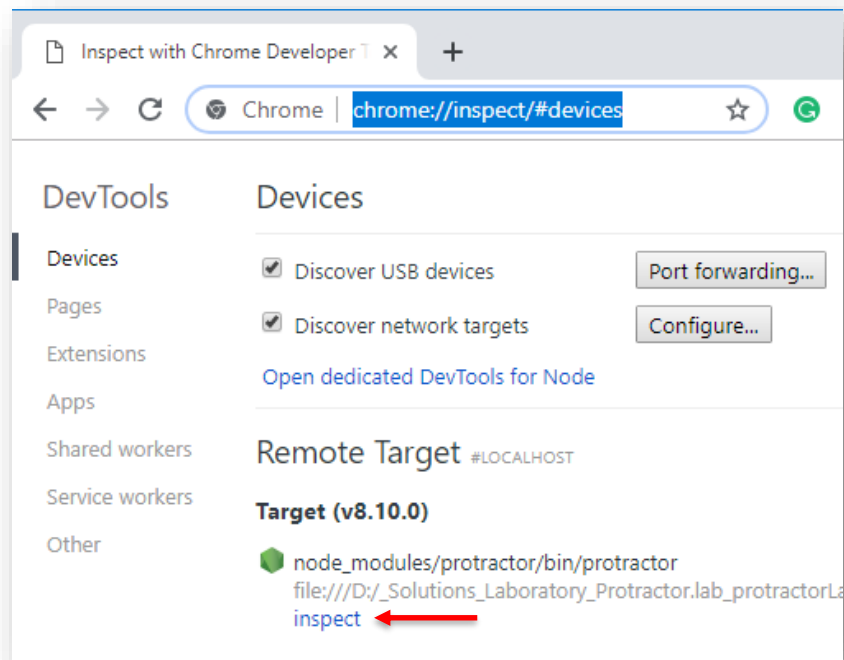


```
npm  
D:\Solutions\Laboratory\Protractor.lab\protractorLab>npm run debuge2e  
> protractorcourse@0.0.0 debuge2e D:\Solutions\Laboratory\Protractor.lab\protractorLab  
> node --inspect-brk node_modules/protractor/bin/protractor e2e/protractor.conf.js  
  
Debugger listening on ws://127.0.0.1:9229/cf42e289-bfff-4094-a69e-279c528785c8  
For help see https://nodejs.org/en/docs/inspector
```



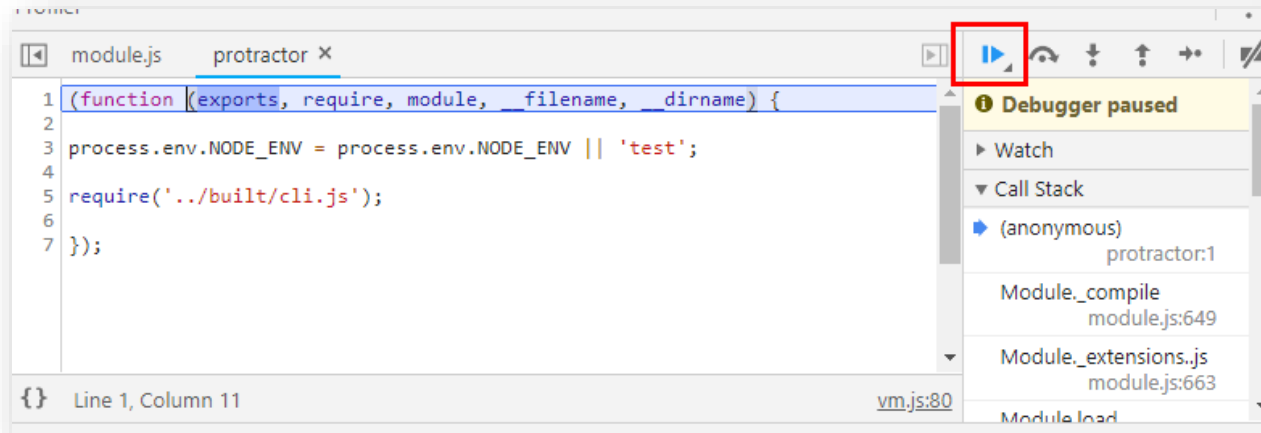
# 使用 Chrome Inspector 偵錯 - 2

1. 開啟 Chrome 瀏覽器
2. 輸入網址 **chrome://inspect/#devices**
3. 點選 **inspect** 啟動程式

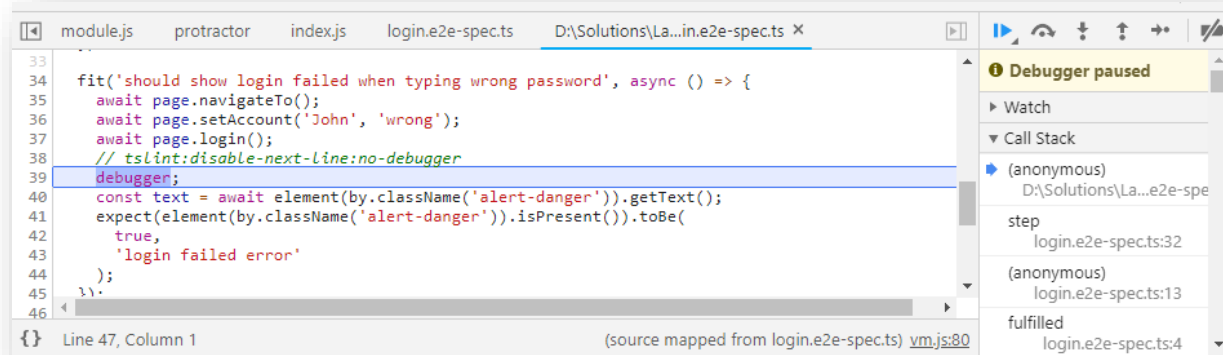


# 使用 Chrome Inspector 偵錯 - 3

- 測試網站會開啟，並一開始會停在起始點  
一點選 **繼續執行** 或 **F8**，然後才會跑新的瀏覽器視窗起來



- 接著會停在 debugger 的位置

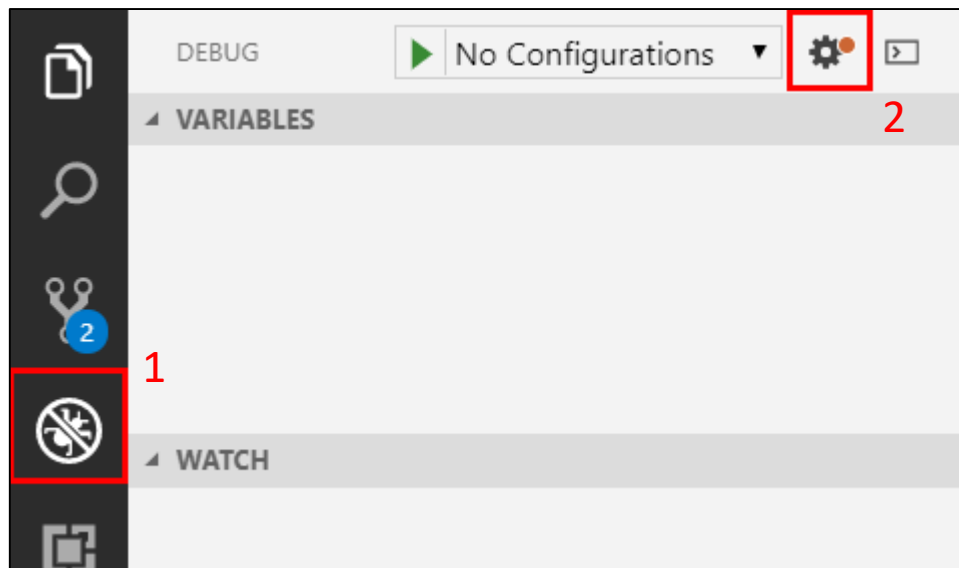


# 實戰演練：使用 Chrome Inspector 偵錯

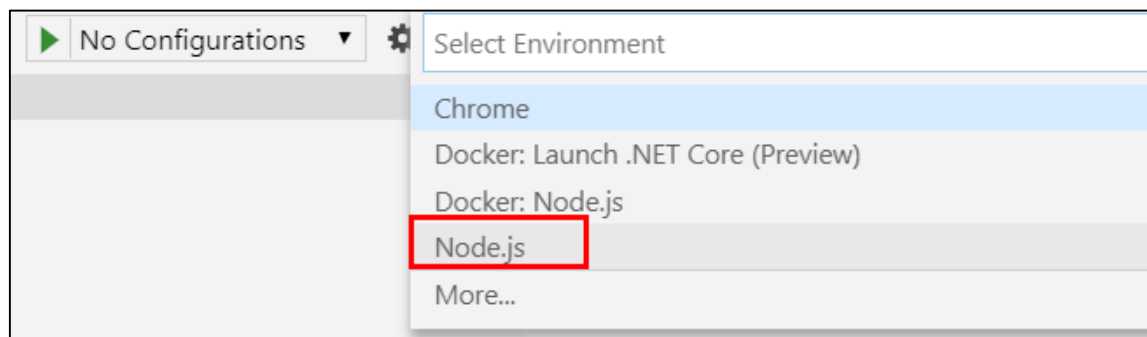
- 主要程式：**\*.e2e-spec.ts**
- 測試案例
  - 找一個 **\*.e2e-spec.ts**
  - 於測試程式碼加入 **debugger;**
  - 執行 chrome inspector 偵錯
- 練習目標
  - 使用 chrome inspector 偵錯
  - 學時使用 chrome inspector 看變數值

# 使用 VS Code 偵錯

- 建立 [launch.json](#)



- 選擇 Node.js



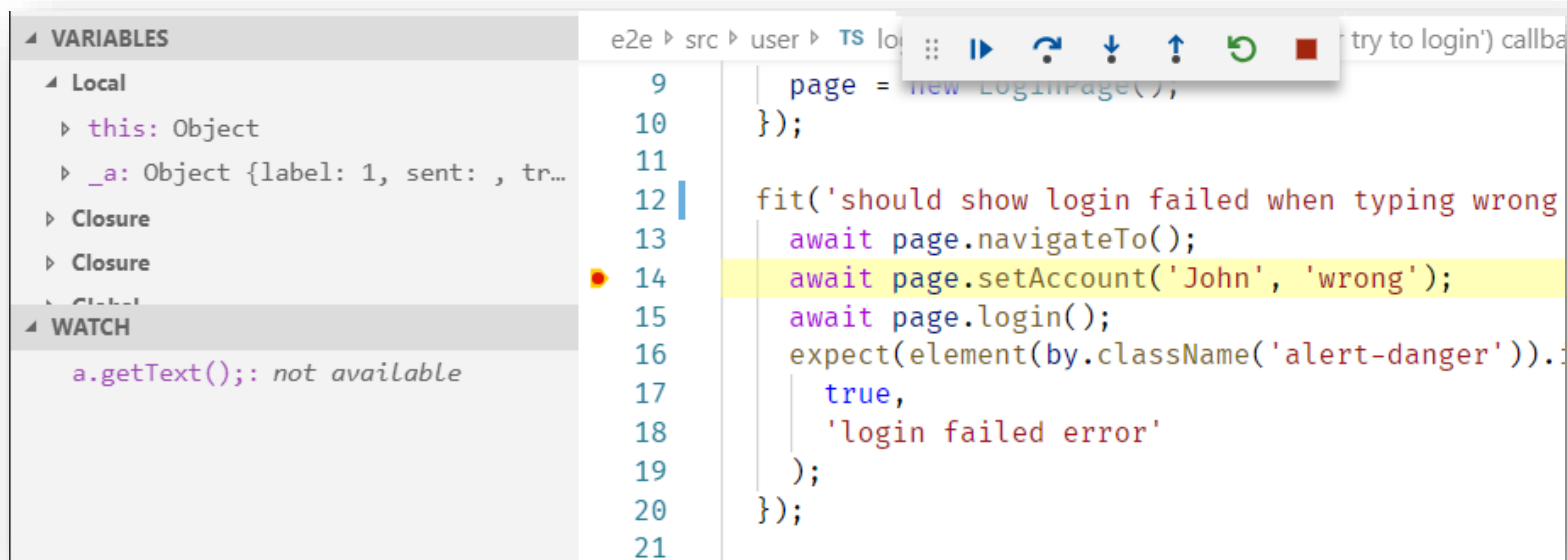
# 使用 VS Code 偵錯

- 複製下方內容到 launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Debug E2E",
      "program":
        "${workspaceRoot}/node_modules/protractor/bin/protractor",
      "args": ["${workspaceRoot}/e2e/protractor.conf.js"]
    }
  ]
}
```

# 使用 VS Code 偵錯

- 執行偵錯 (右圖)
  - 先設定中斷點 (Breakpoints)
- 進入中斷點



# 實戰演練：使用 VS Code 偵錯

- 主要程式：**\*.e2e-spec.ts**
- 測試案例
  - 找一個 \*.e2e-spec.ts
  - 於測試程式碼加入 **debugger;**
  - 執行 VS Code 偵錯
- 練習目標
  - 使用 VS Code 偵錯

# 測試失敗的截圖

- 於 [onPrepare](#) 設定 [jasmine](#) 的事件

```
async onPrepare() {  
  await jasmine.getEnv().addReporter({  
    specDone: async (result) => {  
      if (result.failedExpectations.length > 0) {  
        let png = await browser.takeScreenshot();  
        var stream = require('fs').createWriteStream(  
          './failuretests/failureScreenshot.png');  
        stream.write(new Buffer(png, 'base64'));  
        stream.end();  
      }  
    }  
  });  
}
```



# 實戰演練：自動截圖測試

- 主要程式：**protractor.conf.js**
- 測試案例
  - 於 **protractor.conf.js** 加入測試失敗的截圖程式碼
  - 打開 **create-event.e2e-spec.ts** 調整其中一個 **expect**，使測試失敗
  - 確認 failuretests 資料夾有測試失敗的圖片
- 練習目標
  - 設定 jasmine 事件
  - 設定測試失敗後，自動截圖功能

[Working with Spec and Config Files](#)

# 管理 Protractor 測試案例與設定



# 使用 suites 組織測試計畫

- 設定 **protractor.conf.js** 並建立 [suites](#) 組態

```
suites: {  
  events: './src/events/**/*.e2e-spec.ts',  
  search: [  
    './src/search.e2e-spec.ts',  
    './src/search-by-name.e2e-spec.ts'  
  ]  
}
```

設定 suite 名稱

- 執行 suite 的測試指令
  - protractor protractor.conf.js --suite events
  - protractor protractor.conf.js --suite events,search

# 傳入 Chrome 瀏覽器參數

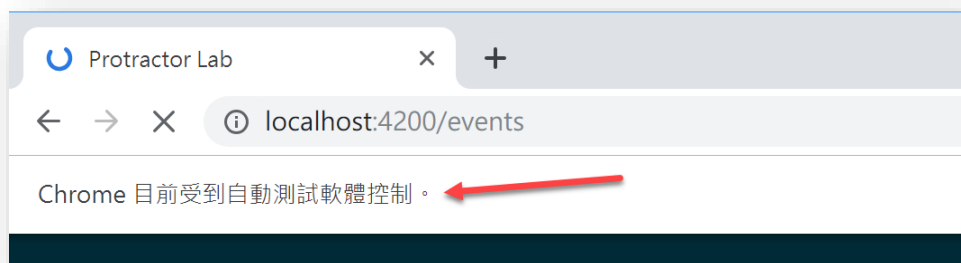
- 設定 **protractor.conf.js** 並啟用 [headless](#) 範例

```
capabilities: {  
  browserName: 'chrome',  
  chromeOptions: {  
    args: [  
      "--headless"  
    ]  
  }  
}
```

- 參考資訊
  - [Capabilities & ChromeOptions - ChromeDriver](#)
  - [List of Chromium Command Line Switches « Peter Beverloo](#)
  - [Where can I find a list of all available ChromeOption arguments?](#)

# 常用 chromeOptions 參數

- `--window-size=800,600` 指定視窗大小
- `--window-position=0,0` 指定視窗位置
- `--disable-popup-blocking` 停用跳出視窗封鎖
- `--proxy-server=localhost:8888` 指定代理伺服器位址
- `--start-maximized` 啟動視窗最大化
- `--ignore-certificate-errors` 忽略憑證錯誤
- `--user-data-dir=G:/Chrome` 指定使用者資料目錄
- `--disable-gpu` 停用 GPU 資源
- `--disable-infobars` 停用資訊列



# 設定瀏覽器

- 切換瀏覽器 (切換至 Firefox 為例)

```
capabilities: {  
  browserName: 'firefox'  
},
```

- 執行多個瀏覽器 (同時執行)

```
multiCapabilities: [  
  { 'browserName': 'firefox' },  
  { 'browserName': 'chrome' }  
]
```

- Protractor 支援的瀏覽器清單
  - <https://www.protractortest.org/#/browser-support>

# timeout 的設定

- `allScriptsTimeout: 60000`
  - 全部測試的執行時間 ( 整份測試計畫所花費最長時間上限 )
  - 單位 ms
- `jasmineNodeOpts: {  
 defaultTimeoutInterval: 30000,  
}`
  - 單一測試的執行時間 ( 每個 it 測試案例的執行時間上限 )
  - 單位 ms

# 建立 HTML Reporter

- 安裝 [protractor-beautiful-reporter](#)
  - `npm install protractor-beautiful-reporter --save-dev`
- `protractor.conf.js` 加入[以下設定](#)

```
const HtmlReporter = require('protractor-beautiful-reporter');
async onPrepare() {
  jasmine.getEnv().addReporter(new HtmlReporter( {
    baseDirectory: 'tmp/screenshots',
    takeScreenShotsOnlyForFailedSpecs: false,
    preserveDirectory: false})).getJasmine2Reporter();
}
```
- 執行測試
  - 打開報告路徑 `tmp/screenshots/report.html`



# 實戰演練：設定 chrome 參數

- 主要程式：**protractor.conf.js**
- 測試案例
  - 調整 **capabilities** 並加入 **chromeOptions**
  - 將 **chromeOptions** 設定為 **--headless** 模式
  - 執行測試
  - 確定測試於 **headless** 模式下執行
- 練習目標
  - 了解 **headless** 模式

# 實戰演練：執行 Firefox 瀏覽器

- 主要程式：**protractor.conf.js**
- 測試案例
  - 調整 **capabilities** 將瀏覽器設定為 **firefox**
  - 執行測試
  - 確定 Firefox 正確執行測試
- 練習目標
  - 設定不同瀏覽器執行測試
- 進階練習
  - 請試著調整設定讓 Firefox 也跑在 headless 模式下
  - <https://www.protractortest.org/#/browser-setup>

# 實戰演練：改用 HtmlReporter 報告

- 主要程式：**protractor.conf.js**
- 測試案例
  - 調整 **onPrepare()** 將 [HtmlReporter](#) 加入 Jasmine
  - 執行測試
  - 查看 HtmlReporter 報告
- 練習目標
  - 學會如何安裝額外的 Reporter 外掛

<https://www.protractortest.org/#/server-setup>

# Selenium Server



# Protractor 操控瀏覽器的三種方式

- 透過 directConnect 直接控制
  - 僅支援 Chrome / Firefox 兩種瀏覽器
- 透過 Selenium Standalone Server 控制
  - `directConnect: false`
  - `seleniumServerJar: '../node_modules/protractor/node_modules/webdriver-manager/selenium/selenium-server-standalone-3.141.59.jar'`
- 透過 Selenium Standalone Server 遠端連接
  - 先啟動 Selenium 伺服器 (下一頁有說明)
  - `directConnect: false`
  - `seleniumAddress: 'http://localhost:4444/wd/hub'`

# 在獨立伺服器執行 Selenium Server

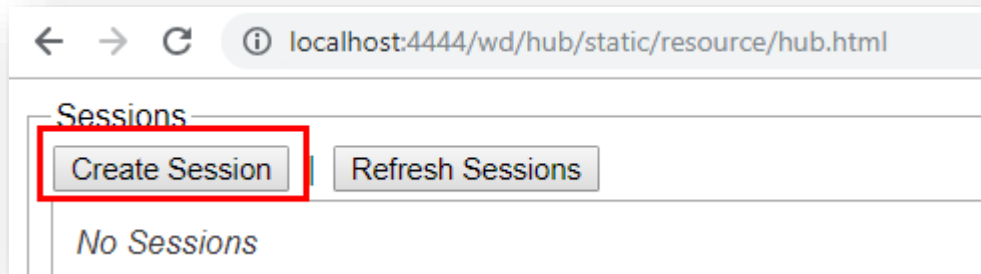
- 安裝想要用來執行 E2E 測試的瀏覽器 (Chrome, Firefox, IE, ...)
- 自動下載與更新 WebDriver 瀏覽器驅動程式
  - **npx webdriver-manager update**
  - **npx webdriver-manager update help** ([完整參數說明](#))
  - 預設全域 WebDriver Manager 的安裝路徑  
`%APPDATA%\npm\node_modules\webdriver-manager\selenium`
- 啟動 Selenium Server
  - **npx webdriver-manager start**
  - `npx webdriver-manager start --seleniumPort=5555`
  - **npx webdriver-manager start help**
- 開啟遠端連接通訊埠的防火牆設定
  - 預設通訊埠 Port **4444**

# Selenium Server 提供的功能

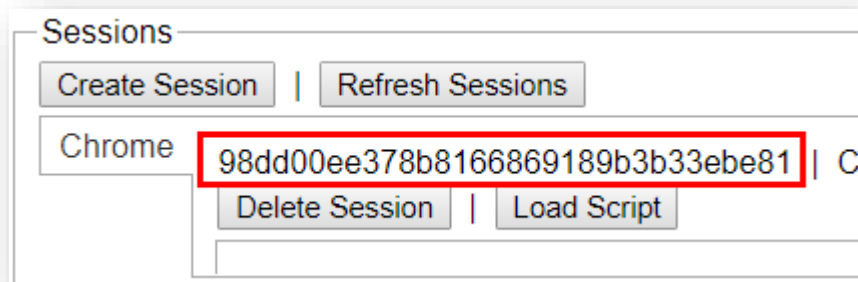
- 透過**獨立主機**執行大量 E2E 測試
  - 可由遠端伺服器上的瀏覽器執行
  - 可以指定遠端伺服器 WebDriver 版本
- 操控沒有支援 **directConnect** 的瀏覽器
  - Safari, IE, Opera, iOS, Edge ( [Microsoft WebDriver](#) )
- 可以啟用 [Blocking Proxy](#)，有助於觀察測試結果
- 可透過 WD Hub 管理瀏覽器工作階段 (Sessions)

# 預先建立工作階段來進行 E2E 測試

- 先開啟 Selenium 管理頁面 <http://localhost:4444/wd/hub>
- 建立新的瀏覽器 Session (此時會先開啟瀏覽器)



- 複製 Selenium Session Id 起來



- 修改 `protractor.conf.js`
  - `seleniumSessionId: '98dd00ee378b8166869189b3b33ebe81'`



# 指定 Chrome WebDriver 版本

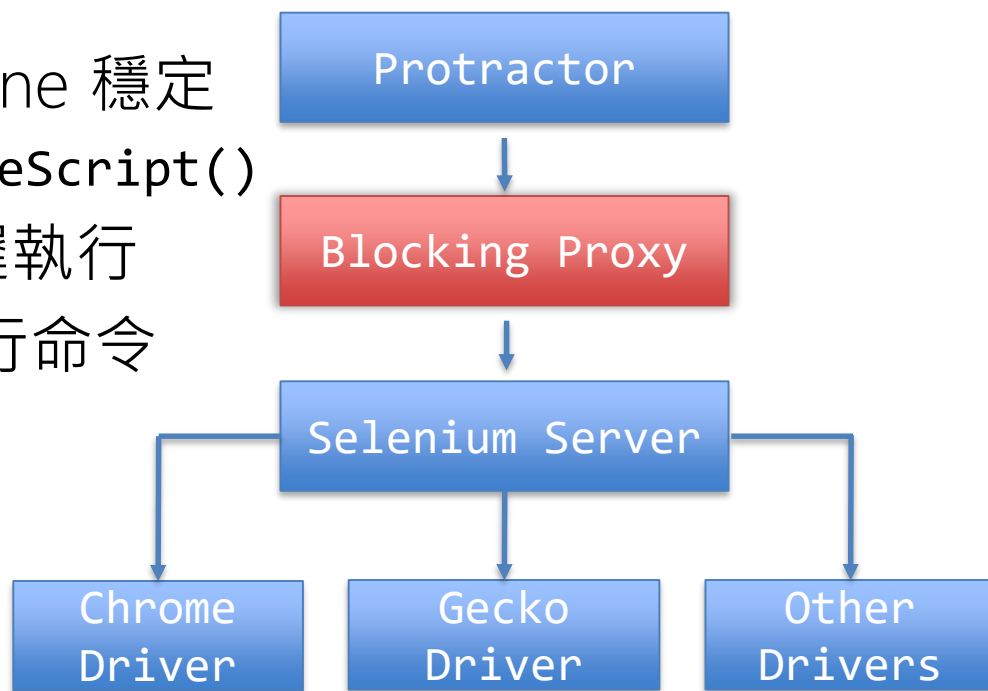
- 選擇 Chrome WebDriver 版本
  - <http://chromedriver.chromium.org/downloads>
- 下載特定 Chrome WebDriver 版本
  - `webdriver-manager update --versions.chrome 2.41`
- 執行指定 Chrome WebDriver 版本
  - `webdriver-manager start --versions.chrome 2.41`

# 使用遠端的 Selenium Server 雲端服務

- BrowserStack
  - <https://www.browserstack.com/>
  - **browserstackUser** 你的 BrowserStack 帳號
  - **browserstackKey** 你的 BrowserStack 金鑰
- Sauce Labs
  - <https://saucelabs.com/>
  - **sauceUser** 你的 Sauce Labs 帳號
  - **sauceKey** 你的 Sauce Labs 金鑰

# Blocking Proxy 簡介

- Protractor 5.1 導入的實驗性功能
  - <https://github.com/angular/protractor/blob/master/CHANGELOG.md#510>
- 提供了三種功能
  - 更多指令會等待 NgZone 穩定
    - 例如 `browser.executeScript()`
  - WebDriver 指令的延遲執行
  - 紀錄 WebDriver 的執行命令



# 啟用 Blocking Proxy

- 調整 `protractor.conf.js` 設定
  - `useBlockingProxy: true`
- 也可直接在 CLI 輸入參數啟用此功能
  - `protractor --useBlockingProxy`
- 啟用後會額外等待 NgZone 穩定的 APIs
  - `browser.executeScript()`
  - `browser.takeScreenshot()`
  - `browser.refresh()`
  - `browser.driver.findElement(by.id('xx'))`
  - `browser.actions().mouseMove({x: 123, y: 456}).click().perform();`

啟用 *Blocking Proxy* 後 `browser.waitForAngularEnabled(false)` 還是有效果

# 啟用 WebDriver 執行紀錄

- 先建立需要儲存 Logs 的資料夾
- 調整 protractor.conf.js 設定
  - `webdriverLogDir: './logs/'`

```
webdriver_log_iqwml.dat.txt
1  16:54:01.798 | 2261ms | 8e35d7 | NewSession
2    {"browserName":"chrome","count":1,"chromeOptions":{"args":["--window-size=1024,768"],"
3  16:54:04.069 | 9ms | 8e35d7 | SetTimeouts
4  16:54:04.689 | 101ms | 8e35d7 | Go data:text/html,<html></html>
5  16:54:04.793 | 1661ms | /session/8e35d7b80de223add4346c2d270416ba/execute
6  16:54:06.458 | 28ms | /session/8e35d7b80de223add4346c2d270416ba/execute
7  16:54:06.489 | 15ms | /session/8e35d7b80de223add4346c2d270416ba/execute_async
8  16:54:06.533 | 23ms | 8e35d7 | Waiting for Angular
9  16:54:06.534 | 20ms | 8e35d7 | FindElements
10    Using css selector '*[name="searchTerm"]'
11    Elements: 0.23913291591640595-1
```

- 也可直接在 CLI 輸入參數啟用此功能
  - `protractor --webdriverLogDir ./logs/`
- 會自動啟用 Blocking Proxy

# 啟用 highlightDelay 延遲每個步驟

- 主要用途
  - 設定瀏覽器執行的每個步驟固定會停留的時間
- 調整 protractor.conf.js 設定
  - `highlightDelay: 2000`
- 也可直接在 CLI 輸入參數啟用此功能
  - `protractor --highlightDelay 2000`
- 會自動啟用 Blocking Proxy

# 使用 IE 瀏覽器進行 E2E 測試 (1)

- 安裝 IE 瀏覽器

- `webdriver-manager update --ie` (Global)
- `npx webdriver-manager update --ie` (Local)

- 設定 `protractor.conf.js`

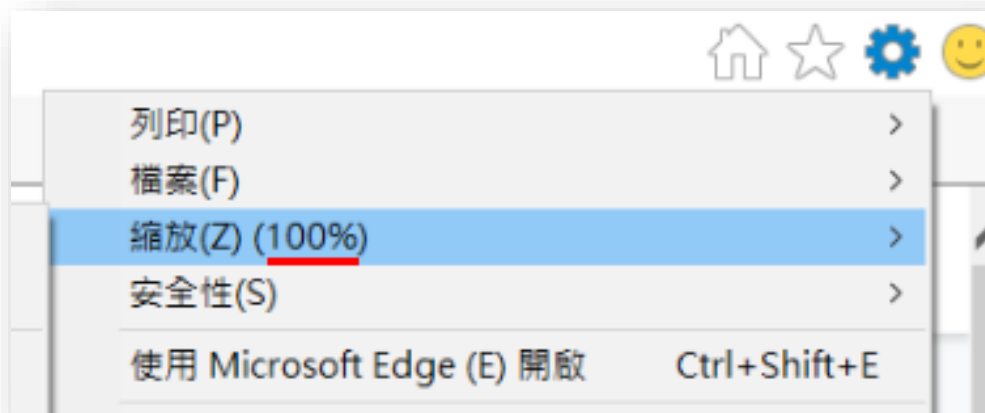
```
capabilities: {  
  browserName: 'internet explorer',  
  version: '11'  
},  
directConnect: false,  
localSeleniumStandaloneOpts: {  
  jvmArgs: ['-Dwebdriver.ie.driver=./node_modules/protractor/node_modules/webdriver-manager/selenium/IEDriverServer3.141.5.exe']},
```

IE 不支援 directConnect

設定 IE WebDriver 路徑

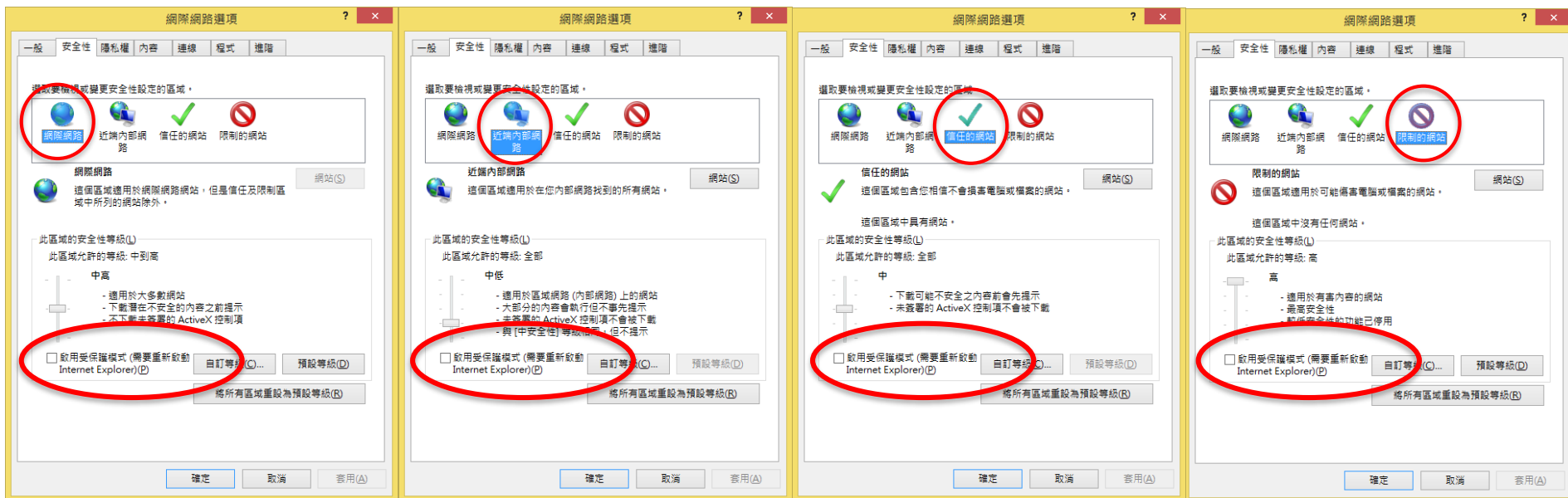
## 使用 IE 瀏覽器進行 E2E 測試 (2)

- 首次使用必須手動調整 IE 的 啟用受保護模式 設定  
工具 -> 網際網路選項 -> 安全性
- 將每個**安全性設定區域**的**啟用受保護模式**設定一致
  - 可設定**全部都啟用**或**全部都停用**都可以，但必須完全一致！
  - Protected Mode exception while launching IE Driver
- 視窗縮放必須設定為 100% (主要針對 High DPI 螢幕)



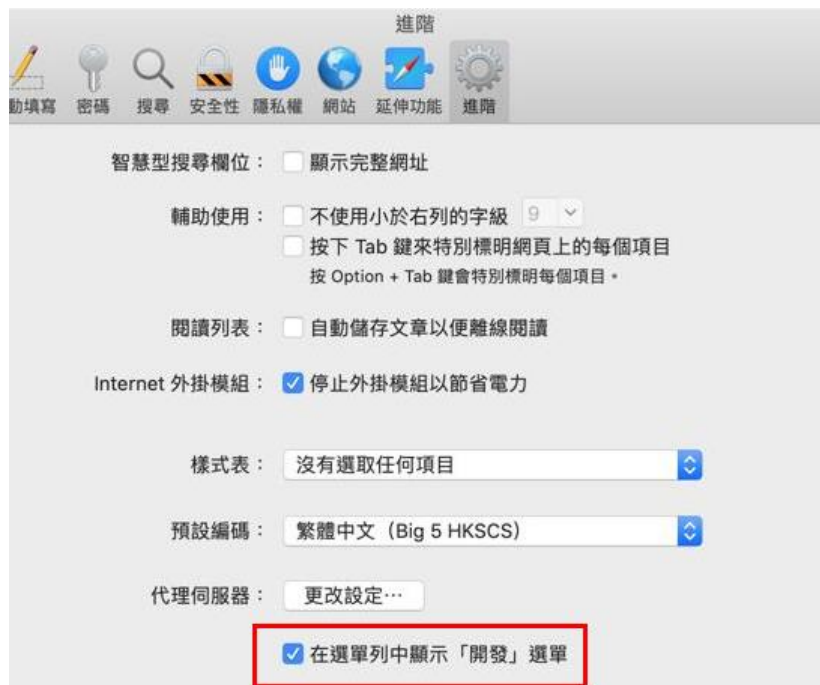


# IE 安全性設定區域設定範例



# 定設執行 Safari 環境

- 啟用 Safari web driver 步驟
  - 輸入指令 **safaridriver --enable**
  - 設定 safari 允許遠端自動化



Cmd+,



# 設定執行 Safari 瀏覽器

- 調整 `protractor.conf.js` 設定

```
capabilities: {  
  browserName: 'safari'  
},  
directConnect: false
```

Page Object

# 設計 Page Object 頁面物件



# 關於 Page Objects 設計概念

- 由於 E2E 測試是站在使用者角度進行測試，因此測試案例若可以站在使用者的角度進行設計與撰寫，將可大幅提高測試程式的可讀性。
- 將畫面取得元素與操作方法封裝到一個獨立的類別中，將可對頁面需求進行封裝，而這就是 [Page Objects](#) 的主要訴求。
- 若畫面微幅調整，只需要異動 Page Object 中的程式碼，而不需要修正原本 E2E 測試流程與程式。

# 以 Login Page 為例的 Page Object

- 建立 Login PageObject ( **login.po.ts** )

```
export class LoginPage {  
  
    async navigateTo() {  
        await browser.get('/');  
    }  
  
    async setAccount(userName: string, password: string) {  
        await element(by.id('userName')).sendKeys(userName);  
        await element(by.id('password')).sendKeys(password);  
    }  
  
    async login() {  
        await element(by.buttonText('Login')).click();  
    }  
}
```

# 改用 Page Object 撰寫測試流程

```
describe('the user try to login', () => {  
  let page: LoginPage;  
  beforeEach(() => {  
    page = new LoginPage();  
  });  
  it('should login to event page', async () => {  
    await page.navigateTo();  
    await page.setAccount('John', '123456');  
    await page.login();  
    expect(await browser.getCurrentUrl()).toContain('events');  
  });  
});
```

建立 Page Object

呼叫 Page Object 方法

# 實戰演練：使用 Page Object 寫測試

- 主要程式：**login.po.ts**, **login.e2e-spec.ts**
- 測試案例
  - 於 login.po.ts 新增 navigateTo 方法
  - 於 login.po.ts 新增 setAccount 方法
  - 於 login.po.ts 新增 login 方法
  - 修改 login.e2e-spec.ts 改用 Page Object 撰寫測試
- 練習目標
  - 了解 Page Object 設計方法



Action Helpers

# 使用 Action Helpers 撰寫測試



# 關於 Action Helpers 設計概念

- 這些 API 來自於 [blue-harvest](#) 套件
- 透過直觀的 API 大幅降低撰寫 E2E 測試的時間
  - 所有 Action Helpers 都內建[自動等待](#)機制 (也會錯誤重試)
- 原作者在 ng-conf 2018 [演講影片](#)中表示，撰寫 Page Objects 佔用了大量撰寫測試的時間，但是頁面又經常異動，導致維護成本增加，因此設計出 Action Helpers 輔助方法，加快開發人員撰寫 E2E 測試！

# 補助方法 APIs 說明 ( [文件都在原始碼中](#) )

APIs	說明
<a href="#">go(url: string)</a>	導向指定頁面 <code>await go('/firing');</code>
<a href="#">click(elem: FlexibleLocator)</a>	點擊元件 <code>await click('姓氏:');</code> <code>await click(by.name('lastName'));</code>
<a href="#">type(text: string)</a>	文字輸入框輸入文字 <code>await type('Coco');</code>
<a href="#">see(elem: FlexibleLocator)</a>	驗證畫面上有看到指定的元件或文字 <code>await see('送出成功');</code> <code>await see(by.className('alert'));</code>

[FlexibleLocator = string|RegExp|Locator](#)

# Action Helpers 補助方法使用範例



```
it('should fire a form', async () => {  
  await go('/labs/fireform');  
  await click('姓氏:');  
  await type('王');  
  await click('名字:');  
  await type('小明');  
  await click('送出');  
  await see('送出成功');  
});
```

# 實戰演練：練習 Action Helpers 輔助方法

- <http://localhost:4200/labs/fireform>
- 主要程式：**fireform.e2e-spec.ts**
- 測試案例
  - 點擊 姓氏 並輸入 **John**
  - 點擊 名字 並輸入 **Bob**
  - 點擊 送出
  - 驗證畫面上出現 **送出成功**
- 練習目標
  - 熟悉 Action Helpers 輔助方法的使用方式

<https://www.protractortest.org/#/style-guide>

# Protractor 測試撰寫準則



# 撰寫 Protractor 的撰寫風格建議 (1)

- 不要測**單元測試**就能測的案例
  - 執行單元測試快多了，避免重複的測試
- 不同的 `*.e2e-spec.ts` 檔案之間必須相互獨立
  - 避免不同的測試群組有相依性，降低 side-effect 發生
- 不要在你的 E2E 測試加上判斷邏輯
  - 像是 `if`, `for`, `switch`, ... 都不應該使用
- 不要再你的 E2E 測試中套用任何 Mock 技巧
  - 你應該使用真實的系統進行 E2E 測試
  - 套用 Mock 技巧可能會讓你錯失發現特定罕見的錯誤
- 建議使用 Jasmine 2 版本 (不要用最新版本)
  - 因為 Protractor 官方強烈建議使用 Jasmine 2 版本

# 撰寫 Protractor 的撰寫風格建議 (2)

- 特別建立一組 suite 瀏覽全網站重要的所有頁面
  - 一般來說測試人員並不會人工瀏覽所有頁面
  - 直接連結全網站幾個最重要的網頁是否可以正常連線
  - 有授權限制的頁面可以透過這組測試確認權限正確！
- 永遠不要用 `by.xpath` 當作 Locator 搜尋 DOM
  - 因為 HTML 結構經常改變，維護成本會很高
  - 建議多用 `by.id`、`by.css`、`by.name` 進行 DOM 搜尋
  - 適量使用 `by.linkText`、`by.buttonText`、`by.cssContainingText` 透過文字搜尋 DOM 物件 (因為文字較常異動)
- 測試案例的目錄結構盡量與應用程式的目錄一致
  - 測試案例必須與真實需求、真實網站架構一致，直覺明瞭為優先
  - E2E 測試與單元測試應該明確分開，因為測試目標不太一樣



# 聯絡資訊

- The Will Will Web

記載著 Will 在網路世界的學習心得與技術分享

- <http://blog.miniasp.com/>

- Will 保哥的技術交流中心 (臉書粉絲專頁)

- <http://www.facebook.com/will.fans>

- Will 保哥的推特

- [https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)