技術筆記

React.js

筆記更新紀錄

|  |  |  |
| --- | --- | --- |
| 日期 yyyy/MM/dd | 更新人員 | 更新備註 |
| 2019/08/21 | Miles | 建立文件 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

目錄

[1 關於Web前端 4](#_Toc18487408)

[2 React 5](#_Toc18487409)

## 1 關於Web前端

|  |
| --- |
|  |
| 簡介: |
| 事實上，在目前的業界，前端工程的定位光譜非常廣泛，有聚焦在網頁設計（Web Design），也有專注在軟體工程（Software Engineering）的部份，本書則是將前端工程定位在軟體工程的範疇。而 HTML、CSS 和 JavaScript 是前端工程最重要的技術基礎。過去一段時間，我們所認為的前端工程主要專注在瀏覽器平台，但現在的 Web 平台已經不再侷限於桌面瀏覽器，而是必須面對更多的跨平台、跨瀏覽器的應用開發場景，其中包含：   1. 網頁瀏覽器（Web Browser），一般的網頁應用程式開發 2. 透過 CLI 指令去操作的 Headless 瀏覽器（Headless　Application）。例如：[phantomJS](http://phantomjs.org/)、[CasperJS](http://casperjs.org/) 等 3. 運作在 WebView 瀏覽器核心（WebView Application）的應用。例如：[Apache Cordova](https://cordova.apache.org/)、[Electron](http://electron.atom.io/)、[NW.js](http://nwjs.io/) 等行動、桌面應用程式開發 4. 原生應用程式（Native Application），透過 Web 技術撰寫原生應用程式。例如：[React Native](https://facebook.github.io/react-native/)、[Native Script](https://www.nativescript.org/) 等   過去幾年，前端開發就像經歷了文藝復興（Rinascimento）的年代，開始了各種框架、套件百花齊放的時代。雖然現在有更多好用工具可以協助開發，但前端工程師似乎並沒有變得比較輕鬆。以往若能妥善運用 jQuery 等函式庫就可以應付大部分前端工程師的工作，但現在前端徵才廣告上不僅要求精通 HTML、CSS 和 JavaScript，還要對於還要對於 [Backbone](http://backbonejs.org/)、[Ember](http://emberjs.com/)、[Angular](https://angularjs.org/)、[React](https://facebook.github.io/react/)、[Vue](https://vuejs.org/) 等 JavaScript 框架或函式庫有一定程度的了解。  在眾多 JavaScript 框架或函式庫中，[React](https://facebook.github.io/react/) 是 Facebook 推出的開源 [JavaScript](https://en.wikipedia.org/wiki/JavaScript) Library，它的出現讓許多革新性的 Web 觀念開始流行起來，例如：Virtual DOM、Web Component、更直覺的宣告式 UI 設計、更優雅地實現 Server Rendering 等。接下來本書將透過介紹 React 生態系（ecosystem）帶領讀者入門 React 的世界，讓讀者可以從零開始真的動手用 React 開發跨平台應用程式。 |

## 2 React 生態環境

|  |
| --- |
|  |
| 簡介 |
| React 是一個專注於 UI（View）的 JavaScript 函式庫（Library）。  ReactJS 是 Facebook 推出的 JavaScript 函式庫，若以 MVC 框架來看，React 定位是在 View 的範疇。在 ReactJS 0.14 版之後，ReactJS 更把原先處理 DOM 的部分獨立出去（react-dom），讓 ReactJS 核心更單純，也更符合 React 所倡導的 “Learn once, write everywhere” 的理念。事實上，ReactJS 本身的 API 相對單純，但由於整個生態系非常龐大，因此學習 React 卻是一條漫長的道路。此外，當你想把 React 應用在你的應用程式時，你通常必須學習整個 React Stack 才能充分發揮 React 的最大優勢。 |
| ReactDom(react-dom) |
|  |
| Learn once, write everywhere |
| 只需學習一次，到處皆可實作 |
|  |

## 3 JSX

|  |
| --- |
|  |
| 簡介 |
| 事實上，JSX 並非一種全新的語言，而是一種語法糖（[Syntatic Sugar](https://en.wikipedia.org/wiki/Syntactic_sugar)），一種語法類似 [XML](https://zh.wikipedia.org/wiki/XML) 的 ECMAScript 語法擴充。在 JSX 中 HTML 和組建這些元素標籤的程式碼有緊密的關係，這和過去我們強調 HTML、JavaScript 分離的觀念有很大不同。當然，你可以選擇不要在 React 使用 JSX，不過相信我，當你真正開始撰寫 React 元件（Component）時，你會很慶幸有 JSX 真好。 |
| ECMAScript |
| ECMAScript是一種由[Ecma國際](https://zh.wikipedia.org/wiki/Ecma%E5%9B%BD%E9%99%85)（前身為[歐洲電腦製造商協會](https://zh.wikipedia.org/wiki/%E6%AC%A7%E6%B4%B2%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%88%B6%E9%80%A0%E5%95%86%E5%8D%8F%E4%BC%9A)）在標準ECMA-262中定義的手稿語言規範。這種語言在[全球資訊網](https://zh.wikipedia.org/wiki/%E4%B8%87%E7%BB%B4%E7%BD%91)上應用廣泛，它往往被稱為[JavaScript](https://zh.wikipedia.org/wiki/JavaScript)或[JScript](https://zh.wikipedia.org/wiki/JScript)，但實際上後兩者是ECMA-262標準的實現和擴充。 |

## 4 NPM

|  |
| --- |
|  |
| 簡介 |
| NPM（Node Package Manager）是 Node.js 下的主流套件管理工具。在 NPM 上有非常多的套件，可以讓你不用再重造輪子，更可以讓你可以輕鬆用指令管理不同的套件。由於 NPM 主要是基於 [CommonJS](https://en.wikipedia.org/wiki/CommonJS) 的規範，通常必須搭配 Browserify 這樣的工具才能在前端使用 NPM 的模組。然而因 NPM 是基於 Nested Dependency Tree，不同的套件有可能會在引入依賴時會引入相同但不同版本的套件，造成檔案大小過大的情形。這和另一個套件管理工具 [Bower](https://bower.io/) 專注在前端套件且使用 Flat Dependency Tree（讓使用者決定相依的套件版本）是比較不同的地方。 |

## 5 ES6+

|  |
| --- |
|  |
| 簡介 |
| [ES6+](https://babeljs.io/blog/2015/06/07/react-on-es6-plus) 係指 ES6（ES2015）和 ES7 的聯集，在 ES6+ 新的標準當中引入許多新的特性和功能，彌補了過去 JavaScript 被詬病的一些特性。由於未來 React 將以支援 ES6+ 為主，因此直接學習 ES6+ 用法是相對好的選擇，本書的所有範例也將會以 ES6+ 撰寫。 |

## 6 Babel

|  |
| --- |
|  |
| 簡介 |
| 由於並非所有瀏覽器都支援 ES6+ 語法，所以透過 [Babel](https://babeljs.io/) 這個 JavaScript 編譯器（可以想成是翻譯機或是翻譯蒟篛）可以讓你的 ES6+ 、JSX 等程式碼轉換成瀏覽器可以看得懂的語法。通常會在資料夾的 root 位置加入 .babelrc 進行轉譯規則 preset 和引用外掛（plugin）的設定。 |

## 7 JavaScript模組化開發

|  |
| --- |
|  |
| 簡介 |
| 隨著 Web 應用程式的複雜性提高，JavaScript 模組化開發已經成為必然的趨勢，以下簡單介紹 JavaScript 模組化的相關規範。事實上，在一開始沒有官方定義的標準時出現了各種社群自行定義的規範和實踐。 |
| CDN-Based |
| 也就是最傳統的 <script> 引入方式，然而使用這種方式雖然簡單方便，但在開發實際中大型應用程式時會產生許多弊端：   1. 全域作用域容易造成變數污染和衝突 2. 文件只能依照 <script> 順序載入，不具彈性 3. 在大型專案中各種資源和版本難以維護 4. 必須由開發者自行判斷模組和函式庫之間的依賴關係 |
| AMD |
| [Asynchronous Module Definition](https://en.wikipedia.org/wiki/Asynchronous_module_definition) 簡稱 AMD，為非同步載入模組的規範，其在宣告時模組時即需定義依賴的模組。AMD 常用於瀏覽器端，其最著名的實踐為 [RequireJS](http://requirejs.org/)  基本格式：define(id?, dependencies?, factory); |
| CMD |
| CMD 全稱為 [Common Module Definition](https://github.com/cmdjs/specification/blob/master/draft/module.md)，其規範和 AMD 類似，但相對簡潔，卻又保持和 CommonJS 的兼容性。其最大特色為：依賴就近，延遲執行。主要實現為：[Sea.js](http://seajs.org/docs/#intro)。 |
| UMD |
| [Universal Module Definition](https://github.com/umdjs/umd) 是為了要兼容 CommonJS 和 AMD 所設計的規範，希望讓模組能跨平台執行。 |
| ES6 Module |
| ECMAScript6 的標準中定義了 JavaScript 的模組化方式，讓 JavaScript 在開發大型複雜應用程式時上更為方便且易於管理，亦可以取代過去 AMD、CommonJS 等規範，成為通用於瀏覽器端和伺服器端的模組化解決方案。但目前瀏覽器和 Node 在 ES6 模組支援度還不完整，大部分情況需要透過 [Babel](https://babeljs.io/) 轉譯器進行轉譯 |