從零到有簡易建立RAECT + WEBPACK + BABEL開發環境

開始之前先確保你已經安裝了NODEJS，不然無法執行NPM指令；另外建議用VS-CODE作為預設編輯器，教學亦會使用VS-CODE做示範，不過這兩部份不會在這裡作詳細說明

這邊先介紹一下NPM指令，一般語法為『npm install software-name --save-dev』

-『npm』沒什麼好說的，就係指令名稱

-『install』就是安裝的意思，亦可以用縮寫『i』代替；有安裝就有反安裝，參數為『uninstall』，但是就不可以用『u』當作縮寫了，更多其他參數就不作說明了，有需要可自行上網搜尋

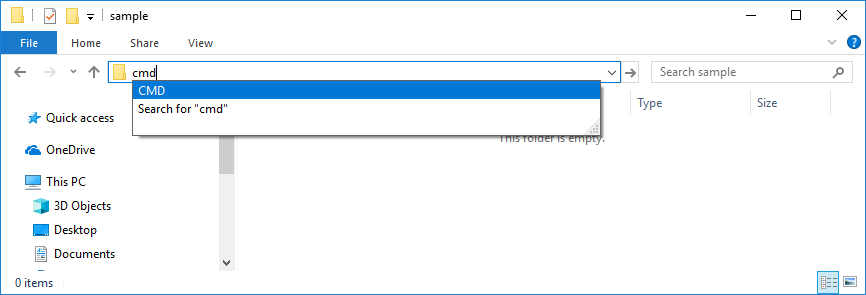
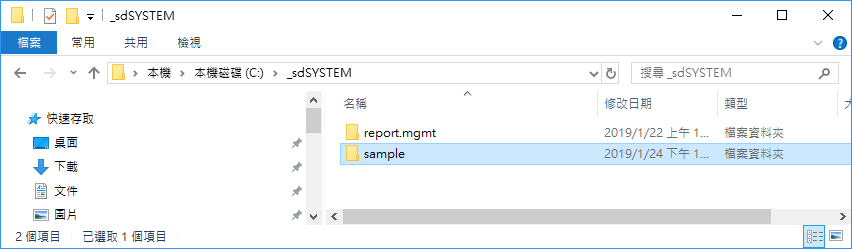
-『software-name』就是要安裝既插件名稱，它支持一次安裝多個插件，用空白字符隔開名稱即可，同時可以用『@verison-number』的方式安裝指定版本號的插件，如果不指明版本號就會安裝默認版；下面會列出教學所用到的插件版本，如果安裝默認版本後無法順利執行，可以在執行安裝指令時加入版本號安裝指定版本，如『npm i babel-loader@7.1.5 --save-dev』

-『--save-dev』這部份常見的參數為『-save』和『--save-dev』，它的縮寫分別為『-S』和『-D』，作用是會自動修改『package.json』檔案內容，這裡先不詳細說明這檔案的作用，就是聲明專案是依賴那些插件，然而它是分為production和development下的環境依賴。舉例來說，BABEL這個插件在開發途中是不可缺少的，因為會用到它來進行編譯動作，但當完成專案後的發佈就不再需要它了，所以它只是個development下的環境依賴，安裝時就只需要用『-D』就是；發佈後還需要的就要『-S』好了，不然別人引用發佈後的專案就會出現找不到依賴的報錯了，所以最後是用『-S』還是『-D』請自行決定，教學內全都是用『-D』就是為了方便而已

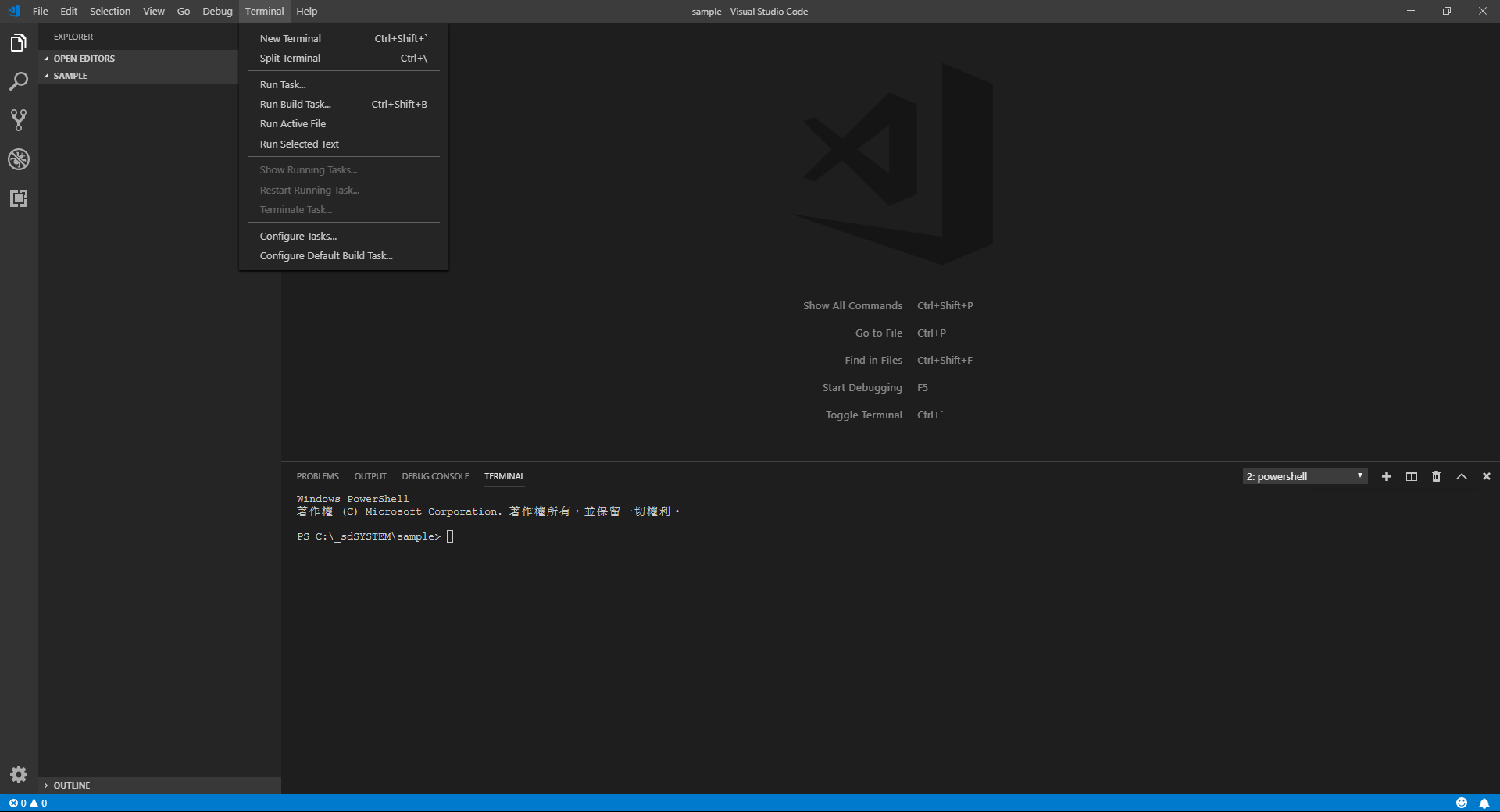
環境配置  
webpack: 4.28.3  
webpack-cli: 3.2.0  
babel-core: 6.26.3  
babel-loader: 7.1.5  
babel-preset-env: 1.7.0  
babel-preset-react: 6.24.1

建立一個資料夾名為『sample』並進入該資料夾

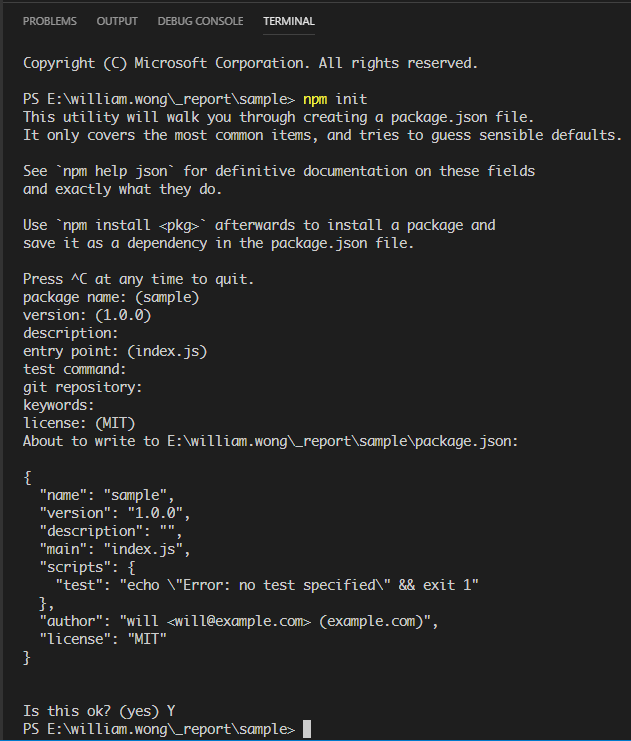
在檔案路徑位置直接輸入『cmd』並按下『enter鍵』可以快速打開cmd視窗及跳轉到當前資料夾位置；如果你並不打算使用VS-CODE，接下來所有指令可在cmd視窗執行



執行指令『code .』打開VS-CODE，然後按下『ctrl + `』跳轉到『terminal』



執行指令『npm init』產生檔案package.json；如有需要可以自行設定相關資訊，不然全部留空即可；VERSION格式必須為x.x.x，否則會報錯



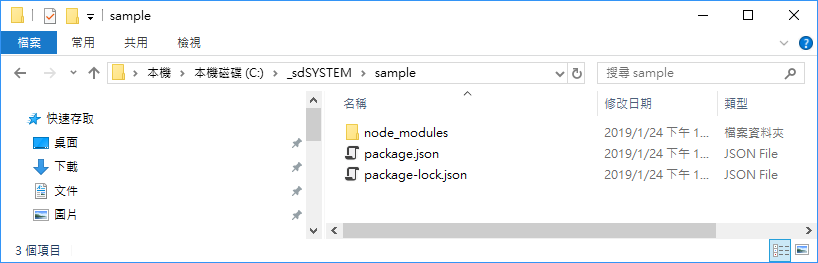
逐一執行以下指令

安裝WEBPACK插件『npm i webpack webpack-cli -D』

安裝REACT插件，『npm i react react-dom -D』

安裝BABEL插件，『npm i babel-core babel-loader@7 babel-preset-env babel-preset-react -D』； 這裹是指定安裝babel-loader version 7是為了對應babel-core會默認安裝VERSION 6的要求，將來可不再適用

完成了安裝主要插件部份後，資料夾的內容如下，接下來需要做一些檔案設定



修改『package.json』，主要是scripts部份，內容如下

這邊的修改主要對應之後compile時執行的指令，在之後的開發環境下我們需要執行指令把我們的source code封裝成對應的html等檔案，主要是執行指令『npm run start』，而當中start的字眼就是藍色的start，然後執行後部份的值，所以這裹是可以自己修改或添加

"scripts": {



"start": "webpack --mode development",

"build": "webpack --mode production"

},

新增檔案『webpack.config.js』，內容如下

不作說明，目的就是讓webpack封裝時知道要找『.js』檔

module.exports = {



module: {

rules: [

{

test: /\.js$/,

exclude: /node\_modules/,

use: {

loader: "babel-loader"

}

}

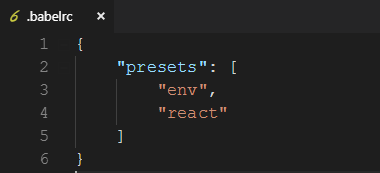
]

}

};

新增檔案『.babelrc』，內容如下

這個檔案也是無法說明，但這裹有個小技巧，如要在window下建立此檔案，可以新增一個文本檔並把他命名為 『.babelrc.』即可，否則無法名命只有副檔名檔案



{

"presets": [

"env",

"react"

]

}

安裝打包HTML插件html-webpack-plugin，『npm i html-webpack-plugin -D』

修改檔案『webpack.config.js』，內容如下

這裹的動作是引入插件，然後設定資源及目的地位置來生成html檔案

const HtmlWebPackPlugin = require("html-webpack-plugin");



… …

plugins: [

new HtmlWebPackPlugin({

template: "./src/index.html",

filename: "./index.html"

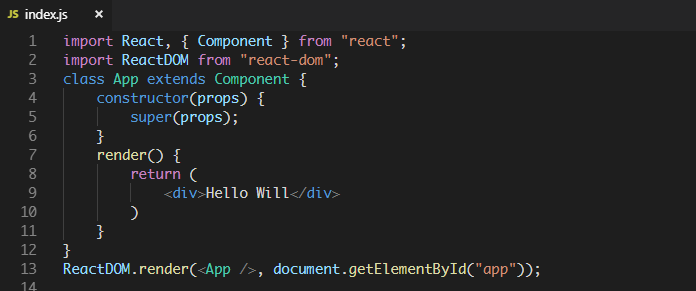
})

]

到此，我們可以正式使用我們設置的開發環境來生成網頁，但這裹暫時也不會就react語法作出任何說明

新增一個資料夾名為『src』並在它底下新增檔案『index.js』，內容如下

import React, { Component } from "react";



import ReactDOM from "react-dom";

class App extends Component {

constructor(props) {

super(props);

}

render() {

return (

<div>Hello Will</div>

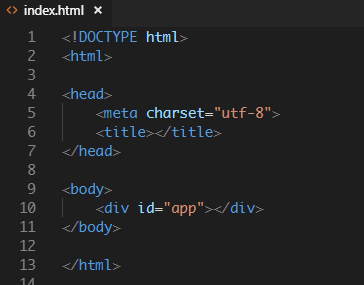
)

}

}

ReactDOM.render(<App />, document.getElementById("app"));

在同一位置下新增檔案『index.html』，內容如下



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title></title>

</head>

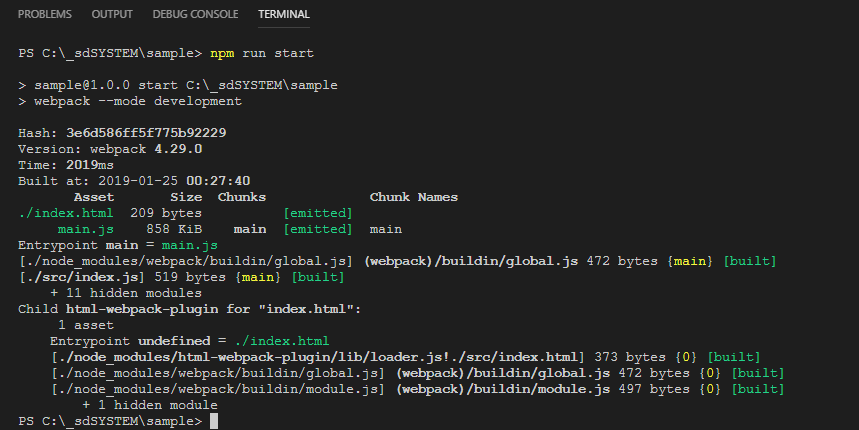
<body>

<div id="app"></div>

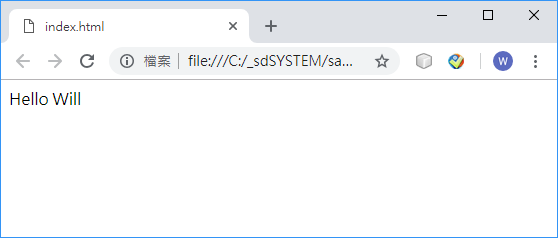
</body>

</html>

然後我們在terminal那邊執行指令『npm run start』把index.js及index.html封裝成html頁面



在『sample』的資料夾下我們會發現多了一個名為『dist』的資料夾，這是webpack打包的預設位置，點進去後可以打開index.html就是我們剛剛製作的網頁



接下來我們還需要為了CSS、圖片、文字等額外資料安裝插件，使webpack打包時懂得這些檔案是什麼，教學內容還會使用LESS及JSX檔案，如果你並不打算使用它們的話，你可以因應需要作出調整

安裝less-loader插件，『npm i less less-loader style-loader css-loader -D』

安裝url-loader插件，『npm i url-loader -D』

安裝file-loader插件，『npm i file-loader -D』

安裝打包CSS插件extract-text-webpack-plugin，『npm i extract-text-webpack-plugin -D』

當所有插件都安裝完成的時候，然後要介紹比較麻煩的webpack setting內容

先以安裝完html-webpack-plugin為例，我們是在頂部引用了它，然後在『plugins』的部份使用它來宣告怎樣打包html的source file到目的地

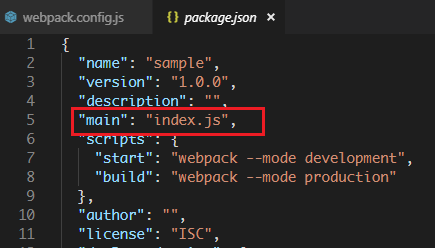


它的屬性『template』就是對應html的外殼，一般來講只要這個檔案只要有一個『div』的元素含一個屬性『id』可以讓reactdom找到就可以，所以我們大可以把之後所有的page都指向同一個tamplate，除非你自己有特別的需要



它的屬性『filename』就是對應生成到目的地的名稱，都可以隨便命名

到這步你會發現我們由頭尾都沒有提及另外一個檔案『index.js』，這個才是page的真正所容，那不提及它為什麼最後又能使用到它的呢，這點我沒有深入研究，就直覺讓我覺得是跟package.json內容有關，就是main所指向的index.js，它應該是當你沒有指定檔案時就默認這個main部份



但是react的規劃就是要把東西細分的零零碎碎，只靠默認部份是肯定沒可能完成我們想要做的事情，所以它的設定是相當相當的有彈性及複雜

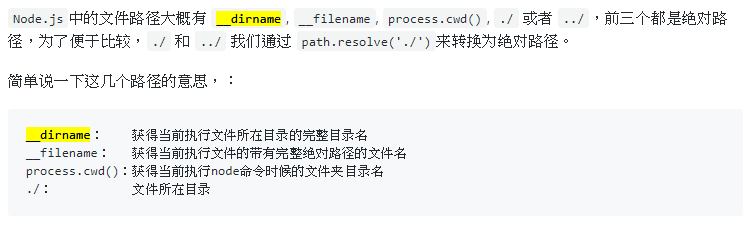
到這部份先讓大家看一下之後會用到的部份，但這並不代表它的全部，是我不清楚而且，反正在教學不會用到就別要求太多了，先說明一下下面4個部份：



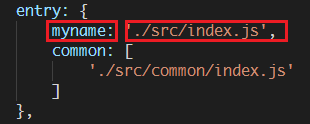
修改『webpack.config.js』檔案內容如下:

const path = require('path');  
const HtmlWebPackPlugin = require("html-webpack-plugin");  
  
module.exports = {  
 entry: {  
 myname: './src/index.js',  
 common: [  
 './src/common/index.js'  
 ]  
 },  
 output: {  
 path: path.resolve(\_\_dirname, 'dist'),  
 filename: 'resource/js/[name]/index.js'  
 },  
 module: {  
 rules: [  
 {  
 test: /\.js$/,  
 exclude: /node\_modules/,  
 use: {  
 loader: "babel-loader"  
 }  
 }  
 ]  
 },  
 plugins: [  
 new HtmlWebPackPlugin({  
 template: "./src/index.html",  
 filename: "./index.html",  
 title: '',  
 favicon: '',  
 chunks: ['myname', 'common']  
 })  
 ]  
};

第1點是引用path的功能，它是用來組成專案檔案的路徑，不過此教學也沒多用就是了，最後就在網上找一些說明補充來輕輕帶過吧

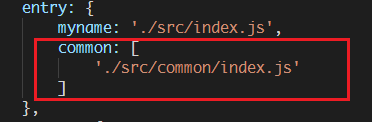


第2點entry部份，這個是一個頁面對應的主要js位置，你會看到我們會給它一個變數對應檔案的path，這個名字是很重要的，將會在webpack打包的時候成為我們用來架構網站位置的名稱，就是目的地folder名稱

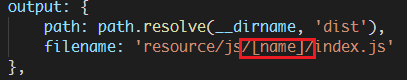


補充一點，這個變數還可以以陣列形式表示，最後實現的結果就是把所有的js都加載到一個檔案去，通常用於共用檔案的佈置

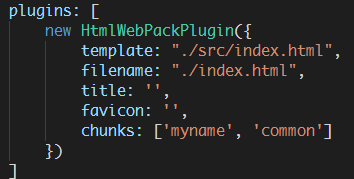
其實共用的js並不會以這個方式實現，共用檔案一般多為字體圖片等資源，這邊是為了教學說明才有這個檔案，所以我們先在src資料夾底下新增一個common資料夾，然後新一個空白的index.js即可



第3點output，path的屬性跟path插件的配合就是用來宣告發佈檔案的位置，filename就是檔案的名字，你會看到路徑上出現[name]的字眼，這個就是第2點提及的變數了，當然webpack打包時還可以用共他參數，這部份就不說明了



第4點就是plugin中的htmlwebpackplugin，這個部份能夠聲明打包後的html內容



title就是<title />部份，不過好像是有bug而不能正常使用

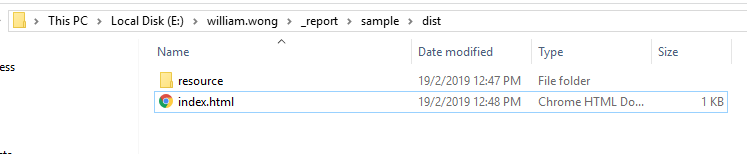
favicon就是html icon，這個因為js、css和html的佈置路徑不一不會使用，建議大家直接在html template內標明位置會比較簡單

filename就是打包後路徑連名字，沒什麼特別

template就是之前說那個html文檔

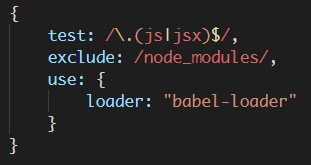
chucks這個部份相對就非常重要，它是對應entry部份聲明的變數，就是說這個html檔案會用到那些js檔，所以你會看到教學中它就引用到一個通用的和主要的js了

然後我們可以再一次執行『npm run start』打包我們的專案，切換路徑到『/dist』底下，你會見到所有檔案會根據我們剛剛定設計來生成了



教學也差不多到尾聲，插件的安裝在教學前半部份已經執行，現在我們只差幾個loader的配置

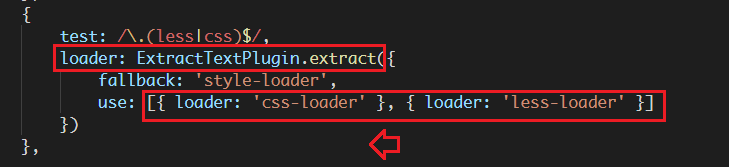
-babel-loader，它本來就支援JS及JSX檔案的讀取，可是我們還是需修改它的設定讓它懂得去讀取兩種檔案



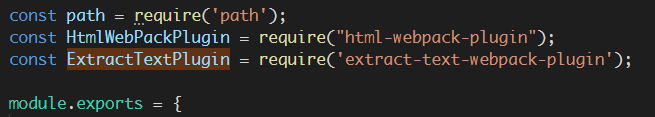
-style-loader、css-loader及less-loader，它們則是用來支援CSS及LESS檔案的讀取，設定內容如下，當中有兩點需要說明一下

1. 設定loader時還需要額外使用ExtractTextPlugin插件，它的目的並不是用作讀取檔案的說明，而是執行打包時會把樣式的內容獨立打包到CSS檔案及在HTML內加入引用路徑等動作

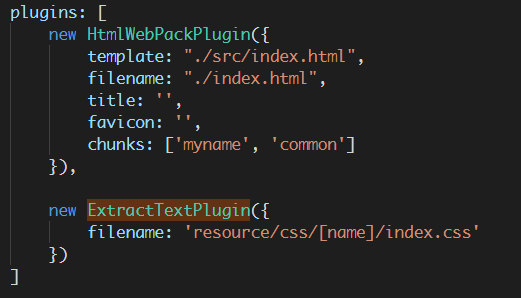
2. use部份支援多個loader的配置，比較有趣的是loader的使用是有次序性的，而它的次序則是由陣列尾至頭來使用



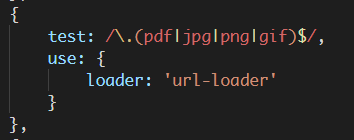
另外，我們需要在『webpack.config.js』置頂位置加入插件的引用的說明



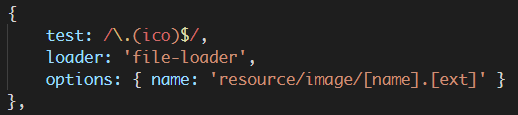
最後，在plugins部份加入以下內容，它會根據你設定的路徑還有[name]來生成樣式檔案



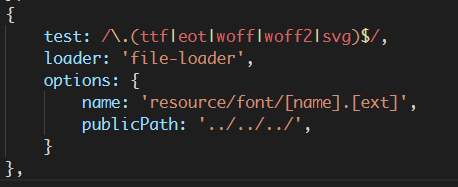
-url-loader，由於我開發的專案對圖片的使用都是把圖片轉化為BASE64字元並直接加載到HTML內，所以圖片讀取都是用url-loader執行；如果你是打算直接把圖片複製到打包後的路徑引用，那你就需要使用file-loader，關於file-loader使用之後會作說明



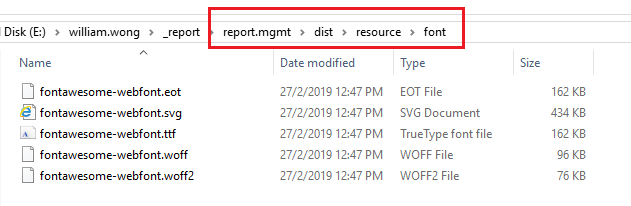
-file-loader，如果你對之前打包HTML的說明還有印象的話，那部份提到ICON的設定好像是有BUG不能直接使用，所以最後實現的方法是使用file-loader把檔案複製到目的地，options部份可以加入name來設定目的地路徑，而[name]是檔案名稱，[ext]則是說明來源的副檔名



除了ICON需要使用file-loader之外，專案的字體檔案也是需要的，而options內的publicPath內容非常不好解釋，我就以我開發的專案為例來說明一下



第一點就是有沒有publicPath對生成檔案到目的地是完全沒有影響的



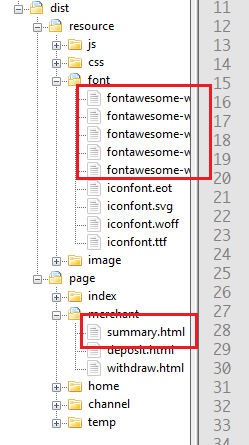
第二點是如果你知道怎麼在HTML內使用別人自定義的字體檔案時，大慨會了解除了字體的SVG、WOFF及TTF等檔案，還有一個重要的CSS檔案，在裡面都會有一條URL PATH指向剛剛說明的檔案，而publicPath則會影響到這條URL PATH的設置，使用到WEBPACK打包，它肯定會覆寫一次你的URL PATH內容而你是阻止不了



看到這個PATH你會知道WEB SERVER會在HTML檔案同一層的資料夾內嘗試找你聲明的檔案，而事實是肯定找不到，因為我們設計網站的架構多半是把資料跟文檔分開的，這個時候正正是publicPath發揮其功效的時候，我們先把那個publicPath加回去



它的內容就變成了『../../../』開頭，所以這個publicPath有點像前置內容，然後我們再看一看網站架構



這樣就能夠讓HTML檔案順利找到需要的字體位置了，當然，這個也需要按個人要求作出適當的調整

修改『webpack.config.js』檔案內容如下:

const path = require('path');  
const HtmlWebPackPlugin = require("html-webpack-plugin");  
const ExtractTextPlugin = require('extract-text-webpack-plugin');  
  
module.exports = {  
 entry: {  
 myname: './src/index.js',  
 common: [  
 './src/common/index.js'  
 ]  
 },  
 output: {  
 path: path.resolve(\_\_dirname, 'dist'),  
 filename: 'resource/js/[name]/index.js'  
 },  
 module: {  
 rules: [  
 {  
 test: /\.(js|jsx)$/,  
 exclude: /node\_modules/,  
 use: {  
 loader: "babel-loader"  
 }  
 },  
 {  
 test: /\.(less|css)$/,  
 loader: ExtractTextPlugin.extract({  
 fallback: 'style-loader',  
 use: [  
 { loader: 'css-loader' },   
 { loader: 'less-loader' }  
 ]  
 })  
 },  
 {  
 test: /\.(pdf|jpg|png|gif)$/,  
 use: {  
 loader: 'url-loader'  
 }  
 },  
 {  
 test: /\.(ico)$/,  
 loader: 'file-loader',  
 options: { name: 'resource/image/[name].[ext]' }  
 },  
 {  
 test: /\.(ttf|eot|woff|woff2|svg)$/,  
 loader: 'file-loader',  
 options: {  
 name: 'resource/font/[name].[ext]',  
 publicPath: '../../../',  
 }  
 },  
 ]  
 },  
 plugins: [  
 new HtmlWebPackPlugin({  
 template: "./src/index.html",  
 filename: "./index.html",  
 title: '',  
 favicon: '',  
 chunks: ['myname', 'common']  
 }),  
 new ExtractTextPlugin({  
 filename: 'resource/css/[name]/index.css'  
 }),  
 ]  
};

到這裡環境配置的教學算是完了，下一課就是如何製作一個簡單REACT網頁了

相關文章的連結

<https://juejin.im/post/5bc41ae0f265da0af161674e>

https://medium.com/@joseph0crick/react-css-modules-less-webpack-4-a50d902d0a3

https://medium.freecodecamp.org/part-1-react-app-from-scratch-using-webpack-4-562b1d231e75