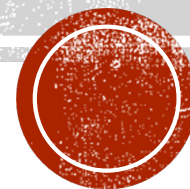


ES5~ES6+

林新德

shinder.lin@gmail.com



<https://github.com/shinder/es6-ts-practice>

1.1 ECMAScript

- ECMA 是歐洲的一個標準制訂組織，JavaScript 依循其制訂的標準，故其標準又稱為 ECMAScript (簡稱 ES)。
- 在 HTML5 之後，ES 標準停滯在 ES5 算是比較長的時間，待 2015 年 ES6 標準發佈，其內容新增較多的業界期待的功能，是功能變動比較大的版本。
- 目前 ECMA 每年都會發佈新的 ES 程式語言標準，如 2016 年的 ES7，相對於從 ES5 到 ES6，ES7 算是變動比較少的，所以之後的版本就總稱為 ES6+。
- 本課程的目標是介紹 ES6+ 重要的功能和應用。



1.2 Babel.js

- <https://babeljs.io/>
- ES6 剛發佈的時候，支援的瀏覽器很少。必須倚靠版本的線上更新來增加對 ES6 的支援。但某些瀏覽器，尤其是 IE 更新相當緩慢，甚至不再更新，完全無法使用 ES6 語法。
- ES6+ 語法簡潔，又可以解決某些語法上明顯的缺點。社群為了上述原因，於是有 Babel 的誕生。
- Babel 是 JS 的編譯器、轉譯器（transpile），主要的功能是將 ES6+ 的語法轉換成 ES5 的語法。
- 市場上所有瀏覽器對 ES5 的支援程度接近 100%。
- Babel 也可以依設定，將 JSX 轉換為 JavaScript。



1.3 Node.js

- 2009年 Ryan Dahl 使用 Chrome 的 JavaScript 引擎（代號：V8），包裝成 JavaScript 執行環境（Runtime）Node.js。
- 可以在瀏覽器以外執行 JavaScript（像 Python 或 Ruby），讀寫檔案、寫服務程式、做資料庫連線等。
- 官網：<https://nodejs.org/>
- 目前前端工具超過九成都是 Node.js 套件或程式。
- 安裝：至官網下載安裝檔。
- 安裝後，開啟命令提示列（command prompt、terminal）。
- 查看版本：`> node --version`
- 查看 npm 版本：`> npm -v`



2. 開發環境

- 目前 **Chrome** 的 **V8** 支援 **ES6+** 情況良好，除了 **import** 和 **export** 不支援外，其餘 **ES6** 的特色幾乎都可以直接執行。
- **Firefox** 支援 **ES6+** 的狀況也和 **Chrome** 差不多。
- 但為了支援程度差的瀏覽器，我們還是得需要建立轉換的開發環境。
- 課程的前半部會先以瀏覽器做為語法測試環境。



3. var 和 let 的差異

- **var** 宣告的全域變數會變成 **window** 物件的屬性；**let** 則不會。
- **var** 宣告的變數以 **function** 的大括號為活動範圍；**let** 則是以所在的大括號為活動範圍。
- 在 **for** 迴圈的起始式內，以 **let** 宣告的變數，在迴圈中會是個別的變數。
- **const** 宣告的變數，除了不能用等號再設定外，其餘特性和 **let** 相同。



4. 解構運算子

- 用在 Object 或 Array 的解構或重組。

```
const data = {  
  name: 'Shinder',  
  age: 28,  
  gender: 'male'  
};  
  
let {name, gender, age, title} = data; // 解構設定  
console.log(name, gender, age, title);  
  
let {name: nickname, age: myAge} = data; // 解構設定  
console.log(nickname, myAge);
```



```
data.age ++;  
({name, gender, age} = data); // 設定給既有的變數  
console.log(name, gender, age);  
let data2 = {  
    ...data, // 解構  
    title: 'developer'  
};  
console.log(JSON.stringify(data2));  
  
const ar = [3, 5, 8];  
let [d, e, f] = ar;  
const [g, h] = ar;  
console.log(d, e, f);  
console.log(g, h);  
  
[d, e, f] = [7, 88, 999];  
console.log(d, e, f);  
  
const ar2 = [2, ...ar, 9];  
console.log(JSON.stringify(ar2));
```



5. 箭頭函式

```
const f1 = a=>a*a;
const f2 = (a=7)=>a*a;

const obj = {
  v: this,          // this 為 window (模組內為 undefined)
  f3: ()=> this,    // this 為 window (模組內為 undefined)
  f4: function (){
    return this;
  },
  f5(){
    return this;
  }
}

console.log(f1(8), f2());
console.log(typeof obj.f3(), typeof obj.f4(), typeof obj.f5());
console.log(typeof obj.v);
```



6. 字串樣版

```
const info = document.querySelector('#info');
const tplBall = o=>{
  return `<div style="border-radius:50%;
    text-align:center; position:absolute;
    width: ${o.size || 60}px; height: ${o.size || 60}px;
    line-height: ${o.size || 60}px;
    left: ${o.left || 0}px; top: ${o.top || 0}px;
    background-color: ${o.bg}; ">${o.n}</div>`;
}
const data = [
  {n:1, bg: 'red'},
  {n:2, bg: 'orange', left:60, top:60},
  {n:3, bg: 'yellow', left:100, top:100, size: 200},
  {n:4, bg: 'green', left:120, top:80, size: 80},
];
data.forEach(el => info.innerHTML += tplBall(el) );
```



7. 陣列的方法補充 (ES5)

```
const ar = [ 1, '2', 3, 4, 5, '6', '7', 8];

const ar2 = ar.filter((el, index, array)=> typeof el==='number');
console.log('ar2:', ar2.toString());

const ar3 = ar.map(el=> el * el);
console.log('ar3:', ar3.toString());

const ar4 = ar.reduce((previousValue, currentValue, index) => {
  console.log(previousValue, currentValue, index);
  return previousValue*1 + currentValue*1;
})
console.log('ar4:', ar4.toString(), typeof ar4);
```



■ 排序

```
const ar2 = [];  
  
for(let i=0; i<20; i++){  
    ar2.push( Math.floor(Math.random()*20));  
}  
console.log(ar2.join(','));  
ar2.sort((a, b)=>{  
    // console.log(`a: ${a}, b: ${b}`);  
    return a-b; // 由小到大  
});  
console.log(ar2.join(','));
```



8. Map 和 Set

```
const map = new Map;  
const obj = {};  
  
map.set({}, {a:1, b:2});  
map.set({}, {a:3, b:4});  
map.set(obj, {a:5, b:6});  
map.set(obj, {a:7, b:8}); // obj 的 reference 同一個  
map.forEach((v,k)=>{  
    console.log( JSON.stringify(k) + ' :::' + JSON.stringify(v) );  
});  
console.log( map.size );
```



```
const set = new Set;

set.add({a:1, b:2}).add({a:1, b:2});
set.add(obj).add(obj).add({a:1, b:2});
set.forEach(v=>{
    console.log(JSON.stringify(v));
});
console.log( set.size );
set.delete(obj);
console.log('刪除 obj 之後：', set.size );
set.clear();
console.log('清空之後：', set.size );
```



9. 使用 Promise

- **Promise** 是用來處理非同步時，改善 **callback functions** 一直往右縮排的情況。
- 讓整個結構拉成直的，不使其往右內縮。
- 但還是使用 **callback function** 的方式。

```
new Promise((resolve, reject)=>{
  setTimeout(function(){
    resolve('Hello');
    // reject('bad');
  }, Math.random()*1000);
})
  .then(result=>{
    console.log(`result: ${result}`);
    return 'hi';
  })
  .then(a=>{
    // throw new Error('abc');
    console.log(`a: ${a}`);
  })
  .catch(ex=>{
    console.log(`ex: ${ex.toString()}`);
  })
  .then(b=>{
    console.log(`b: ${b}`);
  })
  )
```



```
const myFunc = (msec)=>{
  return new Promise((resolve, reject)=>{
    setTimeout(function(){
      resolve(Math.floor(Math.random()*100));
    }, msec);
  });
};

myFunc(1300)
  .then(r=>{
    console.log(r);
    return myFunc(1400);
  })
  .then(r=>{
    console.log(r);
    return myFunc(1200);
  })
  .then(r=>{
    console.log(r);
  })
}
```



10. 使用 `async/await`

- 1. `async-await` 要解決什麼問題？依序執行
- 2. `await` 一定要在 `async` 宣告的函式中使用
- 3. `await` 後面接 `Promise` 物件
- 4. `async` 修飾的函式回傳 `Promise` 物件
- 5. 例外處理使用 `try/catch` 敘述



```
const myFunc = (msec)=>{
  return new Promise((resolve, reject)=>{
    setTimeout(function(){
      resolve(Math.floor(Math.random()*100));
    }, msec);
  });
};

(async ()=>{
  const r1 = await myFunc(1300);
  console.log(r1);
  const r2 = await myFunc(1400);
  console.log(r2);
  const r3 = await myFunc(1200);
  console.log(r3);
})();
```



11.1 安裝 Babel 轉譯環境

1. 初始化專案

```
mkdir es6-plus-v2 # 建立專案資料夾
cd es6-plus-v2    # 進入專案資料夾
npm init -y       # 初始化專案
```

2. 安裝 Babel

```
# 安裝 babel 核心
npm install --save-dev @babel/core @babel/cli
# 安裝 babel 解譯環境
npm i -D @babel/preset-env
# 安裝 可對老舊瀏覽器增進 Promise 功能的 polyfill
npm i @babel/polyfill
```



3. 設定 Babel，建立 `.babelrc` 設定檔，內容如下：

```
{ "presets": ["@babel/preset-env"] }
```

4. 建立 `src` 和 `dist` 資料夾

5. 查看 babel 參數

```
npx babel --help
```

6. 在 `package.json` 檔裡的 `scripts` 設定

```
"transpile": "npx babel src --out-dir dist"
```

7. 在 `src/app.js` 新增內容：

```
let f = name => {  
  console.log(`hello ${name}`);  
};  
f('bill');
```

8. 執行 `npm run transpile` 後，可以在 `dist/app.js` 看到轉譯後的 ES5 寫法。



11.2 Webpack

- Webpack 是一個 task runner 工具，透過不同的外掛可以幫你完成許多事情。

9. 安裝 webpack

```
npm i -D webpack
```

```
npm i -D webpack-cli
```

```
# 安裝 babel-loader
```

```
npm i -D babel-loader
```

10. 建立 build 資料夾，建立 build/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div id="info"></div>
  <script src="bundle.js"></script>
</body>
</html>
```



11. 建立設定檔 webpack.config.js

```
const path = require('path');
module.exports = {
  entry: './src/app.js',
  mode: 'development', // development or production
  output: {
    path: path.resolve(__dirname, 'build'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /(node_modules|bower_components)/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env']
          }
        }
      }
    ]
  }
};
```



12. 修改 src/app.js

```
let f = name=>{  
  document.querySelector('#info').innerHTML = `hello ${name}`;  
};  
f('bill');
```

13. 執行 `npx webpack` 即可看到轉譯好的 `build/bundle.js`。以瀏覽器開啟 `build/index.html` 可看到執行結果。



11.3 Webpack-dev-server

- Webpack-dev-server 是一個搭配 webpack 工作流程的開發用 web server，底層是 node/express。
- 啟動 webpack-dev-server 後，會監看原始碼（包含其它檔案），有變動時，自動轉譯，並透過 websocket 通知頁面自動從新載入，對開發相當方便。



14. 安裝 webpack-dev-server

```
npm i -D webpack-dev-server
```

15. webpack.config.js 加入設定

```
devServer: {  
  contentBase: path.join(__dirname, 'build'),  
  compress: true,  
  port: 9000  
}
```

16. 針對 webpack5，在package.json 檔裡的 scripts 設定

```
"dev": "webpack serve"
```

17. 執行 npm run dev 即可啟動開發環境。



12. 模組的匯入與匯出

my-export01.js

```
export default a=>a*a*a;  
  
const ar1 = [1, 3, 5, 7];  
const ar2 = [2, 4, 6, 8];  
  
export {ar1, ar2};
```

my-export02.js

```
export default a=>a*a*a;  
  
const ar1 = [1, 3, 5, 7];  
const ar2 = [2, 4, 6, 8];  
  
export {ar1};  
export {ar2};
```

my-import01.js

```
import f1 from './my-export01';  
import {ar1} from './my-export01';  
  
console.log(f1(3));  
console.log(JSON.stringify(ar1));
```

my-import02.js

```
import {ar2 as array2} from './my-export02';  
import {ar2 as array3} from './my-export02';  
  
console.log(JSON.stringify(array2));  
console.log(array2===array3);
```



13. 類別 class

```
class Person {  
  // 建構函式  
  constructor(name='unkown', age=NaN, gender='unkown') {  
    this.name = name; // 設定屬性  
    this.age = age;  
    this.gender = gender;  
  }  
  toString() {  
    return `${this.name} ${this.age} ${this.gender}`;  
  }  
}  
export default Person; // 匯出  
console.log(new Person);  
const p2 = new Person('Flora', 23, 'female');  
console.log(p2);  
console.log(p2.constructor.name);
```



```
import Person from './my-class01'; // 匯入
export default class Employee extends Person {
  // 建構函式
  constructor(name='unkown', age=NaN, gender='unkown') {
    super(name, age, gender); // 呼叫父類別的建構函式
    this._employeeId = '';
  }
  toString() {
    return `${this.name}-${this.age}-${this.gender}-${this._employeeId}`;
  }
  get employeeId() {
    return this._employeeId;
  }
  set employeeId(v) {
    this._employeeId = v;
  }
}
const p3 = new Employee('Bill', 28, 'male');
p3.employeeId = 'C006';
console.log(p3);
```



14. TypeScript 開發環境

- 參考 <https://github.com/microsoft/TypeScript-Babel-Starter>
- 若已經安裝（12）的 babel 環境建議另外開一個專案安裝 TypeScript 環境
- Webpack 設定可以參考官方設定 <https://webpack.js.org/guides/typescript/>



14.1 安裝轉譯環境

1. 初始化專案

```
mkdir es6-ts-practice # 建立專案資料夾  
cd es6-ts-practice.   # 進入專案資料夾  
npm init -y           # 初始化專案
```

2. 安裝套件

```
# 安裝 TypeScript 解譯器  
npm install --save-dev typescript  
# 安裝 babel 核心  
npm install --save-dev @babel/core @babel/cli  
# 安裝 babel 解譯環境  
npm i -D @babel/preset-env  
npm i -D @babel/preset-typescript  
# 安裝 plugin  
npm i -D @babel/plugin-proposal-class-properties
```



3. 設定 Babel，建立 .babelrc 設定檔，內容如下：

```
{  
  "presets": [ "@babel/env", "@babel/typescript" ],  
  "plugins": [  
    "@babel/proposal-class-properties"  
  ]  
}
```

4. 建立 src 資料夾

5. 在 package.json 檔裡的 scripts 設定

```
"scripts": {  
  "type-check": "tsc --noEmit",  
  "type-check:watch": "npm run type-check -- --watch",  
  "build": "npm run build:types && npm run build:js",  
  "build:types": "tsc --emitDeclarationOnly",  
  "build:js": "babel src --out-dir dist --extensions '.ts,.tsx' "  
},
```



6. 設定 TypeScript 設定檔 tsconfig.json，內容如下：

```
{  
  "compilerOptions": {  
    "outDir": "./dist/",  
    "sourceMap": true,  
    "noImplicitAny": true,  
    "module": "commonjs",  
    "target": "es5",  
    "jsx": "react",  
    "allowJs": true  
  }  
}
```



14.2 Webpack

- Webpack 是一個 task runner 工具，透過不同的外掛可以幫你完成許多事情。

7. 安裝 webpack

```
npm i -D webpack
npm i -D webpack-cli
# 安裝 ts-loader
npm i -D ts-loader
```

8. 建立 dist/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div id="info"></div>
  <script src="bundle.js"></script>
</body>
</html>
```



9. 建立設定檔 webpack.config.js

```
const path = require('path');
module.exports = {
  mode: 'development', // development or production
  entry: './src/app.ts',
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        use: 'ts-loader',
        exclude: /node_modules/,
      },
    ],
  },
  resolve: {
    extensions: [ '.tsx', '.ts', '.js' ],
  },
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```



10. 安裝 webpack-dev-server

```
npm i -D webpack-dev-server
```

11. webpack.config.js 加入設定

```
devServer: {  
  contentBase: path.join(__dirname, 'build'),  
  compress: true,  
  port: 9000  
}
```

12. 在 package.json 檔裡的 scripts 設定

```
"dev": "webpack serve"
```

13. 執行 npm run dev 即可啟動開發環境。



15. 其他主題

- fetch API: https://developer.mozilla.org/zh-TW/docs/Web/API/Fetch_API/Using_Fetch
- Axios 工具: <https://www.npmjs.com/package/axios>

