模組系統

JavaScript語言長期以來並未內建支援模組系統，社群上發展了兩套知名的模組系統，但它們並不相容:

- CommonJS Modules
- Asynchronous Module Definition (AMD)

ES6中加入了模組系統的支援，它採用了CommonJS與AMD的優點，成了未來JavaScript語言中重要的特性。

模組系統是什麼？

當程式碼愈寫愈多，應用程式的規模愈來愈大時，我們需要一個用於組織與管理程式碼的方式，這個需求相當明確，或許不只是應用程式發展到一定程度才會考慮這些，而是應該在開發程式之前的規劃就需要考量進來。

JavaScript語言是一個沒有命名空間設計的程式語言，也沒有支援類似的組織與程式碼分離的設計。有些人認為使用物件定義的字面文字，可以定義出物件的方法與屬性，但如果你看過"物件"、"this"與"原型物件導向"的章節內容，就知道物件中並沒有區分私有、公開成員或方法的特性，這個組織法頂多只是把方法或屬性整理集中而已。

而在很早之前(2003)在社群上發展出一個稱之為模組樣式(module pattern)，以及之後的變型如 暴露模組樣式(Revealing Module Pattern)，就是第一期的程式碼組織方式。模組樣式實作相當簡單，有許多早期開始發展的函式庫或框架採用這個樣式，甚至到今天也可以看到它的使用身影。一個簡單的範例如下(以下範例來自jQuery):

```
// The module pattern
var feature = (function() {

    // Private variables and functions
    var privateThing = "secret";
    var publicThing = "not secret";

    var changePrivateThing = function() {
        privateThing = "super secret";
    };

    var sayPrivateThing = function() {
        console.log( privateThing );
        changePrivateThing();
    };

    // Public API
    return {
        publicThing: publicThing,
        sayPrivateThing: sayPrivateThing
    };
})();

feature.publicThing; // "not secret"

// Logs "secret" and changes the value of privateThing
feature.sayPrivateThing();
```

它使用了IIFE函式的特性，區分出作用域，不過它並沒有辦法徹底解決問題，它在小型的應用程式可以用得很好，但在複雜的程式中仍然有很大的問題，例如以下的問題:

- 沒辦法在程式中作模組載入
- 模組之間的相依性不易管理
- 異步載入模組
- 除錯與測試都不容易
- 在大型專案中不易管理

模組樣式似乎是一個暫時性的解決方案，但不得不說它的確是上一代很重要的程式碼組織方式，第二代的模組系統，是在2009年之後的CommonJS與AMD計劃，它們實作出真正完整的模組系統，CommonJS是專門設計給伺服器端的Node.js使用的，而AMD的目標對象則是瀏覽器端。當然它們兩者的設計有所不同，也不相容，使用時也可能需要搭配載入工具來一併使用，不過這個階段的模組系統已經是較前一代完善許多，在相依性與模組輸出與輸入，都有相對的解決方式，程式碼的管理與組織方便了許多。

CommonJS與AMD並不會在這裡討論，我們的重點是是ES6中的模組系統，ES6中加入了模組系統的支援，它採用了CommonJS與AMD的優點，是一個語言內建的模組系統，而且它可以使用於瀏覽器與伺服器端，這是一個重大的新特性，可以讓你的開發日子更輕鬆許多。

模組如何開始使用

ES6的模組系統使用上相當簡單，大致上只有三個重點:

- ES6的模組程式碼會自動變成strict-mode(嚴格模式)，不論你有沒有使用"use strict"在程式碼中。
- ES6的模組是一個檔案一個模組
- ES6模組使用export(輸出)與import(輸入)語句來進行模組輸出與輸入。輸出通常位於檔案最後，輸入位於最前面。

模組輸出與輸入

有寫成模組的程式碼檔案，才能讓其他程式碼檔案進入輸入。模組輸出可以使用export關鍵字，在想要輸出(也就是變為公開部份)加在前面，物件、類別、函式與原始資料(變數與常數)都可以輸出，例如以下的範例:

多個輸出名稱

```
export const aString = 'test'

export function aFunction(){
  console.log('function test')
}

export const aObject = {a: 1}

export class aClass {
  constructor(name, age){
    this.name = name
    this.age = age
  }
}
```

上面稱之為多個輸出名稱的情況，有兩種方式可以進行輸入，一種是每個要輸入的名稱都需要定在在花括號({})之中，例如以下的範例:

```
import {aString, aObject, aFunction, aClass} from './lib.js'

console.log(aString)
console.log(aObject)
```

另一種是使用萬用字元(*)，代表要輸入所有的輸出定義的值，不過你需要加上一個模組名稱，例如下面程式碼中的 `myModule` ，這是為了防止命名空間的衝突之用的，之後的程式碼中都需要用這個模組名稱來存取輸出模組中的值，這個作法不常使用:

```
import  * as myModule from './lib.js'

console.log(myModule.aString)
console.log(myModule.aObject)

myModule.aFunction()
const newObj = new myModule.aClass('Inori', 16)
console.log(newObj)
```

單一輸出名稱

這個要輸出成為模組的程式碼檔案中，只會有一個輸出的變數/常數、函式、類別或物件，通常會加上 `default` 關鍵詞。如果要使用有回傳值的函式，通常也是用單一輸出的方式。例如以下的範例:

```
function aFunction(param){
  return param * param
}

export default aFunction
```

對單一輸出的模組就不需要用花括號，這代表只輸入以 `default` 值定義的輸出語句:

```
import aFunction from './lib2.js'

console.log(aFunction(5))
```

這是最特別的，可以在輸入時改變輸入值的名稱，這樣可以讓作輸入檔案中，確保不會有名稱衝突的情況:

```
import square from './lib2.js'

console.log(square(5))
```

合法的輸出語法

```
export var x = 42;                      // export a named variable
export function foo() {};               // export a named function

export default 42;                      // export the default export
export default function foo() {};       // export the default export as a function

export { encrypt };                     // export an existing variable
export { decrypt as dec };              // export a variable as a new name
export { encrypt as en } from 'crypto'; // export an export from another module
export * from 'crypto';                 // export all exports from another module
```

合法的輸入語法

```
import 'jquery';                        // import a module without any import bindings
import $ from 'jquery';                 // import the default export of a module
import { $ } from 'jquery';             // import a named export of a module
import { $ as jQuery } from 'jquery';   // import a named export to a different name
import * as crypto from 'crypto';    // import an entire module instance object
```

參考資料

- http://stackoverflow.com/questions/36795819/react-native-es-6-when-should-i-use-curly-braces-for-import/36796281#36796281
- http://exploringjs.com/es6/ch_modules.html
- http://www.2ality.com/2014/09/es6-modules-final.html
- https://addyosmani.com/resources/essentialjsdesignpatterns/book/#modularjavascript
- https://www.nczonline.net/blog/2016/04/es6-module-loading-more-complicated-than-you-think/