



# 使用ES6語法



designed by freepik

Estimated time:

50 min.

資訊工業策進會 Institute for Information Industry

【Key Points】：

# 學習目標

- **5-1: 使用類別**
- **5-2: 箭頭函式**
- **5-3: 模組的引入和輸出**



5-1

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

本模組將依序學習：

- 使用類別
- 箭頭函式
- 模組的引入和輸出

最後會有「自我評量」與「課後練習步驟(Lab)」，請各位依照說明進行練習

Node.js是許多新創公司選擇的開發語言，跟它的開發生態有關，因為JavaScript在網頁開發上幾乎很難短時間被取代，作為一個後端開發它又有豐富的套件生態，甚至前端套件也可以運用在後端，所以只要了解套件使用，就可以快速開發。

## 【Key Points】：

- 使用類別
- 箭頭函式
- 模組的引入和輸出

## 5-1: 使用類別

- Function
- Class
- 繼承
- 原型鏈



designed by freepik



designed by freepik

5-2

III 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 每個 function 都有 prototype 屬性。
2. 透過 new 關鍵字，可以生成一個物件。
3. ES6 的繼承作法都是透過 extends 關鍵字，繼承某類別。
4. 子類別的 this 會指向父類別。
5. 所有 class 的原型是 Object。

### 【Key Points】：

每個 function 都有 prototype 屬性

ES6 的繼承作法都是透過 extends 關鍵字，繼承某類別

所有 class 的原型是 Object

# Function

- JavaScript的函式就是物件。

```
function Car(){  
    this.name='BMW'  
}  
  
const myCar=new Car()  
const yourCar=new Car()  
console.log(myCar.name) //BMW  
console.log(yourCar.name) //BMW
```

5-3



1. 每個function都有prototype屬性。
2. 透過new 的關鍵字，就可以生成一個物件。
3. 共有方法是透過物件的\_\_proto\_\_屬性指向到函式的方法。
4. 可以透過Object.getPrototypeOf( )確認物件的原型為何。
5. 建議不要輕易地去改動prototype，可能會影響到所有物件。

## 【Key Points】：

JavaScript的函式就是物件

每個function都有prototype屬性

透過new 的關鍵字，就可以生成一個物件

# Class

```
class Car{  
    name='BMW'  
    static crash() {  
        console.log('car crash')  
    }  
    static get license(){  
        return this.license  
    }  
    static set setLicense(license){  
        this.license=license  
    }  
  
const car=new Car()  
console.log(car.name) // BMW  
Car.crash() //car crash  
Car.setLicense='ABC-123'  
console.log(Car.License) //ABC-123|
```

- Class name大寫宣告
- Class 屬性宣告
- static
- getter/setter

1-4



1. 透過new關鍵字，可以生成一個物件。
2. static 是靜態方法關鍵字，可以不用透過new的動作產生物件。
3. getter/setter在物件導向主要是用來提供介面讓外界可以修改或讀取class 內部私有變數值。
4. getter/setter 在JavaScript尚有一些遺憾之處，因為在ES6還沒有private variable，所以可以直接 this.license ="xxx" 紿值，在ES2019才會有private variable解決這問題。(變數前面加上#)
5. 建議直接在get function 利用 return value會更為安全些。

## 【Key Points】：

透過new關鍵字，可以生成一個物件

說明的時候，務必說明到：

class、static、get、set、return 等關鍵字

說明物件封裝的 private 在 ES6 仍得靠程式設計師自律

## 繼承

- **extends**
- **super**
- **constructor**

```
class Phone{  
    sendSMS=function(){  
        console.log('send sms')  
    }  
}  
  
class HTC extends Phone{  
    constructor(){  
        super()  
    }  
    name='HTC'  
}  
  
const htc=new HTC()  
console.log(htc.name) //HTC  
htc.sendSMS() //send sms
```

1-5



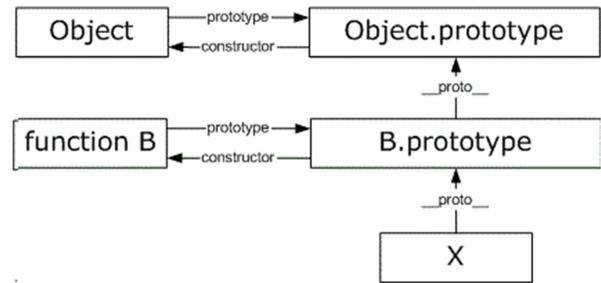
1. ES6的繼承作法都是透過**extends** 關鍵字，繼承某類別。
2. **super()**可以訪問父類別的**constructor**。
3. **super()**只能在**constructor**函式裏頭呼叫。
4. 如果類別有繼承別的類別，要透過**this**宣告類別的屬性，得先在**constructor**函式裏頭宣告 **super()**，並且在**this**之前宣告。
5. **constructor**一般稱為建構函式，可以替物件宣告內部變數。

### 【Key Points】：

ES6的繼承作法都是透過**extends** 關鍵字  
**super()**可以訪問父類別的**constructor**  
**Constructor** 建構函式

## 原型鏈

- **this**
- **Object.getPrototypeOf()**
- **Object.hasOwnProperty()**



1-6

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 子類別的this會指向父類別。
2. Object.getPrototypeOf()可以確認兩個類別是否存在繼承關係。
3. Object.hasOwnProperty()不會檢查到父類別的屬性。
4. 所有class的原型是Object。
5. 所有的繼承骨子裡也都還是透過prototype，只是在寫法上ES6提供更好的語法。

### 【Key Points】：

子類別的this會指向父類別

Object.getPrototypeOf()可以確認兩個類別是否存在繼承關係

Object.hasOwnProperty()不會檢查到父類別的屬性

## 5-2: 箭頭函式

- 箭頭符號 =>
- 常見的寫法
- 與舊有寫法的對照
- 箭頭函式中的this



designed by freepik



designed by freepik

5-7

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

1. 箭頭函式顧名思義最大特徵就是=>符號。
2. 箭頭函式最大功用就是提供一個更簡潔的寫法。
3. 所有的箭頭函式都是匿名函式。
4. 在沒使用箭頭函式的情況下，this有可能會指向全域物件。
5. 箭頭函式沒有prototype也就代表沒有原型。

### 【Key Points】：

箭頭函式顧名思義最大特徵就是=>符號  
提供一個更簡潔的函式寫法  
箭頭函式沒有prototype也就代表沒有原型

## 箭頭符號 =>

```
const helloworld=()=>{
  console.log('hello world')
}

const getNumber=(num)=>num+1

helloworld()
console.log(getNumber(1))
```

```
class Car{
  getName()=>{}
}
```

```
const sayHi=()=>{
  return 'hi'
}

const hi=new sayHi()
console.log(hi)
```

5-8



1. 箭頭函式顧名思義最大特徵就是=>符號。
2. 程式碼更為簡潔，可讀性提高。
3. `(a)=>a+1`，相當於`function(a){ return a+1 }`。
4. `class` 的方法宣告不能使用箭頭函式。
5. 無法對一個箭頭函式使用`new`關鍵字產生物件。

### 【Key Points】：

箭頭函式顧名思義最大特徵就是=>符號

對照兩種寫法，讓學員體會箭頭函式簡潔的特性。

無法對一個箭頭函式使用`new`關鍵字產生物件

## 常見的寫法

- 如果函式只有一個參數，可以省略( )  
const sum=x=>x+1
- 如果函式有兩個以上的參數，( )不能省略  
const sum=(x,y)=>x+y
- 函式的參數可以先給定預設值  
const sum=(x=1,y=2)=>x+y
- const getArgs= (...args)=>{console.log(args)}
- const obj={  
    getArgs:(...args)=>{console.log(args)}  
}

5-9



1. 如果函式只有一個參數，可以省略( )，並且可以省略return關鍵字。
2. 如果函式有兩個以上的參數，( )不能省略。
3. 函式的參數可以先給定預設值。
4. 透過...符號，就可以讀取所有傳進函式的參數。
5. 物件的屬性方法也可以使用箭頭函式。

### 【Key Points】：

如果函式只有一個參數，可以省略( )  
如果函式有兩個以上的參數，( )不能省略  
函式的參數可以先給定預設值

## 與舊有寫法的對照

```
const sum=x=>x+1  
  
function sum (x){  
    return x+1  
}
```

```
const sum=(x,y)=>x+y  
  
function sum(x,y){  
    return x+y  
}
```

```
const sum=(x=1,y=2)=>x+y  
  
function sum(x=1,y=2){  
    return x+y  
}
```

```
const getArgs= (...args)=>{  
    console.log(args)  
}  
  
function getArgs(...args){  
    console.log(args)  
}
```

```
const obj={  
    getArgs: (...args)=>{  
        console.log(args)  
    }  
}  
  
const obj={  
    getArgs: function(...args){  
        console.log(args)  
    }  
}
```

```
; (function(){  
    console.log('hello')  
})()  
  
; (()=>{  
    console.log('hi')  
})()
```

5-10



- 箭頭函式最大功用就是提供一個更簡潔的寫法。
- 所有的箭頭函式都是匿名函式。
- 立即函式也可以使用箭頭函式。
- 物件的屬性方法也可以使用箭頭函式。
- 類別的方法宣告不能使用箭頭函式。

### 【Key Points】：

立即函式也可以使用箭頭函式  
物件的屬性方法也可以使用箭頭函式。  
類別的方法宣告不能使用箭頭函式。

# 箭頭函式中的this

- this 指向
- this與作用域的關係
- var self = this
- 箭頭函式沒有prototype
- apply, call, bind無法覆寫this

5-11



1. 在沒使用箭頭函式的情況下，this有可能會指向全域物件。
2. 在箭頭函式中的this與其作用域有關，this指向不會超出其作用域，會指向到呼叫該function的物件。
3. 在沒有箭頭函式前，var self = this常被加在物件的函式內部，以解決this指向全域物件的問題。
4. 箭頭函式沒有prototype也就代表沒有原型。
5. apply, call, bind在非箭頭函式的使用裡，可以用來改變function中this的指向，但箭頭函式中則無法。

## 【Key Points】：

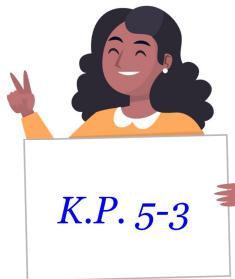
在箭頭函式中的this與其作用域有關

在沒有箭頭函式前，var self = this常被加在物件的函式內部，以解決this指向全域物件的問題

箭頭函式沒有prototype也就代表沒有原型

## 5-3:模組的引入和輸出

- CommonJS
- AMD
- ES6 Modules
- CommonJS的輸出/引入



designed by freepik



designed by freepik

5-12

 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

本知識點將依序學習：

CommonJS

AMD

ES6 Modules

CommonJS的輸出/引入

最後會有「自我評量」與「課後練習步驟(Lab)」，請各位依照說明進行練習

【Key Points】：

CommonJS

ES6 Modules

CommonJS的輸出/引入

# CommonJS

- 是一個**module**規範
- Node.js 遵循**CommonJS**規範
- Node.js內部有**Module object**
- **module**作用域獨立
- 程式執行前都會先加載**module**再執行
- **load module**同時就有**cache**

5-13



- CommonJS 並非一個 JavaScript 庫，而是一個標準化組織。
- Node.js 遵循CommonJS規範。
- module間的作用域都是獨立不互相汙染。
- 加載module的執行順序都是同步。
- 加載module的同時Node.js就會做cache，加快速度。

【Key Points】：

Node.js 遵循CommonJS規範

module間的作用域都是獨立不互相汙染

加載module的執行順序都是同步

# AMD

- 是一個**module**規範
- 採用**AMD規範的RequireJS**
- 非同步加載 **module**
- **RequireJS** 的**module**輸出與引用
- 依賴前置，提前執行

5-14



- AMD也是一個**module** 規範。
- RequireJS實現了AMD規範，所以AMD也是指RequireJS。
- 相對於CommonJS的同步加載 **module**，AMD是採非同步加載**module**。
- RequireJS透過**define**來宣告**module**，**require**引用**module**。
- 在**define**方法裡傳入的依賴模塊，會在一開始就下載並執行。

## 【Key Points】：

AMD也是一個**module** 規範

RequireJS實現了AMD規範，所以AMD也是指RequireJS

相對於CommonJS的同步加載 **module**，AMD是採非同步加載**module**

# ES6 Modules

- 是一個module規範
- 靜態編譯無法動態加載
- 可單獨加載其中的某個接口
- ES6 Modules輸出的是值的引用
- 需要使用babel來編譯

5-15



- ES6 Modules也是一個module規範。
- ES6 Modules的module不是object，它的對外接口只是一種靜態的定義。
- 與CommonJS不同，ES6 Modules可單獨加載某個module object的某接口。
- 與CommonJS不同，ES6 Modules輸出的是值的引用。
- 目前Node.js還沒有原生支持ES6 modules，所以要透過babel編譯。

## 【Key Points】：

ES6 Modules也是一個module規範

目前Node.js還沒有原生支持ES6 modules

這張投影片的內容提一下就好。請輕輕介紹帶過。

# CommonJS的輸出/引入

- 假設有一個 Node.js 模組，檔案名稱為 lib.js，內容如下  
`module.exports = { title: "title value", price: 100 }`
- 如果想要在 index.js 調用 lab.js，程式寫法：  
`var m = require('./lib.js');`
- m 的值為 { title: 'title value', price: 100 }
- 簡單地說，**module.exports** 被設定成什麼，**require()** 就傳回什麼。
- lib.js 的另一種寫法：  
`exports.title = 'title value';`  
`exports.price = 100;`
- 輸出方式有兩種：**module.exports** 和 **exports**  
引用的方式僅有一種：**require()**

5-16



假設有一個 Node.js 模組，檔案名稱為 lib.js，內容如下

```
module.exports = {  
    title: "title value", price: 100  
}
```

如果想要在 index.js 調用 lab.js，程式寫法：

```
var m = require('./lib.js');  
console.log(m);  
輸出結果: { title: 'title value', price: 100 }
```

lib.js 也可以寫成這樣：

```
exports.title = 'title value 2';  
exports.price = 200;
```

在 index.js 調用 lab.js：

```
var m = require('./lib.js');  
console.log(m); // 輸出結果: { title: 'title value 2', price: 200 }
```

【Key Points】：

輸出方式有兩種，**module.exports** 和 **exports**

使用 **exports** 在程式碼可讀性中較高

引用的方式僅有一種：**require()**

## 精彩回顧

- 了解類別的宣告、繼承、原型鏈。
- 了解箭頭函式與this。
- 了解ES6更好的寫法。
- 了解各種module規範。
- 了解怎麼輸出與引入模組。



5-17

III 財團法人資訊工業策進會  
INSTITUTE FOR INFORMATION INDUSTRY

- 每個function都有prototype屬性。
- 透過new關鍵字，可以直接宣告一個物件。
- ES6的繼承作法都是透過extends 關鍵字，繼承某類別。
- 子類別的this會指向父類別。
- Node.js 遵循CommonJS規範。
- 輸出方式有兩種，module.exports 和exports。

### 【Key Points】：

類別的宣告、繼承  
箭頭函式與this  
怎麼輸出與引入模組

## 程式練習題

- 題目一：練習 **class** 語法？

請宣告一個Plane class，並在constructor 函式中宣告name,speed 屬性，屬性值分別為'plane\_name',400。

- 題目二：改寫傳統的**function**為箭頭函式的寫法？

請將物件的 sayHi function改寫成箭頭函式

- 題目三：**exports** 語法練習

請用 exports 改寫練習描述的 module

5-18



請宣告一個Plane class，並在constructor 函式中宣告name,speed 屬性，屬性值分別為'plane\_name',400。

請將物件的 sayHi function改寫成箭頭函式

請用 exports 改寫練習描述的 module

閱讀題目說明與現有程式

在作答區修改程式

【Key Points】：

Class

箭頭函式

用 exports

## 課後練習題

- 箭頭函式的使用已相當廣泛，但其this指向卻很容易被忽略，透過此次練習加深理解。類別的使用不管是在物件導向還是架構大型後端程式都十分重要。
- Lab01:理解箭頭函式中的 this**  
請補齊程式碼，讓程式可以console.log出{ test1: [Function: test2] }
- Lab01:理解class**  
請宣告一個類別，必須有name,age屬性與一個類別方法say()
- Lab03:理解module輸出**  
承上題，請將類別輸出成一個module，另請新增另外一個js 檔案，引用該module並呼叫say()
- 參考解答請參考本模組 labFiles 資料夾的內容

Estimated time:

20 min.

5-19



### 情節描述：

箭頭函式的使用已相當廣泛，但其this指向卻很容易被忽略，透過此次練習加深理解。類別的使用不管是在物件導向還是架構大型後端程式都十分重要，再將此類別輸出成一個module。

### Lab01:理解箭頭函式中的 this

請補齊程式碼，讓程式可以console.log出{ test1: [Function: test2] }。

### Lab01:理解class

請宣告一個類別，必須有name,age屬性與一個類別方法say()

### Lab03:理解module輸出

承上題，請將類別輸出成一個module，另請新增另外一個js 檔案，引用該module並呼叫say()

參考解答請參考本模組 labFiles 資料夾的內容

### 【Key Points】：

This  
Class  
Exports

# 範例程式使用說明

- 範例程式資料夾: Module\_05\_example
- 使用步驟:
  1. 安裝 Node.js
  2. 安裝 Visual Studio Code
  3. 以 Visual Studio Code 開啟本模組的範例資料夾
  4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
  5. 在終端機視窗，輸入 node module\_05\_lab\_03answer\_part2.js

5-20



使用步驟:

1. 安裝 Node.js  
<https://nodejs.org/en/>
2. 安裝 Visual Studio Code  
<https://code.visualstudio.com/>
3. 以 Visual Studio Code 開啟範例資料夾  
在檔案總管，滑鼠右鍵點按「本範例資料夾」，從快捷功能表點選「以Code開啟」  
或者，啟動 Visual Studio Code 之後，功能表 File | Open Folder，選擇「本範例資料夾」
4. 在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗。
5. 在終端機視窗，輸入 node module\_05\_lab\_03answer\_part2.js

【Key Points】：

程式執行環境 Node.js

程式開發編輯環境: Visual Studio Code

在 Visual Studio Code 按下「Ctrl + 滑鼠右鍵」開啟 terminal 終端機視窗，輸入：「node 主程式.js」