函式 (Function)

Eddy Chang

✉️ hello@eddychang.me

函式宣告

```
// 函式定義 - 使用有名稱的函式 Function Declarations (FD)
function sum(a, b) {
  return a + b
}

// 函式表達式 - 常數指定為匿名函式 Function Expressions (FE)
const sum = function (a, b) {
  return a + b
}

// (⭐常用必會) 箭頭函式，函式表達式的縮寫 Arrow Function
const sum = (a, b) => a + b
```

函式呼叫(執行)

> 函式名稱加上圓括號(())，其中加入傳入的參數值，即可呼叫(執行)函式

```
const sum = (a, b) => a + b

sum() // undefined + undefined = NaN
sum(1, 2) // 3
```

函式傳入參數

```
// a 與 b 為函式的傳入參數
function sum(a, b) {
  return a + b
}

sum(1, 2) // 3

// (⭐常用必會) 函式傳入參數預設值 Default parameters
// 給定 a 與 b 預設值，當沒給傳入參數值時(即 undefined)會套用預設值
function sum(a = 1, b = 1) {
  return a + b
}

sum() // 2
```

函式回傳值

```
// 函式必有回傳值，沒寫就是 undefined 值
function foo() {}
function bar() {
  return
}
// return 會跳出函式中的程序運行，代表函式此次執行結束
function sum(a, b) {
  if (a === 0) return 0
  return a + b
}
// return 後可以加上表達式(一樣會執行求值)，在箭頭函式裡很常用
function log(a) {
  return console.log(a)
}
// 上面函式相當於這個箭頭函式
const log = (a) => console.log(a)
```

以函式作為傳入參數值

```
// 傳入一個函式到另一個函式中，"高階函式(HOF)"特性
const log = (a) => console.log(a)

const sum = (a, b, fn) => {
  fn(a + b)
  return a + b
}

console.log(sum(1, 2, log))
```

函式最後回傳另一個函式

```
// 函式可以最後回傳另一個函式，"高階函式(HOF)"特性
function sum(a) {
  return function (b) {
    return a + b
  }
}
// 箭頭函式簡寫整個語法
const sum = (a) => (b) => a + b
// 這裡是回傳另個函式
const sumA = sum(1)
// 這裡才會計算回傳最後結果值
sumA(2)
// 直接寫成兩個函式呼叫圓括號
sum(1)(2)
```

作用域(作用範圍)

> 註: ES6 後，作用域是以區塊({})為分界

```
const x = 1

function outer(a) {
  const y = 2
  function inner(b) {
    const z = 3
    // 這個區塊中可以存取得到所有的變數值
    console.log(a, b, x, y, z)
  }

  return inner
}

outer(99)(100) // 99 100 1 2 3
```

閉包 Closure

閉包 Closure 一種資料結構，包含函式以及記住函式被建立時當下環境。每當函式被建立時，一個閉包就會被產生(自然特性)。

- 閉包結構中所記憶的環境值是用參照指向的(傳址)
- 函式作用域連鎖規則：內部函式可以有三個作用域：1.) 自己本身的 2.)外部函式的 3.)全域的
- 閉包的記憶環境例外變數：
    - this: 執行外部函式時的 this 值
    - arguments: 函式執行時的一個隱藏偽陣列物件

callback 回調、回呼函式

```
const el = document.getElementById('myButton')

// 函式作為傳入參數的資料
el.addEventListener(
  'click',
  function () {
    console.log('hello!')
  },
  false
)
```

提升(Hoist) - 變數

> `var` 變數提升，值為 `undefined` ； `let/const` 變數提升，但有 `TDZ` 造成的參照錯誤，相當來說變數使用會更安全

```
// --- file A --------------
console.log(x) // undefined
var x = 5

// --- file B --------------
console.log(y) // ReferenceError(TDZ)
let y = 5
```

提升(Hoist) - 函式

> 一般函式定義(FD)可以正常被提升

```
foo() // valid 合法

function foo() {
  console.log('Hello1')
}
```

> 函式表達式名稱被提升，但函式主體無法提升，TDZ 特性造成參照錯誤

```
bar() // ReferenceError(TDZ)

const bar = function () {
  console.log('Hello2')
}
```

IIFE(立即呼叫函式表達式)

> 只會在定義時呼叫(執行)一次的函式

```
(function () { //... })()
(function () { //...}())
```