JavaScript 程式設計新手村

單元13 - JavaScript 函數 Function 基礎（上）

@kdchang

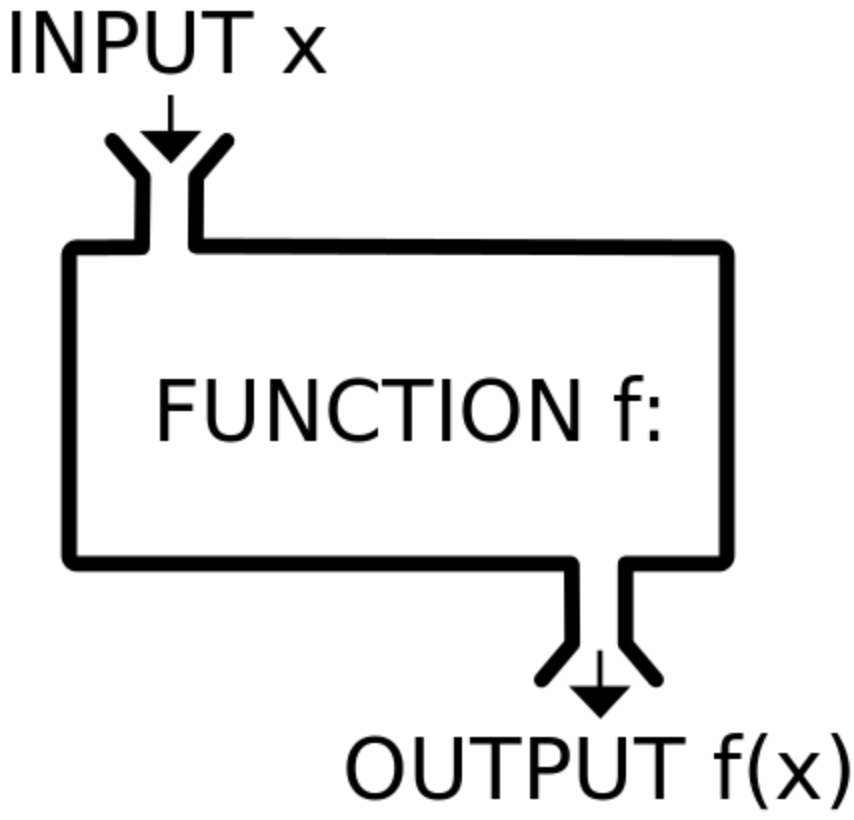
Outline

1. 函數基礎概念
2. JavaScript 常用內建函數
3. 自己建立 JavaScript 函數
4. Var hoisting、Scope Chain

函數基礎概念

函數就像個黑盒子，有輸入有輸出

但你不需要知道裡面如何運作



為什麼要寫函數？

- 當我們程式越寫越大就會碰到程式碼`重複的問題`，而函數就是要解決這個問題，將一些`共用`的程式碼集合成一個區塊，可以傳入`參數 (Parameter)`和`回傳執行結果`
- 可以想成數學上的函數，將參數傳入後，會產生結果，而那個回傳結果就是函數的 return 值
- 執行函數稱為`Call Function`，呼叫函數時不一定要傳入參數，但函數一定會有回傳值，若沒有設置則回傳`undefined`

JavaScript 常用內建函數

JavaScript 常用內建函數

1. parseInt() / parseFloat()
2. isNaN() / isFinite()
3. encodeURIComponent() / decodeURIComponent()
4. eval()
5. alert() / confirm() / prompt()

parseInt() / parseFloat()

- parseInt('數值', 基數)
  返回由第二個參數所指定的 radix （基數）的整數
- parseFloat()
  返回浮點數，不過只支持10進位

isNaN() / isFinite()

- isNaN() 透過 `isNaN()` 可以得知傳入的數值是否是非數字，也可以同時知道是否能運用 `parseInt()` 、 `parseFloat()` 成功
- `NaN` 不存在等值的概念，故 `NaN == NaN` 會 `return false`
- isFinite() 檢驗傳入數字是否為一既非 infinity 且非 NaN的數字

URI 編碼/解碼

根據 Wiki 表示：統一資源標識符（Uniform Resource Identifier，或URI)是一個用於標識某一網際網路資源名稱的字元串，而URL (定義位置) 和 URN (定義身份) 則是URI的子集

在處理網頁(址)資料，常會有遇到中文字或是空白、標點符號等問題(英文、數字通常不會有問題) ，此時就可以使用 URI 編碼函數(通常標點轉為十六進位ASCII、中文轉成十六進位統一字元碼)，如需轉回則使用解碼函數

encodeURIComponent() / decodeURIComponent()

1. escape()
   除了ASCII Code、數字、英文字和符號包括: @ * / + 不編碼外，其餘都編碼，如果想對URI編碼，建議不用
2. encodeURI() 不編碼符號包括: ~ ! @ # $ & * ( ) = : / , ; ? + '
   URI中的合法字符都不會被編碼，適合用於網址編碼
3. encodeURIComponent() 不編碼符號包括: ~ ! * ( ) '
   適合參數編碼，應用範圍廣

```
console.log(decodeURIComponent('%E7%A8%8B%E5%BC%8F%E8%A8%AD%E8%
```

eval()

eval()函數可以將運算式的字串當做運算式，傳回運算式的計算結果。但使用上有安全顧慮(eval is evil)，特別是用它執行第三方的JSON數據(其中可能包含惡意代碼)，盡量少用

```
eval(1+4); // 5
```

alert() / confirm() / prompt()

alert()、confirm()、prompt () 三者皆會在瀏覽器顯示對話框，但用處各自不同，其中 confirm("text") 和 alert("text") 類似，都會把參數內的字串顯示在對話視窗中，但confirm()有確定和取消，會回傳 `true` 或 `false`

```
const name = prompt("請問你叫什麼名字呀？", "使用者未輸入的預設值"
 //若取消則回傳 null
```

練習一

使用 prompt 和 if else 建立一個判斷是否能投票的程式

自己建立 JavaScript 函數

主要函數建立方法

1. 定義命名函數（常見用法）

```
function sum(a, b){
  return a + b;
}
```

2. 定義匿名函數（函數可以當做變數傳遞）

```
var sum = function(a, b){
   return a + b;
};
```

3. Function 物件匿名函數（不建議使用）

```
var sum = new Function('a', 'b', '{return a + b }');
```

JavaScript 自定函數執行

```
function sum(x, y) {
        return x + y;
}

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

Var hoisting、Scope Chain

JavaScript 函數執行和 hoisted

一般而言，JavaScript 有內建函數和自定義的函數。在 JS 中函數在被執行之前會被解析（hoisted），因此它可以在任意的地方都是有宣告的，即便在比這個函式還早呼叫

```
console.log(sum(1, 2));
function sum(a, b) { //定義了傳入參數
    var c = a + b;      //執行內容
    return c;      //回傳值，若未設定回傳undefined
}
console.log(sum(1, 2));
```

Var hoisting

JavaScript 是 function scope ，無論你在函數內那裏宣告變數 `var` ，都會提升到最前面宣告，稱為變數的拉升(Variable Hoisting)

建議把變數的宣告放在函數的最頂端。否則可能導致混亂的情況，或是使用 `let` 可避免這種狀況

```
function func() {
    console.log(a); // 儘管還沒定義但會回傳 undefinded
        var a = 1;
}

func();
```

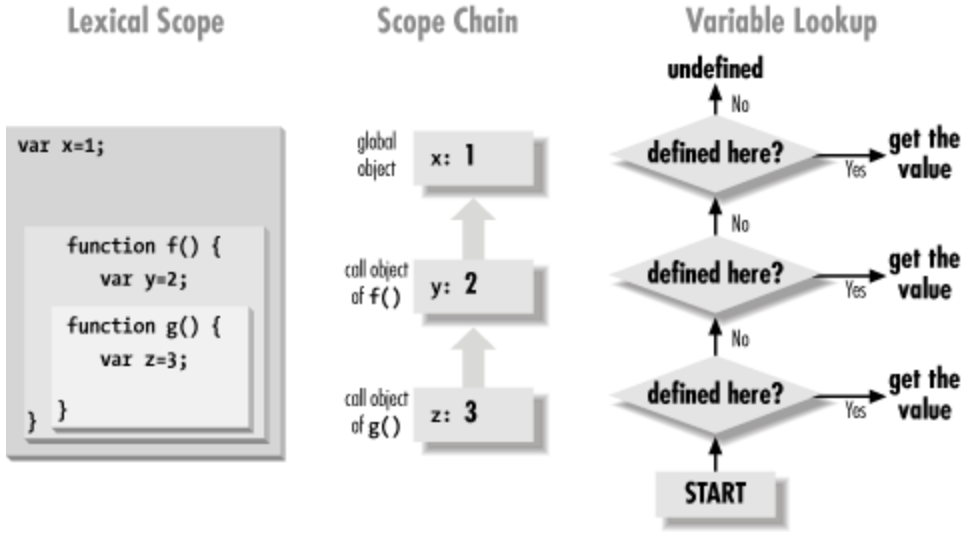
Scope Chain

用 `var` 所宣告的變數，作用範圍是在當時所在環境(函數內)，而不使用 `var` 直接指定值而建立的變數，則是全域物件（window）上的一個屬性，也就全域範圍

在 JS 中有稱作 `scope chain` (範圍鏈)的特性，JS在查找變數時，會循著範圍鏈（Scope Chain）一層一層往外找，若函數內找不到，則往外找

注意：內層函式都可以存取外部函式的變數

Scope Chain


Lexical Scope
Scope Chain
Variable Lookup
var x=1;
function f() {
var y=2;
function g() {
var z=3;
}
}
global object
x: 1
call object of f()
y: 2
call object of g()
z: 3
undefined
No
defined here?
Yes
get the value
No
defined here?
Yes
get the value
No
defined here?
Yes
get the value
START

練習二

運用 function 製作一華氏攝氏溫度轉化器，輸入值為攝氏，回傳值為華氏，轉換公式： F = (9 . 0 * C) / 5 . 0 + 32 . 0

總結

在這個章節中我們了解了：

1. 函數基礎概念
2. JavaScript 常用內建函數
3. 自己建立 JavaScript 函數
4. Var hoisting、Scope Chain

image via mit