JavaScript 程式設計新手村

單元14 - JavaScript 函數 Function 基礎（下）

@kdchang

Outline

1. 自調用函數
2. closure 閉包
3. JavaScript 函數式程式設計（map、filter、reduce）

自調用函數/立即函數

自調用函數/立即函數

`Self-invoking functions/Immediately-Invoked Function Expression (IIFE)` （自調用函數/立即函數）是一種，不用額外呼叫可自己立刻執行，方便建構自己的生存域

```
(function (name) {
        var cat = name;
        document.write(cat);
})('momo');

alert(cat); //undefined
```

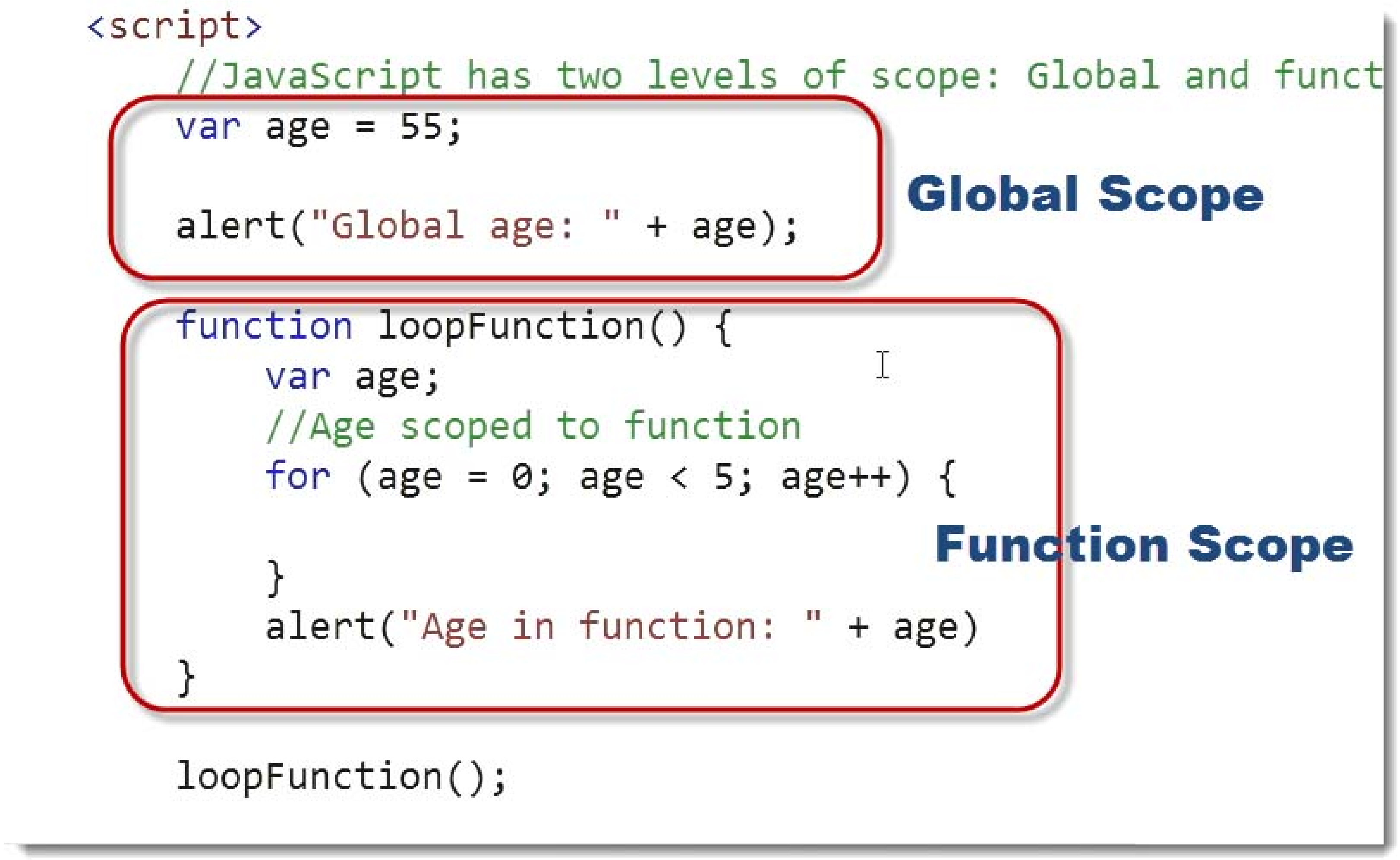
jQuery Plug-in

```html
<p id="idName">Hello jQuery</p>
<script src="https://code.jquery.com/jquery-3.1.0.js"></script>
```

```javascript
(function($) {
    $.fn.setColor = function() {
        // 私有變數
        var shade = "red";
        this.css( "color", shade );
        // 回傳選取元素 jQuery 物件
        return this;
    };
}(jQuery));

$('#idName').setColor();
```

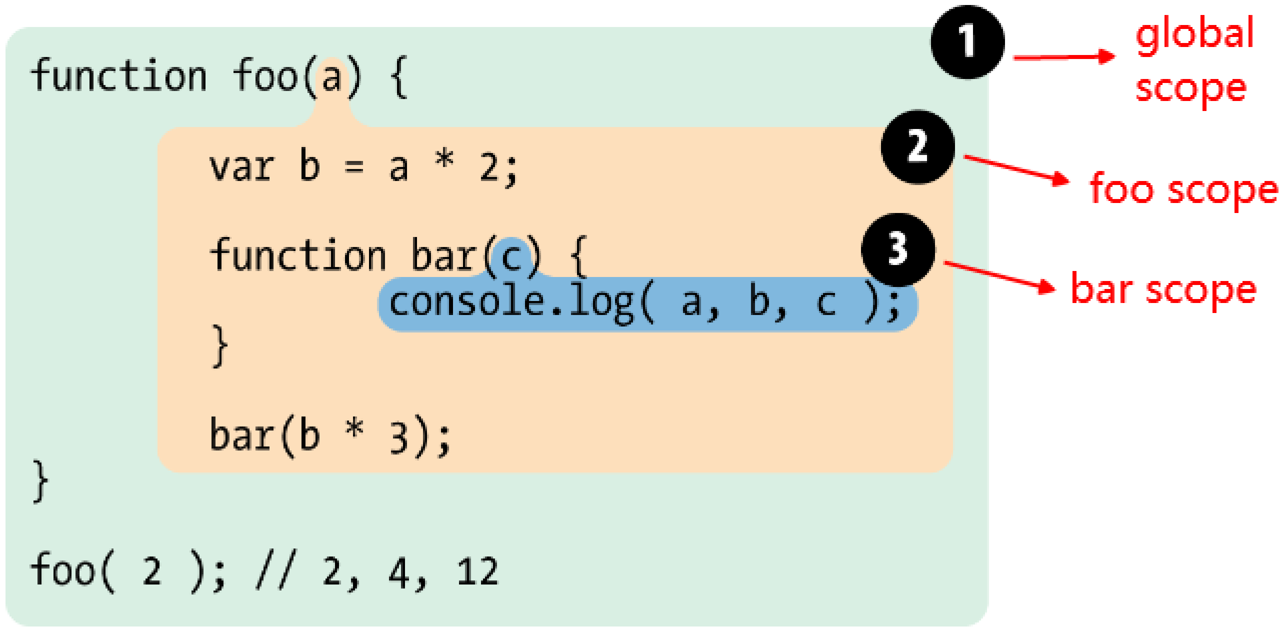
closure 閉包

function scope



```
<script>
    //JavaScript has two levels of scope: Global and funct
    var age = 55;

    alert("Global age: " + age);

    function loopFunction() {
        var age;
        //Age scoped to function
        for (age = 0; age < 5; age++) {

        }
        alert("Age in function: " + age)
    }

    loopFunction();
```

function scope

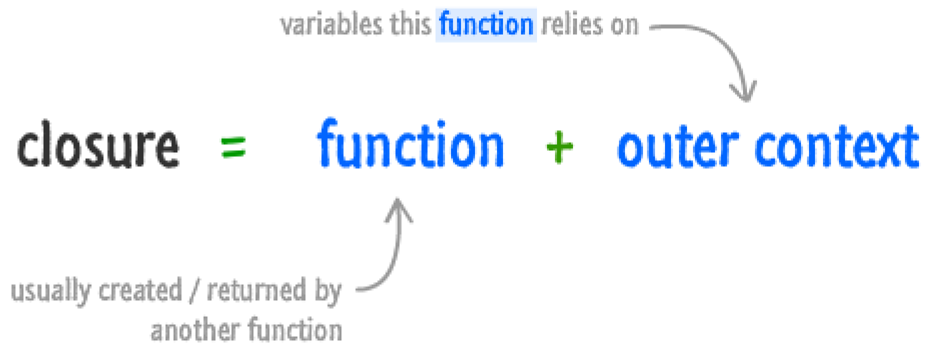


```
function foo(a) {

        var b = a * 2;

        function bar(c) {
                console.log( a, b, c );
        }

        bar(b * 3);
}

foo( 2 ); // 2, 4, 12
```

closure 閉包

閉包（Closure）是擁有閒置變數（Free variable）的物件。建立函數不等於建立閉包。如果函數的閒置變數與當時語彙環境綁定，該函數才稱為閉包

閒置變數是指對於函式而言，既非區域變數也非參數的變數


variables this function relies on
closure = function + outer context
usually created / returned by another function

延伸閱讀：閉包（Closure）

closure 閉包範例

```
function makeFunc() {
  var name = "Mozilla";
  function displayName() {
    alert(name);
  }
  return displayName;
}

// closure，理論上 name 在函數執行完就消失，
// 但由於內部函數 displayName 參考到 name 變數，所以當 displayName
// 生存域突破成全域，記憶了創建函數時的環境變數參考，所以 name 活下

var myFunc = makeFunc();
myFunc();
```

延伸閱讀：MDN Closures

closure 閉包範例

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

```
function a() {
    var i = 0;
    function b() {
        alert(++i);
    }
    return b;
}
var c = a();
c();
```

closure 應用

1. 嵌套 callback 函數、非同步處理（Ex. 事件處理）
2. 實現 private

closure 模擬 private

使用閉包來定義公共函數，且其可以訪問私有函數和變數。這個方式也稱為模組模式（module pattern）

範例程式

closure 閉包應用嵌套 callback 函數

```html
<a href="#" id="size-12">12</a>
<a href="#" id="size-14">14</a>
<a href="#" id="size-16">16</a>
```

```javascript
function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size + 'px';
  };
}

// 每個函數的創建都會有自己獨特生存環境
var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);

document.getElementById('size-12').onclick = size12;
document.getElementById('size-14').onclick = size14;
document.getElementById('size-16').onclick = size16;
```

closure 閉包處理非同步問題

非同步處理問題（callback 只看到迴圈最後結果）：

```
for(var i = 0; i < 5; i++){
  setTimeout(function() { console.log(i); }, 1000);
}
```

使用 closure（每次創建函數都會記憶獨立生存空間）：

```
function printLog(i){
  // closure
  return function(){
    console.log(i);
  }
}

for(var i = 0; i < 5; i++){
  setTimeout(printLog(i), 1000);
}
```

箭頭函數 (Arrow Function)

箭頭函數 (Arrow Function) 是 ES6 之後可以使用的功能，主要可以解決 `this context` 綁定和讓程式碼更為簡潔，一方面也發揮 JavaScript 函數式程式設計的特性

延伸閱讀：MDN 箭頭函數 (Arrow Function)

箭頭函數 (Arrow Function)

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// 等於 :  => { return expression; }

// 只有一個參數時,括號才能不加:
(singleParam) => { statements }
singleParam => { statements }

//若無參數,就一定要加括號:
() => { statements }
```

箭頭函數 (Arrow Function) 看起來更簡潔：

```
const a = [
  "Hydrogen",
  "Helium",
  "Lithium",
  "Beryllium"
];

// map 為陣列方法，會迭代陣列內容
var a2 = a.map(function(s){ return s.length });

var a3 = a.map( s => s.length );
```

JavaScript 函數式程式設計（map、filter、reduce）

map 迭代函數

```
const numbers = [1, 2, 3, 4];

const newNumbers = numbers.map(function(number, index){
    return number * 2;
});

console.log("The doubled numbers are", newNumbers); // [2, 4, 6
```

filter 過濾函數

```
const numbers = [1, 2, 3, 4];

const newNumbers = numbers.filter(function(number){
    return (number % 2 !== 0);
}).map(function(number){
    return number * 2;
});

console.log("The doubled numbers are", newNumbers); // [2, 6]
```

reduce 累計函數

```
var numbers = [1, 2, 3, 4];

var totalNumber = numbers.reduce(function(total, number){
    return total + number;
}, 0);

console.log("The total number is", totalNumber); // 10
```

總結

在這個章節中我們了解了：

1. 自調用函數
2. closure 閉包
3. JavaScript 函數式程式設計（map、filter、reduce）