JavaScript 進階

問題解答

p.13 實作練習一(1)

```
// Deep Copy
// 作法一： Object 逐一賦值
var peter = { name: 'Peter', data: { gender: 'male' } };
var mary = { name: peter.name, data: peter.data };
mary.name = 'Mary';
mary.data.gender = 'female';
console.log(peter);
console.log(mary);

// 作法二： Object.assign() 指定新值
var peter = { name: 'Peter' };
var john = Object.assign({}, peter);
john.name = 'John';
console.log(peter);
console.log(john);

// 作法三： 轉JSON
var peter = { name: 'Peter', greet: function () { return 'Hi'; } };
var mary = JSON.parse(JSON.stringify(peter));
mary.name = 'Mary';
console.log(peter);
console.log(mary);

// 作法四： 使用lodash
var _ = require('lodash'); // 這是nodeJS的寫法

var peter = { name: 'Peter', greet: function () { return 'Hi'; } };
var mary = _.cloneDeep(peter);
mary.name = 'Mary';
console.log(peter);
console.log(mary);
```

p.13 延伸思考

延伸思考：如果我不能用Lodash呢？

答：如果不能用Lodash，我們可以使用es6中提供的WeakMap物件。

```
// 使用WeakMap以弱關聯(weak reference)方式儲存物件
function deepCopy(obj, cache = new WeakMap()) {
    // primitive type 和 function
    if (obj == null || typeof (obj) !== 'object') {
        return obj;
    }
    // 特別處理正則表示式，取得建構式並重新建構物件
    if (obj instanceof RegExp) {
        return obj.constructor(obj);
    }
    // 特別處理 Date，重新建構物件
    if (obj instanceof Date) {
        return new Date(obj);
    }
    // 如果WeakMap已經有此物件，則直接取出回傳
    if (cache.has(obj)) {
        return cache.get(obj);
    }
    // 依原始物件型別建立一新物件，並放入WeakMap以建立新物件
    const copy = new obj.constructor;
    cache.set(obj, copy);
    // 所有的屬性和Symbol依序遞迴拷貝
    [...Object.getOwnPropertyNames(obj), ...Object.getOwnPropertySymbols(obj)].forEach(key => {
        copy[key] = deepCopy(obj[key], cache);
    });

    return copy;
}

var a = { name: 'Peter', func: function () { } };

console.log(a);  { name: 'Peter', func: [λ: func] }
console.log(deepCopy(a));  { name: 'Peter', func: [λ: func] }
console.log(a === deepCopy(a));  false
```

p.18 問題思考

為什麼我們要用到IIFE？在什麼樣的情境下我有必要立刻執行一個函式？

答：因為使用IIFE可以做到立即切割範疇(Scope)，讓變數不要去污染全域環境

```
26  var name = 'Peter';
27  (function sayHi() {
28      var name = 'Joseph';
29      console.log('Hi, ' + name);  Hi, Joseph
30  }());
31  console.log(name);  Peter
```

補充：

在JavaScript中，使用外部套件是非常常見的一件事，但載入外部套件其實就是合併各套件的程式碼，根據JS單緒執行的語言特性，非常容易因為載入順序導致對全域變數的污染或錯誤，而IIFE可以避免這種錯誤，所以幾乎所有函式庫都會用IIFE保護自己和使用者的程式碼。

p.27 問題思考

每一個function都可以有回傳值：

1. 如果我在function內另外寫一個回傳物件，new的執行結果會是什麼？

答：會以回傳物件(return)為主

```
function Person() {
    this.name = 'Peter';
    this.age = 25;

    return {
        name: 'John',
        age: 20
    }
}

console.log(new Person());  { name: 'John', age: 20 }
```

2. 如果回傳的不是物件呢，結果又會是什麼？

答：會以函式建構式為主

```
function Person() {
    this.name = 'Peter';
    this.age = 25;

    return 'a';
}

console.log(new Person());  Person { name: 'Peter', age: 25 }
```

記法：new 關鍵字一定要回傳物件

p.28 實作練習一(2)

```
1   function Singleton() {
2
3       // 如果不用new，this會指向全域物件
4       if (!(this instanceof Singleton)) {
5           throw new Error('請用new建立物件');   請用new建立物件
6       }
7
8       // function也是物件，取得function的instance屬性
9       var instance = Singleton.instance;
10      if (typeof (instance) == 'object') {
11          return instance;
12      }
13
14      // this代指new出來的物件，將它存在function的instance屬性中
15      Singleton.instance = this;
16  }
17
18  var s1 = new Singleton();
19  var s2 = new Singleton();
20  console.log(s1 === s2);  true
21  var s3 = Singleton();  請用new建立物件
```

p.33 實作練習一(3)

```
var john = {
    name: 'John',
    age: 20,
    greet: function () {
        return 'Hi, ' + this.name;
    }
}

// 使用Object.create(Obj)，以Obj為原型並創新物件
var jason = Object.create(john);
jason.name = 'Jason';
jason.gender = 'Male';

var jack = Object.create(jason);
jack.name = 'Jack';

console.log(jack.age);  20
console.log(jack.gender);  Male
console.log(jack.greet());  Hi, Jack
```

```
var john = {
    name: 'John',
    age: 20,
    greet: function () {
        return 'Hi, ' + this.name;
    }
}

var jack = {
    name: 'Jack'
}

var jason = {
    name: 'Jason',
    gender: 'male'
}

// 使用Object.setPrototypeOf方法，給予物件設定原型
Object.setPrototypeOf(jack, jason);
Object.setPrototypeOf(jason, john);

console.log(jack.age);  20
console.log(jack.gender);  male
console.log(jack.greet());  Hi, Jack
```

p.43 案例探討

會印出 5個 5和許多JS的特性有關，首先，因爲閉包的緣故function裡的 i 會記憶並且參照宣告在function外的 i，導致每一次 i 的值改變都會連動影響。再者，因為setTimeout這個API是屬於非同步執行的，也就是說當下執行的時候是註冊至事件佇列，而不是馬上運行而印出當下的 i 值。綜上所述，setTimeout的function註冊了 i 的內容，因為形成閉包 (相同範疇) 的緣故，連動影響到了每一次註冊的內容。

你以為是這樣

```
setTimeout(function () {
    console.log(0);
}, 1000);

setTimeout(function () {
    console.log(1);
}, 1000);

setTimeout(function () {
    console.log(2);
}, 1000);

setTimeout(function () {
    console.log(3);
}, 1000);

setTimeout(function () {
    console.log(4);
}, 1000);
```

```
setTimeout(function () {
    console.log(i);
}, 1000);

setTimeout(function () {
    console.log(i);
}, 1000);

setTimeout(function () {
    console.log(i);
}, 1000);

setTimeout(function () {
    console.log(i);
}, 1000);

setTimeout(function () {
    console.log(i);
}, 1000);
```

實際上是這樣

p.52 觀念測驗

這段程式碼不難，我們首先看到this中提過的明確綁定關鍵字call，所以會傳入一個物件：{ x: 'outer' }作為this，陣列的第二個參數無庸置疑會是'outer'，但第一個參數照理說也用明確綁定關鍵字bind，應該要是'inner'才對，但別忘記箭頭函示的this是遵循詞彙環境的，明確綁定無作用，故兩者皆為 'outer'。

```
let rtnArr = (function () {
    return [
        (() => this.x).bind({ x: 'inner' })(),
        (() => this.x)()
    ]
}).call({ x: 'outer' });

console.log(rtnArr);  [ 'outer', 'outer' ]
```

p.56 實作練習一(4)

改成以function作為函式建構式，請注意sayHi是直接作為john物件的屬性，而以class宣告則會將所有函式寫入原型鍊中。

```
function Person(name) {
    this.name = name;
    this.sayHi = () => {
        return `Hi, my name is ${name}.`;
    }
}

let john = new Person('John');
console.log(john);  Person { name: 'John', sayHi: [λ] }
console.log(john.sayHi());  Hi, my name is John.
```

p.69 問題思考

Promise的then方法串連彈性較大，可以針對不同的情境做處理，而如果是async和await方法在進入reject的情境的時後，則會視同錯誤，所以我們以try、catch語法進行處理。

```
(async () => {
    console.log('start');
    let result;
    try {
        result = await promiseA();
        console.log(result);
        result = await promiseB();
        console.log(result);
        result = await promiseC();
        console.log(result);
    } catch (err) {
        console.log(err);
    }

})();
```

p.80 實作練習一(5)

請使用前面提到的陣列方法，以55頁提供的studentList作為原始資料，完成以下幾件事：

1. 取得所有男性的學生的年齡(age)加總，以數字回傳

```
console.log(  102
    studentList.filter(x => x.gender == 'male')
        .map(x => x.age)
        .reduce((x, y) => x + y)
);
```

2. 取得所有學號為偶數的女性學生姓名，以字串回傳 (名字間以逗號間隔)

```
console.log(  Emma,Gloria
    studentList.filter(x => x.gender == 'female' && !(x.number % 2))
        .map(x => x.name).join()
);
```

p.81 實作練習一(6)

請使用你學會的陣列方法，完成以下的arrayDiff方法。

要求：source為來源陣列，remove為需要過濾掉的元素陣列，請將source中每一筆remove有包含的元素移除後回傳

```
function arrayDiff(source, remove) {
    remove.forEach(x => {
        source = source.filter(y => y != x);
    });
    return source;
}

let rtnArr = arrayDiff([1, 2, 2, 3, 2, 4, 5], [2]);
console.log(rtnArr);  [ 1, 3, 4, 5 ]
```

說明：以移除項陣列作為基準，用forEach取得每一項元素，逐項作為filter的過濾條件

p.84 觀念測驗

只有e是純函式，其餘都有副作用。

```
let name = 'Peter';

let a = name => {
    console.log(name); // console.log為 I/O，影響外部狀態
    return name;
}

let b = newName => {
    name = newName;     // 改變外部全域變數內容
    return name;
}

let c = id => document.getElementById(id).value; // 調用DOM元素

let d = url => axios.get('https://jsonplaceholder.typicode.com/users')
    .then(response => { // 發出HttpRequest
        console.log(response.data.map(x => x.name));
    }
);

let e = name => `Hello, ${name}.`; // Pure Function
```

p.89 實作練習一(7)

將調整的函式以柯里化作為第一層封裝，接受傳入的函式作為調整的計算標準。

```
// 將調分方式以柯里化包裝
let adjustWithCurry = method => arr => arr.map(method);

// 全體加十分
let adjustAddTen = adjustWithCurry(x => x + 10);
console.log(adjustAddTen(originalScores));  [ 70, 80, 85, 80, 90, 90 ]
// 全體加一成
let adjustTenPercent = adjustWithCurry(x => x * 1.1);
console.log(adjustTenPercent(originalScores));  [ 66, 77, 82.5, 77, 88, 88 ]
```