

# 陣列 (Array)

Eddy Chang

✉ [hello@eddychang.me](mailto:hello@eddychang.me)



## 建立、頭尾加入新成員

```
// 陣列為有序資料集合  
// 以"常數"來宣告文字字面，索引自0開始由左往右遞增0,1,2...  
const aArray = []  
  
// (★常用必會) `[]` <-- 1` 從後面加入新成員，原陣列 => [1]  
aArray.push(1)  
  
// `2` --> []` 從前面加入新成員，原陣列 => [2, 1]  
aArray.unshift(2)
```

## 頭尾移除成員

```
const aArray = [1, 2, 3]
```

```
// `[] --> 3` 從後面移除成員，回傳3。原陣列 => [1, 2]
```

```
let popValue = aArray.pop()
```

```
// `1 <-- []` 從前面移除成員，回傳1。原陣列 => [2]
```

```
let shiftValue = aArray.shift()
```

## 存取成員

```
const aArray = [1, 2, 3]
```

```
// 用索引存取成員值(索引自0開始由左往右遞增0,1,2...)
```

```
const a = aArray[2]
```

```
// 對成員指定一個新值，也是使用方括號([])，使用索引來存取成員的值
```

```
aArray[2] = 99
```

```
// 解構賦(指定)值語法 (destructuring assignment)
```

```
// 類似鏡子的1對1映對作指定運算，把陣列中的資料解開擷取成為獨立變數
```

```
// 執行後結果`x=1, y=2, z=3`
```

```
const [x, y, z] = aArray
```

## 對每個成員處理 - for

★ 常用必會

範例：對每個成員乘 2，最後要得到 [2, 4, 6, 8]

使用 for 迴圈語法，自索引為 0 開始依序遞增處理

```
const aArray = [1, 2, 3, 4]

// for 迴圈
for (let i = 0; i < aArray.length; i++) {
  aArray[i] = aArray[i] * 2
}
```

## 對每個成員處理 - forEach

使用 forEach 迭代語法，由陣列呼叫開始，依次輪流進入其中的回調(呼)函式中處理

```
const aArray = [1, 2, 3, 4]

// forEach 迭代(iteration)
aArray.forEach(function (v, i, array) {
  // callback function(回調、回呼函式)
  // 每個成員的索引、值，會依序在這區塊裡面被得到，視情況運算執行
  array[i] = v * 2
})
```

## 對每個成員處理 - map

★ 常用必會


使用 map 迭代語法，由陣列呼叫開始，依次輪流進入其中的回調(呼)函式中處理，之後回傳處理過的值。map 會終會回傳一個新陣列。

```
const aArray = [1, 2, 3, 4]

// map 迭代(iteration)
const bArray = aArray.map(function (v, i, array) {
  return v * 2
})
```

# callback 是什麼

回調或回呼函式 callback function，常簡稱為 callback，「是一個函式以傳入參數值的方式，傳遞到另一個函式中，準備接下來要繼續執行」

- 函式(function)在 JS 中有**頭等函式**與**高等函式(HOF)**的設計
- 與 JS 中一種程式碼編寫風格 CPS(延續傳遞風格) 有關，CPS 主要是用來解決阻塞，也就是要達成非阻塞(non-block)、可異步(asnyc)執行
-  並非所有的 callback 都是異步執行的

註：**CPS - 延續傳遞風格(Continuation-passing style)**



## 拷貝陣列成員(淺拷貝 shallow copy)

```
const aArray = [1, 2, 3, 4]
```

```
// (★常用必會) 方式一：ES6 - 展開運算符
```

```
const bArray = [...aArray]
```

```
// 方式二：slice
```

```
const cArray = aArray.slice()
```

## 組合多個陣列為一個陣列

```
const aArray = [1, 2, 3]  
const bArray = [99, 100]
```

```
// 用展開運算子為最方便  
const cArray = [...aArray, ...bArray]
```

## 清空陣列

```
// 方式一：使用length屬性(又稱truncate(截短)語法)
```

```
const aArray = [1, 2, 3, 4]
```

```
aArray.length = 0
```

```
// 方式二：直接指定空白陣列
```

```
// ⚠ 注意要用 let 宣告才行
```

```
let bArray = [7, 8, 9, 10]
```

```
bArray = []
```

## 尋找成員

(★ 常用必會)

```
const aArray = [1, 2, 3, 4]

// find index, 如果有找到回傳索引, 沒找到會回傳 -1
const index = aArray.findIndex((v, i, array) => v === 2)

// find value, 如果有找到回傳值, 沒找到會回傳 undefined
const value = aArray.find((v, i, array) => v === 2)
```

註: 另有 `indexOf` 方法, 只能用來尋找一般基礎值(數字/字串/布林...), 所以用途很有限。

## 分割一個陣列為多個 - slice

語法: `slice(開始索引, [結束索引(不包含)])`

```
const aArray = [1, 2, 3, 4]
// slice(start, [end(不包含)]) 會回傳一個新陣列
// 結束索引沒給定時，會分割到最後一個。結束索引負數-1 代表從最後面算起第 1 個
// 得到 [1, 2]
const bArray = aArray.slice(0, 2)
// 得到 [2, 3]
const cArray = aArray.slice(1, -1)
// 得到 [3, 4]
const dArray = aArray.slice(2)
```

## 從中移除成員 - filter

(★ 常用必會)

```
const aArray = [1, 2, 3, 4]
```

```
// 移除 index=1 -> 相當於新建立一個陣列不要有索引 1 的成員
```

```
const bArray = aArray.filter((v, i, array) => i !== 1)
```

```
// 移除 value=3 -> 相當於新建立一個陣列不要有值 3 的成員
```

```
const cArray = aArray.filter((v, i, array) => v !== 3)
```

## 從中移除成員 - splice

⚠ 有副作用，會改變呼叫它的陣列

語法: `splice(要移除的成員索引, 1)`

```
const aArray = [1, 2, 3, 4]
// 移除自索引 2 的成員，第二傳入參數1代表移除一個
// 結果: `aArray = [1,2,4]`
aArray.splice(2, 1)
```

## 從中插入新成員 - slice

```
const aArray = [1, 2, 3, 4]

// 範例1：在索引 1 處插入新的成員 99，其它成員往後
// 先用slice切割出兩個陣列
const bArray = aArray.slice(0, 1)
const cArray = aArray.slice(1)

const newArray = [...bArray, 99, ...cArray]

// 範例1：在索引 1 處插入一個陣列的中的值
const newArray = [...bArray, ...[99, 100, 101], ...cArray]
```



## 從中插入新成員 - slice 函式

```
// insertItem(aArray, 1, 99)
const insertItem = (arr, index, newItem) => [
  ...arr.slice(0, index),
  newItem,
  ...arr.slice(index),
]

// insertItems(aArray, 1, [99, 100, 101])
const insertItems = (arr, index, newItems) => [
  ...arr.slice(0, index),
  ...newItems,
  ...arr.slice(index),
]
```

## 從中插入新成員 - splice

⚠ 有副作用，會改變呼叫它的陣列

語法: `splice(要插入值在它後面的成員索引, 0, [值])`

```
const aArray = [1, 2, 3]
```

```
// 範例1：在索引 2 後插入新的成員 99，其它成員往後
```

```
// 結果：aArray = [1, 2, 999, 3]
```

```
// splice 第二傳入參數要保持為0，它是刪除成員用的，在這不用這個。
```

```
aArray.splice(2, 0, 99)
```

# 排序成員

⚠ 有副作用，會改變呼叫它的陣列

```
const aArray = [2, 1, 3, 4, 5]
```

```
// 由小到大排序(只針對數字)
```

```
aArray.sort(function (a, b) {  
  return a - b  
})
```

```
// 由大到小排序(只針對數字)
```

```
aArray.sort(function (a, b) {  
  return b - a  
})
```

## 與字串交互應用

```
const aString = '1,2,3,4,5'  
const aArray = myString.split(',')
```

```
const bArray = ['a', 'b', 'c']  
const bString = bArray.join('-')
```