工作時經常用到的 JS 知識點及技巧(3)－陣列的處理

Via 134340號小行星 by pluto • Dec 28, 2021

如果有看過 MDN 介紹 Array 的部分應該就會知道在 JavaScript 中有超級多處理 Array 的方法，但實際應用來說不可能全部都會用到，有些甚至幾乎沒用到過，既然這篇文章是工作常用系列，那我就只會介紹一部分我比較常使用到的方法，其他的就要請大家到 MDN 瀏覽啦。

> Array – MDN

Array方法分類

這邊簡單整理一個 Array 方法的表格：

類別	方法
會改變原始陣列	copyWithin()、fill()、pop()、push()、reverse()、shift()、sort()、splice()、unshift()
產生新陣列或新值	concat()、entries()、filter()、flat()、from()、join()、keys()、map()、of()、reduce()、slice()、toString()、values()
用於判斷並回傳布林值	every()、includes()、isArray()、some()
回傳元素值或索引值	find()、findIndex()、indexOf()、lastIndexOf()
遍歷執行方法(回傳`undefined`)	forEach()

方法依 MDN 介紹順序排序

map()

`map()` 應該和 `forEach()` 並列為最常用的 array 方法第一，起碼對我來說是這樣的。因為我不喜歡寫 for 迴圈所以基本上都是用 map 來處理遍歷的情形，但如果只是要遍歷元素執行 function 那就會用 `forEach()`。

map() 的最基本使用方式如下，可以直接用箭頭函數回傳陣列中的元素：

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.map(item => item)); // [1, 2, 3, 4, 5]

// [].map((element, index, array) => {})

const arr = [1, 2, 3, 4, 5];
console.log(arr.map((element, index, array) => {
 console.log(element, index, array);
}));
/*
  1, 0, [1, 2, 3, 4, 5]
  2, 1, [1, 2, 3, 4, 5]
  3, 2, [1, 2, 3, 4, 5]
  4, 3, [1, 2, 3, 4, 5]
  5, 4, [1, 2, 3, 4, 5]
*/
```

從上方例子可以看出 map 後回傳的陣列和原本的 arr 長的一模一樣，但是做嚴格比對後就會發現它們並不相同，因為每個物件都是獨立存在的實體，兩者的記憶體位址並不相同（物件比較的是記憶體位址而不是值）。

```
const arr = [1,2,3,4,5];
const arr2 = arr.map(item => item);
console.log(arr === arr2); // false
console.log(arr === arr.map(item => item)); // false
```

那我們怎麼知道 map 是回傳新的陣列還是改變原本的陣列呢？我們可以看下方的例子：

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.map(item => item + 1)); // [2, 3, 4, 5, 6]
console.log(arr); // [1, 2, 3, 4, 5]
```

若改變原本的陣列，`console.log(arr)` 時輸出的應該是 `[2, 3, 4, 5, 6]` 而非 `[1, 2, 3, 4, 5]`，因此我們可以得知 map 回傳的是新的陣列。

提示

因為 `map()` 會回傳新的陣列，所以希望遍歷元素執行方法時可以改用 `forEach()` 或 `for…of`。

forEach()

`forEach()` 方法會將陣列內的每個元素，皆傳入並執行給定的函式一次。

```
const arr = [1, 2, 3, 4, 5];
const result = arr.forEach(item => item);
const result2 = arr.forEach(item => {
 console.log(item);
});
console.log(result); // undefined
console.log(result2); // 1  2  3  4  5

// [].forEach((element, index, arr) => {})

const arr = [1, 2, 3, 4, 5];
console.log(arr.forEach((element, index, arr) => {
 console.log(element, index, arr);
}));

/*
1, 0, [1, 2, 3, 4, 5]
2, 1, [1, 2, 3, 4, 5]
3, 2, [1, 2, 3, 4, 5]
4, 3, [1, 2, 3, 4, 5]
5, 4, [1, 2, 3, 4, 5]
*/
```

其實和使用 for 迴圈是一樣的效果：

```
const arr = [1, 2, 3, 4, 5];

for(let i = 0; i < arr.length; i++) {
 console.log(arr[i]);
}

arr.forEach(item => console.log(item));
```

如果遍歷的 array 中有留空的元素，會自動填上 `undefined`：

```
[1, 2, , , 5].forEach((item, index, arr) => console.log(item, index, arr));

/*
  1, 0, [1, 2, undefined, undefined, 5]
  2, 1, [1, 2, undefined, undefined, 5]
  5, 4, [1, 2, undefined, undefined, 5]
*/
```

filter()

filter 也是在處理array資料時非常常用到的一個方法，用於篩選陣列中需要的元素成組新的陣列：

```
const arr = [1, 2, 3, 4, 5];
console.log(arr.filter(item => item >= 2)); // [2, 3, 4, 5]

// [].filter((item, index, arr) => {})
arr.filter((item, index, arr) => {
 console.log(item, index, arr)
})
/*
  1, 0, [1, 2, 3, 4, 5]
  2, 1, [1, 2, 3, 4, 5]
  3, 2, [1, 2, 3, 4, 5]
  4, 3, [1, 2, 3, 4, 5]
  5, 4, [1, 2, 3, 4, 5]
*/
```

最常應用的情形應該就是搜索了，使用 `filter()` 篩選出搜索後的結果：

```
const name = ["Tom", "Jerry", "Yeon", "Kevin"];

function filterItems(keyword) {
 return name.filter(item => item.toLowerCase().indexOf(keyword.toLowerCase()) > -1);
}

console.log(filterItems("E")); // ["Jerry", "Yeon", "Kevin"]
```

join()

`join()` 也是滿實用的一個方法，可以將陣列的元素合併成一個字串，並用輸入的值作為分隔：

```
const name = ["Tom", "Jerry", "Yeon", "Kevin"];

console.log(name.join(", ")); // "Tom, Jerry, Yeon, Kevin"
console.log(typeof name.join(", ")); // string
```

concat()

`concat()` 方法被用來合併兩個或多個陣列。回傳的是新陣列，不會改變原來的陣列。

按照參數的順序合併：

```
const arr = [1, 2, 3, 4];
const arr2 = [2, 4, 5, 6, 7];
console.log(arr.concat(arr2)); // [1, 2, 3, 4, 2, 4, 5, 6, 7]
```

可合併的陣列數不止兩個，你也可以合併同一個陣列無限次....：

```
const arr = [1, 2, 3, 4];
const arr2 = [2, 4, 5, 6, 7];
const arr3 = [3, 5, 6, 8];
console.log(arr.concat(arr2, arr3)); // [1, 2, 3, 4, 2, 4, 5, 6, 7, 3, 5, 6, 8]
console.log(arr.concat(arr2, arr2, arr2)); // [1, 2, 3, 4, 2, 4, 5, 6, 7, 2, 4, 5, 6, 7, 2, 4, 5, 6, 7]
```

但如果只是簡單的陣列元素合併，其實用**擴展運算子**也可以達到同樣效果：

```
const arr = [1, 2, 3, 4];
const arr2 = [2, 4, 5, 6, 7];
const arr3 = [3, 6, 8, 23, 1];
```

```
console.log(arr.concat(arr2, arr3)); // [1, 2, 3, 4, 2, 4, 5, 6, 7, 3, 6, 8, 23, 1]
console.log([...arr, ...arr2, ...arr3]); // [1, 2, 3, 4, 2, 4, 5, 6, 7, 3, 6, 8, 23, 1]
```

reduce()

`reduce()` 方法將一個累加器及陣列中每項元素（由左至右）傳入回呼函式，將陣列化為單一值。

```
const arr = [1, 2, 3, 4];
console.log(arr.reduce((prev, current, index, arr) => {
 console.log(prev, current, index, arr);
 return prev + current;
}));

/*
  1, 2, 1, [1, 2, 3, 4]
  3, 3, 2, [1, 2, 3, 4]
  6, 4, 3, [1, 2, 3, 4]
  10
*/
```

reduce 可以傳入四個參數：prev, current, index 和 array 本身，而方法累加器的方向是由前至後，如果希望由後至前累加可以使用 `reduceRight()`。

```
const arr = [1, 2, 3, 4];
console.log(arr.reduceRight((prev, current, index, arr) => {
 console.log(prev, current, index, arr);
  return prev + current;
}));

/*
  4, 3, 2, [1, 2, 3, 4]
  7, 2, 1, [1, 2, 3, 4]
  9, 1, 0, [1, 2, 3, 4]
  10
*/
```

includes()

`includes()` 方法會判斷陣列是否包含特定的元素，並以此來回傳 true 或 false。

```
const arr = [1, 2, 3, 4];
console.log(arr.includes(2)); // true
console.log(arr.includes(5)); // false
console.log(arr.includes(2, 2)); // false
console.log(arr.includes(2, -3)); // true
```

includes 的第一個參數為要搜尋的元素，第二個參數則為開始搜尋的索引值(含本身)。如為負數值，則自 array.length + fromIndex 開始向後搜尋。預設值為 0。

如果索引值比陣列本身長度還大，則會回傳 `false`。

indexOf()

`indexOf()` 方法會回傳給定元素於陣列中**第一個被找到之索引**，若不存在於陣列中則回傳 -1。

要注意的是回傳的是 “**第一個被找到的**” 索引，而不是全部索引。

```
const arr = [1, 2, 3, 4];
console.log(arr.indexOf(2)); // 1
console.log(arr.indexOf(2, 2)); // -1
console.log(arr.indexOf(2, -3)); // 1
```

使用的方式和 `includes()` 很像，只不過 `indexOf()` 回傳的是索引值。

如果想要找出該元素在陣列中的所有索引值可以搭配 `forEach()` 來寫：

```
const arr = [1, 2, 3, 2, 4, 7, 4, 2];
let result = [];
let index = arr.indexOf(2);

arr.forEach(() => {
  if (index != -1) {
    result.push(index);
    index = arr.indexOf(2, index + 1);
  }
})

console.log(result); // [1, 3, 7]
```

要特別注意的一點是，`indexOf()` 使用的是**嚴格相等比較**(===)，因此當型別不同時，比如 `"1"` 和 `1` 就不嚴格相等，就不會找到該值。

```
const arr = [1, 2, 3, "2", 4];
console.log(arr.indexOf("2")); // 3
console.log(arr.indexOf(2, 2)); // -1
```

注意

1. `indexOf()` 使用的是嚴格相等比較。
2. `indexOf()` 是由左至右，如果希望由右至左可以使用 `lastIndexOf()`。

sort()

`sort()` 方法會原地（in place）對一個陣列的所有元素進行排序，並回傳此陣列。排序**不一定是穩定的**（stable）。預設的排序順序是根據字串的 Unicode 編碼位置（code points）而定。

我比較常使用到的情形如下：

```
const arr = [
  {name: "Tom", score: 99},
  {name: "Jerry", score: 75},
  {name: "Sandy", score: 45},
  {name: "Monica", score: 81},
];

console.log(arr.sort((a, b) => a.score > b.score ? 1 : -1));

/*
[{
  name: "Sandy",
  score: 45
}, {
  name: "Jerry",
  score: 75
}, {
  name: "Monica",
  score: 81
}, {
  name: "Tom",
  score: 99
```

```
}]
*/
```

當 `sort()` 回傳 `true` (or `1`) 時會將 a, b 兩元素調換，a, b 由左至右開始遍歷。

如果要由大至小則是寫**小於時回傳 1**，當 a 比 b 小時才會調換順序，這樣就能達到**由大到小**排序的效果：

```
const arr = [
  {name: "Tom", score: 99},
  {name: "Jerry", score: 75},
  {name: "Sandy", score: 45},
  {name: "Monica", score: 81},
];

console.log(arr.sort((a, b) => a.score < b.score ? 1 : -1));

/*
[{
  name: "Tom",
  score: 99
}, {
  name: "Monica",
  score: 81
}, {
  name: "Jerry",
  score: 75
}, {
  name: "Sandy",
  score: 45
}]
*/
```

push()

`push()` 方法會添加一個或多個元素至陣列的末端，並且回傳陣列的新長度。

這應該也是特別常用的一個方法：

```
const arr = [
  {name: "Tom", score: 99},
  {name: "Jerry", score: 75},
  {name: "Sandy", score: 45},
  {name: "Monica", score: 81},
];

console.log(arr.push({
  name: "Kevin",
  score: 93
})); // 5

arr.push({
  name: "Peter",
  score: 66
})

console.log(arr);
```

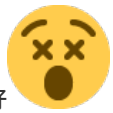
```
/*
[{
  name: "Tom",
  score: 99
}, {
  name: "Jerry",
  score: 75
}, {
  name: "Sandy",
  score: 45
}, {
  name: "Monica",
  score: 81
}, {
  name: "Kevin",
  score: 93
}, {
  name: "Peter",
  score: 66
}]
*/
```

也可以 push 多個元素：

```
const arr = [1, 2, 3];
console.log(arr.push(4, 5, 6)); // 6
arr.push(4, 5, 6);
console.log(arr); // [1, 2, 3, 4, 5, 6, 4, 5, 6]
```

並且要記住 `push()` 會改變原始陣列，因此當我前面 `console.log` 時 push 了 `4,5,6`，再次 push `4, 5, 6` 時陣列的長度會為 `9` 而不是 `6`。

工作時幾乎時時刻刻都在處理陣列，所以對陣列的處理方法必須非常熟悉，那這次的工作系列又結束了，開始頭禿想下次該更什麼好



這篇文章 工作時經常用到的 JS 知識點及技巧(3)－陣列的處理 最早出現於 134340號小行星。
