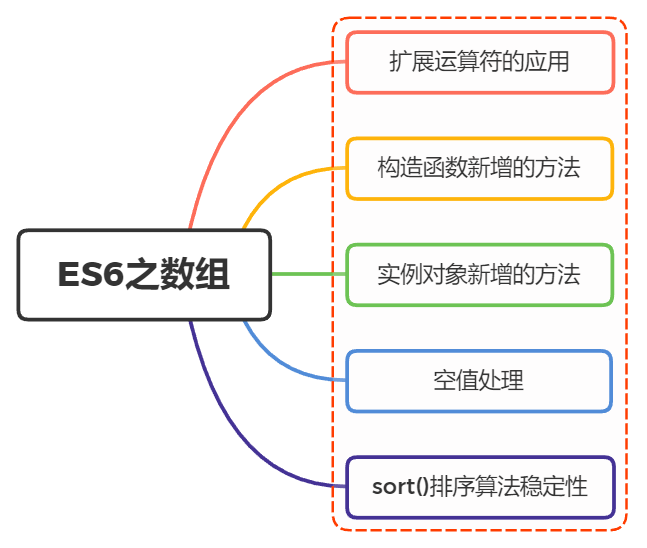
# 面试官：数组新增了哪些扩展？



## 一、扩展运算符的应用

ES6通过扩展元素符...，好比 rest 参数的逆运算，将一个数组转为用逗号分隔的参数序列

console.log(...[1, 2, 3])  
// 1 2 3  
  
console.log(1, ...[2, 3, 4], 5)  
// 1 2 3 4 5  
  
[...document.querySelectorAll('div')]  
// [<div>, <div>, <div>]

主要用于函数调用的时候，将一个数组变为参数序列

function push(array, ...items) {  
 array.push(...items);  
}  
  
function add(x, y) {  
 return x + y;  
}  
  
const numbers = [4, 38];  
add(...numbers) // 42

可以将某些数据结构转为数组

[...document.querySelectorAll('div')]

能够更简单实现数组复制

const a1 = [1, 2];  
const [...a2] = a1;  
// [1,2]

数组的合并也更为简洁了

const arr1 = ['a', 'b'];  
const arr2 = ['c'];  
const arr3 = ['d', 'e'];  
[...arr1, ...arr2, ...arr3]  
// [ 'a', 'b', 'c', 'd', 'e' ]

注意：通过扩展运算符实现的是浅拷贝，修改了引用指向的值，会同步反映到新数组

下面看个例子就清楚多了

const arr1 = ['a', 'b',[1,2]];  
const arr2 = ['c'];  
const arr3 = [...arr1,...arr2]  
arr[1][0] = 9999 // 修改arr1里面数组成员值  
console.log(arr[3]) // 影响到arr3,['a','b',[9999,2],'c']

扩展运算符可以与解构赋值结合起来，用于生成数组

const [first, ...rest] = [1, 2, 3, 4, 5];  
first // 1  
rest // [2, 3, 4, 5]  
  
const [first, ...rest] = [];  
first // undefined  
rest // []  
  
const [first, ...rest] = ["foo"];  
first // "foo"  
rest // []

如果将扩展运算符用于数组赋值，只能放在参数的最后一位，否则会报错

const [...butLast, last] = [1, 2, 3, 4, 5];  
// 报错  
  
const [first, ...middle, last] = [1, 2, 3, 4, 5];  
// 报错

可以将字符串转为真正的数组

[...'hello']  
// [ "h", "e", "l", "l", "o" ]

定义了遍历器（Iterator）接口的对象，都可以用扩展运算符转为真正的数组

let nodeList = document.querySelectorAll('div');  
let array = [...nodeList];  
  
let map = new Map([  
 [1, 'one'],  
 [2, 'two'],  
 [3, 'three'],  
]);  
  
let arr = [...map.keys()]; // [1, 2, 3]

如果对没有 Iterator 接口的对象，使用扩展运算符，将会报错

const obj = {a: 1, b: 2};  
let arr = [...obj]; // TypeError: Cannot spread non-iterable object

## 二、构造函数新增的方法

关于构造函数，数组新增的方法有如下：

* Array.from()
* Array.of()

### Array.from()

将两类对象转为真正的数组：类似数组的对象和可遍历（iterable）的对象（包括 ES6 新增的数据结构 Set 和 Map）

let arrayLike = {  
 '0': 'a',  
 '1': 'b',  
 '2': 'c',  
 length: 3  
};  
let arr2 = Array.from(arrayLike); // ['a', 'b', 'c']

还可以接受第二个参数，用来对每个元素进行处理，将处理后的值放入返回的数组

Array.from([1, 2, 3], (x) => x \* x)  
// [1, 4, 9]

### Array.of()

用于将一组值，转换为数组

Array.of(3, 11, 8) // [3,11,8]

没有参数的时候，返回一个空数组

当参数只有一个的时候，实际上是指定数组的长度

参数个数不少于 2 个时，Array()才会返回由参数组成的新数组

Array() // []  
Array(3) // [, , ,]  
Array(3, 11, 8) // [3, 11, 8]

### 三、实例对象新增的方法

关于数组实例对象新增的方法有如下：

* copyWithin()
* find()、findIndex()
* fill()
* entries()，keys()，values()
* includes()
* flat()，flatMap()

### copyWithin()

将指定位置的成员复制到其他位置（会覆盖原有成员），然后返回当前数组

参数如下：

* target（必需）：从该位置开始替换数据。如果为负值，表示倒数。
* start（可选）：从该位置开始读取数据，默认为 0。如果为负值，表示从末尾开始计算。
* end（可选）：到该位置前停止读取数据，默认等于数组长度。如果为负值，表示从末尾开始计算。

[1, 2, 3, 4, 5].copyWithin(0, 3) // 将从 3 号位直到数组结束的成员（4 和 5），复制到从 0 号位开始的位置，结果覆盖了原来的 1 和 2  
// [4, 5, 3, 4, 5]

### find()、findIndex()

find()用于找出第一个符合条件的数组成员

参数是一个回调函数，接受三个参数依次为当前的值、当前的位置和原数组

[1, 5, 10, 15].find(function(value, index, arr) {  
 return value > 9;  
}) // 10

findIndex返回第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1

[1, 5, 10, 15].findIndex(function(value, index, arr) {  
 return value > 9;  
}) // 2

这两个方法都可以接受第二个参数，用来绑定回调函数的this对象。

function f(v){  
 return v > this.age;  
}  
let person = {name: 'John', age: 20};  
[10, 12, 26, 15].find(f, person); // 26

### fill()

使用给定值，填充一个数组

['a', 'b', 'c'].fill(7)  
// [7, 7, 7]  
  
new Array(3).fill(7)  
// [7, 7, 7]

还可以接受第二个和第三个参数，用于指定填充的起始位置和结束位置

['a', 'b', 'c'].fill(7, 1, 2)  
// ['a', 7, 'c']

注意，如果填充的类型为对象，则是浅拷贝

### entries()，keys()，values()

keys()是对键名的遍历、values()是对键值的遍历，entries()是对键值对的遍历

or (let index of ['a', 'b'].keys()) {  
 console.log(index);  
}  
// 0  
// 1  
  
for (let elem of ['a', 'b'].values()) {  
 console.log(elem);  
}  
// 'a'  
// 'b'  
  
for (let [index, elem] of ['a', 'b'].entries()) {  
 console.log(index, elem);  
}  
// 0 "a"

### includes()

用于判断数组是否包含给定的值

[1, 2, 3].includes(2) // true  
[1, 2, 3].includes(4) // false  
[1, 2, NaN].includes(NaN) // true

方法的第二个参数表示搜索的起始位置，默认为0

参数为负数则表示倒数的位置

[1, 2, 3].includes(3, 3); // false  
[1, 2, 3].includes(3, -1); // true

### flat()，flatMap()

将数组扁平化处理，返回一个新数组，对原数据没有影响

[1, 2, [3, 4]].flat()  
// [1, 2, 3, 4]

flat()默认只会“拉平”一层，如果想要“拉平”多层的嵌套数组，可以将flat()方法的参数写成一个整数，表示想要拉平的层数，默认为1

[1, 2, [3, [4, 5]]].flat()  
// [1, 2, 3, [4, 5]]  
  
[1, 2, [3, [4, 5]]].flat(2)  
// [1, 2, 3, 4, 5]

flatMap()方法对原数组的每个成员执行一个函数相当于执行Array.prototype.map()，然后对返回值组成的数组执行flat()方法。该方法返回一个新数组，不改变原数组

// 相当于 [[2, 4], [3, 6], [4, 8]].flat()  
[2, 3, 4].flatMap((x) => [x, x \* 2])  
// [2, 4, 3, 6, 4, 8]

flatMap()方法还可以有第二个参数，用来绑定遍历函数里面的this

### 四、数组的空位

数组的空位指，数组的某一个位置没有任何值

ES6 则是明确将空位转为undefined，包括Array.from、扩展运算符、copyWithin()、fill()、entries()、keys()、values()、find()和findIndex()

建议大家在日常书写中，避免出现空位

### 五、排序稳定性

将sort()默认设置为稳定的排序算法

const arr = [  
 'peach',  
 'straw',  
 'apple',  
 'spork'  
];  
  
const stableSorting = (s1, s2) => {  
 if (s1[0] < s2[0]) return -1;  
 return 1;  
};  
  
arr.sort(stableSorting)  
// ["apple", "peach", "straw", "spork"]

排序结果中，straw在spork的前面，跟原始顺序一致

## 参考文献

* https://es6.ruanyifeng.com/#docs/array