【前端面试】JS数组方法中哪些会改变原数...

前言

作为一名前端开发人员，我们每天都会与数组打交道。JS 也提供了很多操作数组的原生 API 供我们调用。在这些方法里面，有的方法会改变原数组，有些不会改变原数组。别看这一点小小的区别，往往会造成巨大的影响，特别是在算法层面，有可能会造成算法复杂度的飙升。

今天我们就来好好理一理哪些数组方法操作会改变原数组，哪些数组方法不会改变原数组。

1.会改变原数组的方法

1.1 push

push 可以说是最常用的数组操作方法了，它也非常好理解，本身就是往数组里面添加元素的意思。

作用：

push 方法往数组里面添加元素，返回数组的长度。

示例代码：

```
let arr1 = ['小猪课堂', 23];
let length1 = arr1.push('张三');
console.log(arr1); // ['小猪课堂', 23, '张三']
console.log(length1); // 3
```

上段代码中 arr1 数组是变化了的，push 方法返回的是数组最新的长度。

1.2 unshift

作用：

unshift 方法会在数组的开头添加一个元素，它会返回数组新的长度。

示例代码：

```
let arr2 = ['小猪课堂', 23];
let length2 = arr2.unshift('张三');
console.log(arr2); // ['张三', '小猪课堂', 23]
console.log(length2); // 3
```

1.3 pop

作用：

pop() 方法从数组中删除最后一个元素，并返回该元素的值。

示例代码：

```
let arr3 = ['小猪课堂', 23];
let value1 = arr3.pop();
console.log(arr3); // ['小猪课堂']
console.log(value1); // 23
```

1.4 shift

作用：

shift() 方法从数组中删除第一个元素，并返回该元素的值。

示例代码：

```
let arr4 = ['小猪课堂', 23];
let value2 = arr4.shift();
console.log(arr4); // [23]
console.log(value2); // 小猪课堂
```

1.5 sort

作用：

sort() 方法用原地算法对数组的元素进行排序，并返回数组。默认排序顺序是在将元素转换为字符串，然后比较它们的 UTF-16 代码单元值序列时构建的。它返回的就是排序后的数组。

示例代码：

```
let arr5 = [321, 23, 12, 6666, 2];
let value3 = arr5.sort();
console.log(arr5); // [12, 2, 23, 321, 6666]
console.log(value3); // [12, 2, 23, 321, 6666]
```

注意：

因为 sort 的默认排序是将元素转化为字符串后排序的，所以上述代码中的排序结果可能不是我们想要的，想要获得正确的排序结果，我们可以传入一个函数，来规定排序的规则。

代码如下：

```
arr5.sort((a, b) => a - b);
console.log(arr5); // [2, 12, 23, 321, 6666]
```

1.6 splice

作用：

splice() 方法用于添加或删除数组中的元素，如果删除一个元素，则返回一个元素的数组。 如果未删除任何元素，则返回空数组。

示例代码：

```
let arr6 = ['张三', '小猪课堂', 23];
let item1 = arr6.splice(0,1); // 删除一个元素，返回删除元素的数组
console.log(item1); // ['张三']
console.log(arr6); // ['小猪课堂', 23]
```

1.7 reverse

作用：

reverse() 方法用于颠倒数组中元素的顺序。

示例代码：

```
let arr7 = ['张三', '小猪课堂', 23];
let item2 = arr7.reverse();
console.log(item2); // [23, '小猪课堂', '张三']
console.log(arr7); // [23, '小猪课堂', '张三']
```

2.不会改变原数组的方法

2.1 concat

作用：

concat() 方法用于连接两个或多个数组。该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。

示例代码：

```
let arr8 = ['小猪课堂'];
let arr9 = ['张三'];
let arr10 = arr8.concat(arr9);
console.log(arr8); // ['小猪课堂']
console.log(arr9); // ['张三']
console.log(arr10); // ['小猪课堂', '张三']
```

2.2 join

作用：

join() 方法用于把数组中的所有元素转换一个字符串，元素是通过指定的分隔符进行分隔的。

示例代码：

```
let arr11 = [1, 2, 3, 4, 5, 6];
let item3 = arr1.join(',');
console.log(item3); // 小猪课堂,23,张三
```

```
console.log(arr11); // [1, 2, 3, 4, 5, 6]
```

2.3 reduce

作用：

reduce() 方法接收一个函数作为累加器，数组中的每个值（从左到右）开始缩减，最终计算为一个值。reduce 方法的使用情况稍微复杂，不熟悉的小伙伴建议去官网好好学学，它的应用范围还是挺宽泛的。

示例代码：

```
let arr12 = [1, 2, 3, 4, 5, 6];
let item4 = arr12.reduce((total,currentValue)=>{
  return total + currentValue;
}, 0);
console.log(item4); // 21
console.log(arr12); // [1, 2, 3, 4, 5, 6]
```

2.4 map

作用：

map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

示例代码：

```
let arr13 = [1, 2, 3, 4, 5, 6];
let item5 = arr13.map((item) => {
  return item + 100;
});
console.log(item5); // [101, 102, 103, 104, 105, 106]
console.log(arr13); // [1, 2, 3, 4, 5, 6]
```

2.5 forEach

作用：

forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

示例代码：

```
let arr14 = [1, 2, 3, 4, 5, 6];
let item6 = arr14.forEach((item) => {
  item + 100;
});
console.log(item6); // undefined
console.log(arr14); // [1, 2, 3, 4, 5, 6]
```

注意：

forEach 方法没有返回值，而且它也不会改变原数组，有些同学误以为 forEach 会改变原数组，通常是因为在 forEach 方法的回调函数中，我们自己做了更改原数组的操作。

2.6 filter

作用：

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

示例代码：

```
let arr15 = [1, 2, 3, 4, 5, 6];
let item7 = arr15.filter((item) => {
  return item > 3;
});
console.log(item7); // [4, 5, 6]
console.log(arr15); // [1, 2, 3, 4, 5, 6]
```

2.7 slice

作用：

slice() 方法可从已有的数组中返回选定的元素。slice() 方法可提取字符串的某个部分，并以新的字符串返回被提取的部分。

示例代码：

```
let arr16 = [1, 2, 3, 4, 5, 6];
let item8 = arr15.slice(0, 3);
console.log(item8); // [1, 2, 3]
console.log(arr16); // [1, 2, 3, 4, 5, 6]
```

2.8 findIndex

作用：

findIndex 接收一个测试函数，也可以叫做条件函数，最终返回满足该测试函数的元素的索引位置，如果没有找到符合条件的元素，则返回-1。如果满足条件的有多个，那么只会返回第一个满足条件的元素索引。

示例代码：

```
let arr17 = [1, 2, 3, 4, 5, 6];
let item9 = arr17.findIndex((item) => {
  return item > 2;
});
console.log(item9); // 2
console.log(arr17); // [1, 2, 3, 4, 5, 6]
```

总结

数组方法基本上每一版本的 JS 中都会增加一些，本篇文章主要总结了我们开发中比较常用而且容易混淆的。至于为什么要区分是否更改原数组，那就得根据实际情况来决定了。打个比方，如果有一道算法题，可以用数组方法中的 unshift 和 slice 来解决，那么选用哪一个呢？首先 unshift 会改变原数组，它的事件复杂度是 O(n)，而 slice 方法不会改变原数组，它的时间复杂度是 O(1)，改用哪个一目了然。