2017年06月21日 | Unknown author

JavaScript专题之数组去重

> JavaScript 专题系列第三篇，讲解各种数组去重方法，并且跟着 underscore 写一个 unique API

前言

数组去重方法老生常谈，既然是常谈，我也来谈谈。

双层循环

也许我们首先想到的是使用 indexOf 来循环判断一遍，但在这个方法之前，让我们先看看最原始的方法：

```
var array = [1, 1, '1', '1'];

function unique(array) {

    var res = [];
    for (var i = 0, arrayLen = array.length; i < arrayLen;
        for (var j = 0, resLen = res.length; j < resLen; j
```

```
            if (array[i] === res[j]) {
                break;
            }
        }

        if (j === resLen) {
            res.push(array[i])
        }
    }
    return res;
}

console.log(unique(array)); 复制代码
```

在这个方法中，我们使用循环嵌套，最外层循环 array，里面循环 res，如果 array[i] 的值跟 res[j] 的值相等，就跳出循环，如果都不等于，说明元素是唯一的，这时候 j 的值就会等于 res 的长度，根据这个特点进行判断，将值添加进 res。

看起来很简单吧，之所以要讲一讲这个方法，是因为——————兼容性好！

indexOf

我们可以用 indexOf 简化内层的循环：

```
var array = [1, 1, '1'];

function unique(array) {
```

```
    var res = [];
    for (var i = 0, len = array.length; i < len; i++) {
        var current = array[i];
        if (res.indexOf(current) === -1) {
            res.push(current)
        }
    }
    return res;
}

console.log(unique(array));复制代码
```

排序后去重

试想我们先将要去重的数组使用 sort 方法排序后，相同的值就会被排在一起，然后我们就可以只判断当前元素与上一个元素是否相同，相同就说明重复，不相同就添加进 res，让我们写个 demo：

```
var array = [1, 1, '1'];

function unique(array) {
    var res = [];
    var sortedArray = array.concat().sort();
    var seen;
    for (var i = 0, len = sortedArray.length; i < len; i++

        if (!i || seen !== sortedArray[i]) {
```

```
                res.push(sortedArray[i])
            }
            seen = sortedArray[i];
        }
        return res;
    }

    console.log(unique(array));复制代码
```

如果我们对一个已经排好序的数组去重，这种方法效率肯定高于使用 indexOf。

unique API

知道了这两种方法后，我们可以去尝试写一个名为 unique 的工具函数，我们根据一个参数 isSorted 判断传入的数组是否是已排序的，如果为 true，我们就判断相邻元素是否相同，如果为 false，我们就使用 indexOf 进行判断

```
var array1 = [1, 2, '1', 2, 1];
var array2 = [1, 1, '1', 2, 2];


function unique(array, isSorted) {
    var res = [];
    var seen = [];

    for (var i = 0, len = array.length; i < len; i++) {
        var value = array[i];
```

```
            if (isSorted) {
                if (!i || seen !== value) {
                    res.push(value)
                }
                seen = value;
            }
            else if (res.indexOf(value) === -1) {
                res.push(value);
            }
        }
        return res;
    }

    console.log(unique(array1));
    console.log(unique(array2, true)); 复制代码
```

优化

尽管 unqique 已经可以试下去重功能，但是为了让这个 API 更加强大，我们来考虑一个需求：

新需求：字母的大小写视为一致，比如'a'和'A'，保留一个就可以了！

虽然我们可以先处理数组中的所有数据，比如将所有的字母转成小写，然后再传入unique函数，但是有没有方法可以省掉处理数组的这一遍循环，直接就在去重的循环中做呢？让我们去完成这个需求：

```
var array3 = [1, 1, 'a', 'A', 2, 2];



function unique(array, isSorted, iteratee) {
    var res = [];
    var seen = [];

    for (var i = 0, len = array.length; i < len; i++) {
        var value = array[i];
        var computed = iteratee ? iteratee(value, i, array
        if (isSorted) {
            if (!i || seen !== value) {
                res.push(value)
            }
            seen = value;
        }
        else if (iteratee) {
            if (seen.indexOf(computed) === -1) {
                seen.push(computed);
                res.push(value);
            }
        }
        else if (res.indexOf(value) === -1) {
            res.push(value);
        }
    }
```

```
    return res;
}

console.log(unique(array3, false, function(item){
    return typeof item == 'string' ? item.toLowerCase() :
}));
```

复制代码

在这一版也是最后一版的实现中，函数传递三个参数：

array：表示要去重的数组，必填

isSorted：表示函数传入的数组是否已排过序，如果为 true，将会采用更快的方法进行去重

iteratee：传入一个函数，可以对每个元素进行重新的计算，然后根据处理的结果进行去重

至此，我们已经仿照着 underscore 的思路写了一个 unique 函数，具体可以查看 Github。

filter

ES5 提供了 filter 方法，我们可以用来简化外层循环：

比如使用 indexOf 的方法：

```
var array = [1, 2, 1, 1, '1'];

function unique(array) {
```

```
    var res = array.filter(function(item, index, array){
        return array.indexOf(item) === index;
    })
    return res;
}

console.log(unique(array));复制代码
```

排序去重的方法：

```
var array = [1, 2, 1, 1, '1'];

function unique(array) {
    return array.concat().sort().filter(function(item, ind
        return !index || item !== array[index - 1]
    })
}

console.log(unique(array));复制代码
```

Object 键值对

去重的方法众多，尽管我们已经跟着 underscore 写了一个 unqiue API，但是让我们看看其他的方法拓展下视野：

这种方法是利用一个空的 Object 对象，我们把数组的值存成 Object 的 key 值，比如 Object[value1] = true，在判断另一个值的时候，如果

Object[value2]存在的话，就说明该值是重复的。示例代码如下：

```
var array = [1, 2, 1, 1, '1'];

function unique(array) {
    var obj = {};
    return array.filter(function(item, index, array){
        return obj.hasOwnProperty(item) ? false : (obj[ite
    })
}

console.log(unique(array)); 复制代码
```

我们可以发现，是有问题的，因为 1 和 '1' 是不同的，但是这种方法会判断为同一个值，这是因为对象的键值只能是字符串，所以我们可以使用 typeof item + item 拼成字符串作为 key 值来避免这个问题：

```
var array = [1, 2, 1, 1, '1'];

function unique(array) {
    var obj = {};
    return array.filter(function(item, index, array){
        return obj.hasOwnProperty(typeof item + item) ? fa
    })
}

console.log(unique(array)); 复制代码
```

ES6

随着 ES6 的到来，去重的方法又有了进展，比如我们可以使用 Set 和 Map 数据结构，以 Set 为例，ES6 提供了新的数据结构 Set。它类似于数组，但是成员的值都是唯一的，没有重复的值。

是不是感觉就像是为去重而准备的？让我们来写一版：

```
var array = [1, 2, 1, 1, '1'];

function unique(array) {
   return Array.from(new Set(array));
}

console.log(unique(array)); 复制代码
```

甚至可以再简化下：

```
function unique(array) {
    return [...new Set(array)];
}复制代码
```

还可以再简化下：

```
var unique = (a) => [...new Set(a)]复制代码
```

此外，如果用 Map 的话：

```
function unique (arr) {
    const seen = new Map()
    return arr.filter((a) => !seen.has(a) && seen.set(a, 1
}复制代码
```

JavaScript 的进化

我们可以看到，去重方法从原始的 14 行代码到 ES6 的 1 行代码，其实也说明了 JavaScript 这门语言在不停的进步，相信以后的开发也会越来越高效。

特殊类型比较

去重的方法就到此结束了，然而要去重的元素类型可能是多种多样，除了例子中简单的 1 和 '1' 之外，其实还有 null、undefined、NaN、对象等，那么对于这些元素，之前的这些方法的去重结果又是怎样呢?

在此之前，先让我们先看几个例子:

```
var str1 = '1';
var str2 = new String('1');

console.log(str1 == str2);
console.log(str1 === str2);

console.log(null == null);
console.log(null === null);
```

```
console.log(undefined == undefined);
console.log(undefined === undefined);

console.log(NaN == NaN);
console.log(NaN === NaN);

console.log(/a/ == /a/);
console.log(/a/ === /a/);

console.log({} == {});
console.log({} === {}); 复制代码
```

那么，对于这样一个数组

```
var array = [1, 1, '1', '1', null, null, undefined, undefi
```

以上各种方法去重的结果到底是什么样的呢?

我特地整理了一个列表，我们重点关注下对象和 NaN 的去重情况：

方法	结果	说明
for循环	[1, "1", null, undefined, String, String, /a/, /a/, NaN, NaN]	对象和 NaN 不去重
indexOf	[1, "1", null, undefined, String, String, /a/, /a/, NaN, NaN]	对象和 NaN 不去重
sort	[/a/, /a/, "1", 1, String, 1, String, NaN, NaN, null, undefined]	对象和 NaN 不去重 数字 1 也不去重
filter +	[1, "1", null, undefined, String,	对象不去重 NaN 会被

indexOf	String, /a/, /a/]	忽略掉
filter + sort	[/a/, /a/, "1", 1, String, 1, String, NaN, NaN, null, undefined]	对象和 NaN 不去重 数字 1 不去重
优化后的键值对方法	[1, "1", null, undefined, String, /a/, NaN]	全部去重
Set	[1, "1", null, undefined, String, String, /a/, /a/, NaN]	对象不去重 NaN 去重

想了解为什么会出现以上的结果，看两个 demo 便能明白：

```
var arr = [1, 2, NaN];
arr.indexOf(NaN); 复制代码
```

indexOf 底层还是使用 === 进行判断，因为 NaN ==== NaN的结果为 false，所以使用 indexOf 查找不到 NaN 元素

```
function unique(array) {
   return Array.from(new Set(array));
}
console.log(unique([NaN, NaN])) 复制代码
```

Set 认为尽管 NaN === NaN 为 false，但是这两个元素是重复的。

写在最后

虽然去重的结果有所不同，但更重要的是让我们知道在合适的场景要选择合适的方法。

专题系列

JavaScript专题系列目录地址：github.com/mqyqingfeng...。

JavaScript专题系列预计写二十篇左右，主要研究日常开发中一些功能点的实现，比如防抖、节流、去重、类型判断、拷贝、最值、扁平、柯里、递归、乱序、排序等，特点是研(chao)究(xi) underscore 和 jQuery 的实现方式。

如果有错误或者不严谨的地方，请务必给予指正，十分感谢。如果喜欢或者有所启发，欢迎 star，对作者也是一种鼓励。
