



# JS数组及手动封装ES3和ES5的数组核心方法（一）



jCodeLife

2020.08.09 21:48:39 字数 2,525 阅读 3

编辑文章

跟我们平时理解的数组一样，js数组也是一种数据存储结构，用于用单个变量名存储多个值。js数组的本质是基于对象，都是通过Array构造出来的。

首先，我们来看看怎么创建数组

## 数组的两种创建方式

### 1. 数组字面量

```
1 | var arr = [1,2,3];
```

### 2. 通过new Array(length/content)创建数组

```
1 | var arr = new Array(1,2,3)
```

不过有问题，如下：

```
1 | var arr = new Array();
2 | console.log(arr);//[ ]
3 |
4 | var arr = new Array(5);
5 | console.log(arr); // [empty x 5]
6 |
7 | var arr = new Array(-5);
8 | console.log(arr); // RangeError
9 |
10 | var arr = new Array("aa");
11 | console.log(arr); // ["aa"]
12 |
13 | var arr = new Array(true);
14 | console.log(arr); // [true]
15 |
16 | var arr = new Array({});
17 | console.log(arr); // [{}]
18 |
19 | var arr = new Array(1,2,3,4);
20 | console.log(arr); // [1,2,3,4]
```

我们可以看出，通过new Array来创建数组时，不同的参数导致不同结果，当传入参数为：

- a. 无参数时，返回一个空数组
- b. 单个正整数参数，会当成的新数组的长度，创建对应长度的稀疏数组，每个位置的值都是undefined
- c. 单个非正整数的数值作为参数时，会报错
- d. 单个非数值参数（比如字符串、布尔值、对象等），则该参数是返回的新数组的成员
- e. 多个参数时，才返回正常的数组，多个参数都是数组的成员

所以正是因为Array作为构造函数，行为很不一致。因此，在生成新数组，不建议使用new Array()的方式来创建数组，而是直接使用数组字面量。并且效果是一样的，相当于语法糖

```
1 | var arr1 = [1, 2, 3];
```



jCodeLife

总资产4 (约0.43元)

[深入call,apply,bind到手动封装](#)

阅读 42

[JS数组及手动封装20多个ES3和ES5的数组核心方法](#)

阅读 3

[深入理解JS对象的深度克隆及多种方式实现](#)

阅读 23

## 推荐阅读

[面试官：小伙子，你连Java集合都讲不清楚，怎么就敢开口要8K呀？](#)

阅读 19,600

[三生三世枕上书 续 27. 大开杀戒](#)

阅读 4,791

[做好以下三点的男人，一定很爱你](#)

阅读 2,552

[一味的只读书，好吗？](#)

阅读 4,579

[正确的阅读标记，让你的阅读收获事半功倍](#)

阅读 1,851



```
8 console.log(arr2.constructor);//Array() { [native code] }
9 console.log(arr1.constructor === arr2.constructor);//true
```

接着，来看数组的读、写和删除操作

## 数组的读和写以及删除

### 1. 读取数组元素;

格式: `arr[index]`

```
1 var arr = [1,2,3];
2 arr[0];//1
3 arr[10];//可以溢出读，结果是undefined而已
```

可以溢出读，结果是`undefined`而已

### 2. 写入取数组元素

格式: `arr[index] = xxx`

```
1 var arr = [1,2,3];
2 arr[10] = 10;
3 console.log(arr.length);//11
4 console.log(arr);//[1, 2, 3, empty × 7, 10]
5 console.log(arr[5]);//undefined
```

可以溢出写，中间空的就为空值。读取空置的位置，结果为`undefined`，`length`变成对应长度

可以看出，`js`数组的读和写极其松散，可以溢出读，也可以溢出写，基本不会报错。溢出读时，值为`undefined`；溢出写时，中间空的就为空值，读取空置的位置，结果为`undefined`，`length`会对应变化

### 3. `delete arr[num]`;

通过`delete`关键字可以删除数组中对应下标的元素

```
1 var arr = [1, 2, 3];
2 arr[10] = 10;
3 console.log(arr.length);//11
4 console.log(arr);//[1, 2, 3, empty × 7, 10]
5 console.log(arr[5]);//undefined
6
7 delete arr[10];
8 console.log(arr.length);//11
9 console.log(arr);//[1, 2, 3, empty × 8]
10 console.log(arr[5]);//undefined
```

这里可以看到通过溢出写的方式改变数组，接着通过`delete`删除对应写入的值时，数组的`length`不会发生改变了，被删除的值为空了。

再看看正常删除一个数组中的元素

```
1 var arr = [1,2,3]
2 delete arr[0];
3 console.log(arr);// [empty, 2, 3]
4 console.log(arr.length);// 3
```

原来，通过`delete`的方式删除数组成员，并不会改变原数组的`length`值。相当于将对应位置的内容清空，位置还是被占据

```
3 console.log(index); // 下标 0 1 2
4 console.log(arr[index]); // 值 1 2 3
5 }
```

index遍历表示数组下标

## ES3中数组常用方法（核心方法）

因为数组方法较多，我们分类记忆。大致可分两类：

### 1. 可改变原数组的方法

push、pop、shift、unshift、reverse、splice、sort

### 2. 不改变原数组的方法

concat、join、split、toString、slice

## 一、可改变原数组的方法

### 1. push()

用于在数组最后添加一个或多个元素，并返回添加新元素后的数组长度

#### • 基本使用

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

#### • 手动封装

```
1 Array.prototype._push = function(){
2   for(var i=0;i<arguments.length;i++){
3     this[this.length] = arguments[i];
4   }
5   return this.length;
6 }
7 //test code
8 var arr = [1,2,3];
9 console.log(arr._push(4,5)); // 5
10 console.log(arr) // [1,2,3,4,5]
```

#### • 使用场景：

##### (1) 合并数组

```
1 var a = [1,2];
2 var b = [3,4];
3 var res = Array.prototype.push.apply(a,b);
4 console.log(a) // [1,2,3,4]
5 console.log(b) // [3,4]
6 console.log(res) // 4; 就是a.length
```

或者

```
1 var a = ['a','b'];
2 var b = ['c','d'];
3 var res = a.push.apply(a, b)
4 console.log(a) // [1,2,3,4]
5 console.log(b) // [3,4]
6 console.log(res) // 4; 就是a.length
```

```
1 var obj = {name:'alice'}
2 var length = [].push.call(obj,1,2,3,4);//length:4
3 console.log(obj);
```

```
▼ {0: 1, 1: 2, 2: 3, 3: 4, name: "alice", length: 4} ⓘ
  0: 1
  1: 2
  2: 3
  3: 4
  length: 4
  name: "alice"
  ► __proto__: Object
```

## 2. pop()

用于删除数组最后一位元素，并返回该元素；相当于剪切

### • 基本使用

```
1 var arr = [1,2,3,4]
2 console.log(arr.pop()); //4
3 console.log(arr); // [1,2,3]
```

### • 手动封装

```
1 Array.prototype._pop = function(){
2   var result = this[this.length-1]
3   delete this[this.length];
4   this.length--;
5   return result;
6 }
7 //test code
8 var arr = [1,2,3,4]
9 console.log(arr._pop()); //4
10 console.log(arr); // [1,2,3]
11 console.log(arr.length); //3
```

注意：数组最后一位的下标是length-1

还有个小程序，原pop方法，如果是空数组[]时，不会报错，而是返回undefined。

完善代码

```
1 Array.prototype._pop = function () {
2   if (this.length) {
3     var result = this[this.length - 1]
4     delete this[this.length];
5     this.length--;
6     return result;
7   }
8   return;
9 }
10 //test code
11 //对空数组使用pop方法，不会报错，而是返回undefined。
12 console.log([]._pop());
13 console.log([]._pop());
```

利用push和pop方法，可以构成了“后进先出”的栈结构（stack）

## 3. unshift()

用于在数组的开头添加一个或多个元素，并返回添加新元素后的数组长度

### • 基本使用

注意顺序

- 手动封装

```
1 Array.prototype._unshift = function () {
2   //将数组本来元素移到后面去
3   for (var i = this.length - 1; i >= 0; i--) {
4     this[i + arguments.length] = this[i];
5   }
6   //将要添加的元素，一个一个添加进腾出来的空位上
7   for (var j = 0; j < arguments.length; j++) {
8     this[j] = arguments[j];
9   }
10  return this.length;
11 }
12
13 var arr = [1, 2, 3]
14 console.log(arr._unshift('a', 'b', 'c'));//6
15 console.log(arr);//[ 'a', 'b', 'c', 1, 2, 3]
16 console.log(arr.length);//6
```

#### 4. shift()

用于删除数组的第一个元素，并返回该元素。跟pop一样相当于剪切

- 基本使用:

```
1 var arr = [1, 2, 3]
2 console.log(arr.shift());//1
3 console.log(arr);//[2,3]
4 console.log(arr.length);//2
```

- 手动封装

```
1 Array.prototype._shift=function(){
2   var res = this[0];
3   //让前一位等于后一位，将第一位覆盖掉。
4   for(var index in this){
5     this[index-1] = this[index]
6   }
7   return res;
8 }
9 //test code
10 var arr = [1, 2, 3]
11 console.log(arr._shift());//1
12 console.log(arr);//[2,3]
13 console.log(arr.length);//2
```

核心在于数组遍历时前一位等于后一位，将第一位覆盖掉，最后length也会自动-1.

#### 5. reverse()

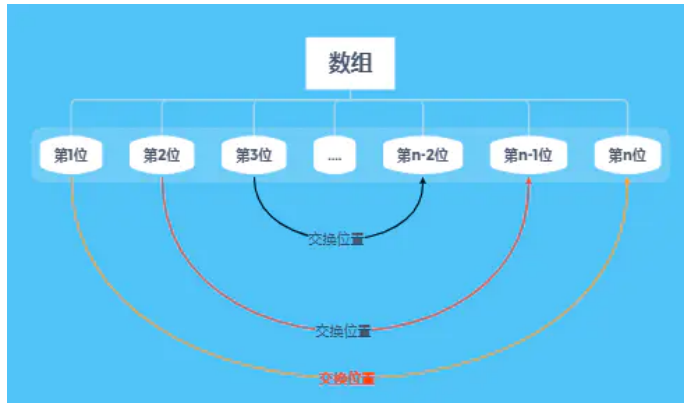
用于颠倒数组中元素的顺序，返回改变后的数组

- 基本使用

```
1 var arr = [1, 2, 3]
2 console.log(arr.reverse());//[3,2,1]
3 console.log(arr);//[3,2,1]
```

- 手动封装

我画了一个简图，如下：



代码实现：

```
1 Array.prototype._reverse = function () {
2   var left = 0; // 存储左边第一个位置
3   var right = this.length - 1; // 存储右边最后一个位置
4   while (left < right) { // 停止进行的条件
5     var temp = this[left]; // 利用一个中间变量来交换位置
6     this[left] = this[right];
7     this[right] = temp;
8     left++;
9     right--;
10  }
11 }
12 }
13 var arr = [1, 2, 3, 4, 5, 6, 7]
14 console.log(arr._reverse()); // undefined
15 console.log(arr); // [7, 6, 5, 4, 3, 2, 1]
```

## 6. splice()

截取并添加，返回被截取部分；添加在切口处添加，添加的新数据在原数组上

`arr.splice(从第几位开始, 截取的长度, 在切口处添加的数据1, 在切口处添加的数据2, ...)`

### • 基本使用

```
1 var arr = [1,2,3,4,5,6]
2 console.log(arr.splice(3,1,'a','b'));//[4]
3 console.log(arr);// [1, 2, 3, "a", "b", 5, 6]
4
5 var arr = [1,2,3,4,5,6]
6 console.log(arr.splice(3))//[4,5,6]
7 console.log(arr);// [1, 2, 3]
8
9 var arr = [1,2,3,4,5,6]
10 console.log(arr.splice())//[ ]
11 console.log(arr);// [1,2,3,4,5,6]
```

如果只提供第一个参数，那就是从该位置开始截取到最后返回；等同于将原数组拆分成两数组

如果没有传参，等同于没有截取、没有添加，原数组不变，返回的空数组[ ]

以上是最常见的情况，但还封装一样的功能，得再来深入具体每种情况，那我们接下来试试当传入其他数据类型的时候，会发生什么？

传入一个值时，各种各样的值类型情况：

```
1 //一个参数值
2 var arr = [1,2,3,4,5,6]
```



```
9
10 var arr = [1,2,3,4,5,6]
11 console.log(arr.splice("3aaa"))//[1,2,3,4,5,6]
12 console.log(arr);// []
13
14 var arr = [1,2,3,4,5,6]
15 console.log(arr.splice("aaa"))//[1,2,3,4,5,6]
16 console.log(arr);// []
17
18 var arr = [1,2,3,4,5,6]
19 console.log(arr.splice(null))//[1,2,3,4,5,6]
20 console.log(arr);// []
21
22 var arr = [1,2,3,4,5,6]
23 console.log(arr.splice(undefined))//[1,2,3,4,5,6]
24 console.log(arr);// []
25
26 var arr = [1,2,3,4,5,6]
27 console.log(arr.splice(false))//[1,2,3,4,5,6]
28 console.log(arr);// []
29
30 var arr = [1,2,3,4,5,6]
31 console.log(arr.splice(true))//[2,3,4,5,6]
32 console.log(arr);// [1]
33
34 var arr = [1,2,3,4,5,6]
35 console.log(arr.splice(0))//[1,2,3,4,5,6]
36 console.log(arr);// []
37
38 var arr = [1,2,3,4,5,6]
39 console.log(arr.splice([]))//[1,2,3,4,5,6]
40 console.log(arr);// []
41
42 var arr = [1,2,3,4,5,6]
43 console.log(arr.splice([1]))//[2,3,4,5,6]
44 console.log(arr);// [1]
45
46
47 var arr = [1,2,3,4,5,6]
48 console.log(arr.splice([3]))//[4,5,6]
49 console.log(arr);// [1,2,3]
50
51
52 var arr = [1,2,3,4,5,6]
53 console.log(arr.splice({}))//[1,2,3,4,5,6]
54 console.log(arr);// []
55
56 var arr = [1,2,3,4,5,6]
57 console.log(arr.splice(-3))//[4,5,6]
58 console.log(arr);// [1,2,3]
59
60 var arr = [1,2,3,4,5,6]
61 console.log(arr.splice(100))//[ ]
62 console.log(arr);// [1,2,3,4,5,6]
```

我们发现，当传入一个正常的`number`数字时，那就是从该位置开始截取到最后返回；等同于将原数组拆分成两数组

当传入的是字符型数字时，会将其转成正常数字，然后从该位置开始截取到最后

当传入的是非数字字符串、`null`、`undefined`、`false`、非一个数字的数组、对象时，都相当于`arr.splice(0)`，从第0位开始截取到最后

当传入的是单位数字的数组时，会将其变成数字，表示从该位置开始截取到最后

当传入的是`true`时，会将其变成1，表示从第一位开始截取到最后

可以传入负数，表示倒数位开始

可以传入大于`arr.length`的数字，返回截取部分是`[]`，表示没有截到

再看当传入两个值时，第二个值各种类型是

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

写下你的评论...

评论0

赞1

...



```
10 console.log(arr);// [1,2,3,4,5,6]
11
12 var arr = [1,2,3,4,5,6]
13 console.log(arr.splice(3,'aaa'))//[]
14 console.log(arr);// [1,2,3,4,5,6]
15
16 var arr = [1,2,3,4,5,6]
17 console.log(arr.splice(3,-1))//[]
18 console.log(arr);// [1,2,3,4,5,6]
19
20 var arr = [1,2,3,4,5,6]
21 console.log(arr.splice(3,100))//[4,5,6]
22 console.log(arr);// [1,2,3]
23
24 var arr = [1,2,3,4,5,6]
25 console.log(arr.splice(3,false))//[]
26 console.log(arr);// [1,2,3,4,5,6]
27
28 var arr = [1,2,3,4,5,6]
29 console.log(arr.splice(3,true))//[4]
30 console.log(arr);// [1,2,3,5,6]
31
32 var arr = [1,2,3,4,5,6]
33 console.log(arr.splice(3,[]))//[]
34 console.log(arr);// [1,2,3,4,5,6]
35
36 var arr = [1,2,3,4,5,6]
37 console.log(arr.splice(3,[1]))//[4]
38 console.log(arr);// [1,2,3,5,6]
39
40 var arr = [1,2,3,4,5,6]
41 console.log(arr.splice(3,{}))//[]
42 console.log(arr);// [1,2,3,4,5,6]
43
44 var arr = [1,2,3,4,5,6]
45 console.log(arr.splice(3,null))//[]
46 console.log(arr);// [1,2,3,4,5,6]
47
48 var arr = [1,2,3,4,5,6]
49 console.log(arr.splice(3,undefined))//[]
50 console.log(arr);// [1,2,3,4,5,6]
51
```

第二个参数表示截取位数：

普通数字就直接表示截取多少位，负数表示截取不到，超过数组长度的表示截取到最后一位

数字型字符串和单位数字数组会转成对应数字表示截取多少位

`true`和`false`会转成数字1和0，分别表示截取1位和0位

对象、非数字型字符串、非单位数字数组以及`null`、`undefined`都表示截取不到，返回`[]`

剩下从第三个参数开始就表示要添加的数据了。

经过上述的测试，我们发现，原来前面两个参数存在隐式类型转换，都是将传入的数据通过`Number()`转成了数字类型，再代入函数中。

这个函数会比较麻烦比较难理解，个人未实现。网上看到有源码大家可以参考：[JS中从Array.slice\(\)与Array.splice\(\)的底层实现原理分析区别](#)—作者：[coder\\_chenz](#)

## 7. sort()

用于对数组成员进行排序，默认是按照ASCII码顺序排序

注意。`sort`方法不是按照大小排序，而是按照对应字符串的字典顺序排序。也就是说，数值会被先转成字符串，再按照字典顺序进行比较，所以101排在11的前面。

如果想让`sort`方法按照自定义方式排序，可以传入一个函数作为参数，该函数本身又接受两个参数，表示进行比较的两个元素。如果比较返回大于0，表示第一个元素排在第二个元素后面；否则第一个元素排在第二个元素前面。

传入的函数的规则是：

1. 必须写2个形参；
2. 当返回值为负数时，那么前面的数放在前面；  
当返回值为正数时，那么后面的数放在前面；

写下你的评论...

评论0

赞1

...





```
3 //return b-a; 两数  
4 });
```

给一个有序数组乱序：

```
1 | arr.sort(function(){return Math.random()-0.5})
```

里面返回的是随机正负，所以是随机换位置或者不换位置实现乱序。

要实现这个sort函数会比较麻烦，个人未实现，我在网上找到V8引擎中sort的源码，给大家研究：

```
1 function InnerArraySort(array, length, comparefn) {  
2   // In-place QuickSort algorithm.  
3   // For short (length <= 22) arrays, insertion sort is used for efficiency.  
4  
5   if (!IS_CALLABLE(comparefn)) {  
6     comparefn = function (x, y) {  
7       if (x === y) return 0;  
8       if (%_IsSmi(x) && %_IsSmi(y)) {  
9         return %SmiLexicographicCompare(x, y);  
10      }  
11      x = TO_STRING(x);  
12      y = TO_STRING(y);  
13      if (x == y) return 0;  
14      else return x < y ? -1 : 1;  
15    };  
16  }  
17  var InsertionSort = function InsertionSort(a, from, to) {  
18    for (var i = from + 1; i < to; i++) {  
19      var element = a[i];  
20      for (var j = i - 1; j >= from; j--) {  
21        var tmp = a[j];  
22        var order = comparefn(tmp, element);  
23        if (order > 0) {  
24          a[j + 1] = tmp;  
25        } else {  
26          break;  
27        }  
28      }  
29      a[j + 1] = element;  
30    }  
31  };  
32  
33  var GetThirdIndex = function(a, from, to) {  
34    var t_array = new InternalArray();  
35    // Use both 'from' and 'to' to determine the pivot candidates.  
36    var increment = 200 + ((to - from) & 15);  
37    var j = 0;  
38    from += 1;  
39    to -= 1;  
40    for (var i = from; i < to; i += increment) {  
41      t_array[j] = [i, a[i]];  
42      j++;  
43    }  
44    t_array.sort(function(a, b) {  
45      return comparefn(a[1], b[1]);  
46    });  
47    var third_index = t_array[t_array.length >> 1][0];  
48    return third_index;  
49  }  
50  
51  var QuickSort = function QuickSort(a, from, to) {  
52    var third_index = 0;  
53    while (true) {  
54      // Insertion sort is faster for short arrays.  
55      if (to - from <= 10) {  
56        InsertionSort(a, from, to);  
57        return;  
58      }  
59      if (to - from > 1000) {  
60        third_index = GetThirdIndex(a, from, to);  
61      }  
62    }  
63  }  
64 }
```

写下你的评论...

评论0

赞1

...

```
66     var v1 = a[to - 1];
67     var v2 = a[third_index];
68     var c01 = comparefn(v0, v1);
69     if (c01 > 0) {
70         // v1 < v0, so swap them.
71         var tmp = v0;
72         v0 = v1;
73         v1 = tmp;
74     } // v0 <= v1.
75     var c02 = comparefn(v0, v2);
76     if (c02 >= 0) {
77         // v2 <= v0 <= v1.
78         var tmp = v0;
79         v0 = v2;
80         v2 = v1;
81         v1 = tmp;
82     } else {
83         // v0 <= v1 && v0 < v2
84         var c12 = comparefn(v1, v2);
85         if (c12 > 0) {
86             // v0 <= v2 < v1
87             var tmp = v1;
88             v1 = v2;
89             v2 = tmp;
90         }
91     }
92     // v0 <= v1 <= v2
93     a[from] = v0;
94     a[to - 1] = v2;
95     var pivot = v1;
96     var low_end = from + 1; // Upper bound of elements lower than pivot.
97     var high_start = to - 1; // Lower bound of elements greater than pivot.
98     a[third_index] = a[low_end];
99     a[low_end] = pivot;
100
101     // From low_end to i are elements equal to pivot.
102     // From i to high_start are elements that haven't been compared yet.
103     partition: for (var i = low_end + 1; i < high_start; i++) {
104         var element = a[i];
105         var order = comparefn(element, pivot);
106         if (order < 0) {
107             a[i] = a[low_end];
108             a[low_end] = element;
109             low_end++;
110         } else if (order > 0) {
111             do {
112                 high_start--;
113                 if (high_start == i) break partition;
114                 var top_elem = a[high_start];
115                 order = comparefn(top_elem, pivot);
116             } while (order > 0);
117             a[i] = a[high_start];
118             a[high_start] = element;
119             if (order < 0) {
120                 element = a[i];
121                 a[i] = a[low_end];
122                 a[low_end] = element;
123                 low_end++;
124             }
125         }
126     }
127     if (to - high_start < low_end - from) {
128         QuickSort(a, high_start, to);
129         to = low_end;
130     } else {
131         QuickSort(a, from, low_end);
132         from = high_start;
133     }
134 }
135 };
```

下一部分将讲解ES3中，不会修改原数组的方法

参考文献

写下你的评论...

评论0

赞1

...



JavaScript常用方法push、pop、shift、unshift、concat、join的封装与使用—作者：麦麦麦兜  
JS中自己封装方法，实现sort、reverse等方法—作者：Curry3Ooo  
数据结构与算法——使用原生js实现js中自带的reverse()方法—作者：tozeroblog  
JS中从Array.slice()与Array.splice()的底层实现原理分析区别—作者：coder\_chenz  
JS V8 引擎中sort的源码



1人点赞&gt;



渡一领跑计划



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



jCodeLife 书山有路勤为径，学海无涯苦作舟

总资产4 (约0.43元) 共写了8.6W字 获得98个赞 共19个粉丝



写下你的评论...

全部评论 0

只看作者

关闭评论

按时间倒序 按时间正序

被以下专题收入，发现更多相似内容

投稿管理

+ 收入我的专题



渡一领跑计划笔记

写下你的评论...

评论 0

赞 1

