## 创建数组的方法?

## 1.字面量创建

var arr = [1,2,"3"];

## 2. Array()方法实例化创建

var arr = new Array(1,2,"3")

## 操作数组的方法?

## 1. indexOf返回元素在数组的索引;

## 2. lastindexOf()返回元素最后一次出现的位置的索引;

## 3. push()向数组的末尾添加元素，返回修改后数组的长度;unshift()向数组的开头添加元素，返回修改后数组的长度;

## 4. pop()删除数组最后一个元素，减少数组的 length 值，然后返回移除的项, 数组为空则返回undefined;shift()删除数组开头元素，减少数组的 length 值，然后返回移除的项, 数组为空则返回undefined;

5. splice（）万能法: splice(起始索引，删除数目，插入的元素) 删除项为0，表示插入元素，删除数目这个参数为0时不能省略，这个参数后面的参数都是插入的元素。

删除项不为0，表示删除后替换，替换元素的数量不必和删除数目一样。

6.arr.sort();按照字符编码的顺序进行从小到大排序, sort()方法会调用每个数组项的 toString()转型方法，然后比较得到的字符串，以确定如何排序。即使数组中的每一项都是数值， sort()方法比较的也是字符串。返回对数组的引用。请注意，数组在原数组上进行排序，不生成副本。

arr.sort(fn)支持传入一个函数自定义排序规则:

function sortNumber(a,b){

if(a < b){

return -1;

}else if(a > b){

return 1;

}else{

return 0;

}

}

var arr = [10,5,40,25,1000,1];

arr.sort(sortNumber);

console.log(arr); // 输出[1,5,10,25,40,1000]

7. reverse()表示反转数组的顺序;

8. slice(start,end), 从arr的索引start开始切取，到索引end结束但是不包括索引end,返回一个新的数组;

只写一个参数,这个参数就是start，第二个参数则默认为数组长度，即从第一个参数开始切取整个数组;

不写参数表示从头到尾切下整个数组.

9. concat():arr1.concat(arr2); 先创建当前数组一个副本，然后将接收到的参数添加到这个副本的末尾，最后返回新构建的数组。concat()方法不会改变原来的数组.

10. join()方法可以拼接一个数组内的元素，并用符号相连。返回一个字符串.默认用逗号为分隔符.

11. includes()

替换 ES5 时代的 indexOf 判断方式。indexOf 判断元素是否为 NaN，会判断错误。

var arr = [NaN, 1, 2, 3]

arr.indexOf(NaN); // -1

arr.includes(NaN); //true

//此外includes接收第二个参数,表示从该下标开始查找

var arr = [1,2,3,4];

arr.includes(2,3); //false,从下标3开始找2元素,不存在,返回false

## 数组排序

冒泡排序:

var arr = [2,3,4,2,6,4,2,1];

for ( i =0;i < arr.length - 1;i ++){

for (j = 0; j < arr.length - 1 -i;j++){

if(arr[j] > arr[j+1]){

arr[j] = [arr[j+1],arr[j+1]=arr[j]][0];

}

}

}

选择排序:

var arr = [2,1,3,2,6,4,2,1];

for(i = 0;i < arr.length - 1; i++){

for(j = i + 1;j < arr.length; j ++){

if(arr[i] > arr[j]){

arr[i] = [arr[j],arr[j] = arr[i]][0];

}

}

}

快速排序:

1.在数据集之中找一个基准点（位于目前的数组的中间的那个数值），

2.建立2个数组，分别存储左边和右边的数组，

3.利用递归进行逐次比较，然后拼接数组达到排序的效果

function jsQuickSort(array) {

if (array.length <= 1) {

return array;

}

const pivotIndex = Math.floor(array.length / 2); //取中间的元素下标

const pivot = array.splice(pivotIndex, 1)[0]; //从数组中取出我们的"基准"元素

const left = [], right = [];

array.forEach(item => {

if (item < pivot) { //left 存放比 pivot 小的元素

left.push(item);

} else { //right 存放大于或等于 pivot 的元素

right.push(item);

}

});

//至此，我们将数组分成了left和right两个部分

return jsQuickSort(left).concat(pivot, jsQuickSort(right)); //分而治之

}

## 数组去重

filter去重:

function filterArr(arr){

var newArr = arr.filter(function(el,index,arr){

return arr.indexOf(el) == index

})

return newArr;

}

indexOf:

function noRepeatArr(arr){

var newArr = [];

for(var i of arr){

if(newArr.indexOf(i) == -1){

newArr.push(i)

}

}

}

Set:

function noRepeatArr(arr){

var set = new Set(arr); // set数据结构没有重复元素

return Array.from(set) // set数据结构转数组

}

sort:

//利用sort排序，相等的元素会排在紧邻的位置，判断紧邻的位置是否相等

function noRepeatArr(arr){

arr.sort();

var newarr = [];

for(var i = 0;i < arr.length;i++){

if(arr[i] !== arr[i + 1]){

newarr.push(arr[i])

}

}

return newarr;

}

## 数组扁平化

flat():

// 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。用Infinity表示无论多少层都拉平为1维数组。

var arr = [1,[2,3],[4,[5,6]],[7,[8,[9,10]]]];

arr.flat(Infinity); // i大写

arr; // [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

递归:

function fn(arr,tmp = []){

for(var i = 0;i < arr.length;i++){

if(Array.isArray(arr[i])){

tmp = tmp.concat(fn(arr[i])) //递归调用fn

}else{

tmp.push(arr[i])

}

}

return tmp;

}

## 数组循环

普通for循环;

优化版for循环:

//先用变量保存数组长度，避免每次循环计算数组长度，在数组长度特别大的时候优化明显。

var arr =[1,2,0,3,9];

var len = arr.length;

for(var i = 0;i < len; i++){

console.log(i + ":" + arr[i])

}

for of循环:

// for of 用于遍历数组中的每一个值，不可用于遍历对象获取对象的属性，如果数组有自定义属性，在遍历时不会获取自定义属性。

var arr = [a,b,c,d];

for(var i of arr){

console.log(i)

}

//输出a,b,c,d遍历数组的值，不是下标，直接遍历数组的每一个值.

for of无法遍历对象,但是可以通过Object.entries, Object.keys, Object.values间接遍历对象的键值:

var person = {

name:"ksi",

age:19

}

for(var [k,v] of Object.entries(person)){

console.log([k,v])

}

打印: ["name", "ksi"] ["age", 19]

for in循环

for-in是为普通对象设计的，用来遍历数组的话除了遍历数组元素外，还会遍历自身可枚举属性,因此不适用于数组遍历。

var arr = [4,5,6,7,8];

for(var i in arr){

console.log(i);

}

// 打印0,1,2,3，4,遍历数组的下标

forEach循环

// forEach(function(item,index,arr))遍历数组全部元素，利用回调函数对数组进行操作.

// forEach无法break中途跳出循环,因此不可控

// forEach不支持return操作输出，return只用于控制循环是否跳出当前循环

var arr = [1,2,3,4,5];

arr.forEach(function(item,index,arr){

console.log(item); // 1,2,3,4,5

console.log(index); // 0,1,2,3,4

console.log(arr) // [1,2,3,4,5]

})

filter()循环

// filter()接收一个函数。filter()把传入的函数依次作用于每个元素，然后根据返回值是true还是false决定保留还是丢弃该元素。

var arr = [2,2,3,4,5,6];

var arr2 = arr.filter(function(a,b,c){

return a % 2 == 0;

}); // 输出[2,2,4,6]

every()循环

// 对数组的每一项都运行给定的函数，每一项都返回 ture,则返回 true,否则返回false。

var arr =[1,2,5,3,2];

var tmp = arr.every(function(cur){

return cur > 3

});

console.log(tmp); // tmp = false;

some()循环

// 对数组的每一项都运行给定的函数，任意一项返回 ture,则返回 true,只有每一项都为false才返回false。

var arr =[1,2,5,3,2];

var tmp = arr.some(function(cur){

return cur > 3

});

console.log(tmp); // tmp = true;

map()循环

// 对数组的每一项都运行给定的函数，返回每次函数调用的结果组成一个新数组.

var arr = [1,2,4,6,3];

var newarr = arr.map(function(cur,index,arr){

return cur > 3;

}) // 返回[false,false,true,true,true]

// 把数组每一位都变成偶数。

var arr = [1,2,3,4,5,6];

var newarr = arr.map(function(cur){

if(cur % 2 != 0){

return cur + 1

}else{

return cur

}

}) // 输出[2,2,4,4,6,6];

reduce (callback,initvalue)循环

// reduce() 方法接收一个函数作为累加器（accumulator），数组中的每个值（从左到右）开始缩减，最终为一个值。

// callback接受四个参数：初始值（或者上一次回调函数的返回值），当前元素值，当前索引，调用 reduce 的数组。

var arr = [1, 2, 3, 4, 5];

sum = arr.reduce(function(pre, cur, index, arr) {

console.log(pre, cur, index);

return pre + cur;

})

console.log(sum);

/\*结果：

1 2 1

3 3 2

6 4 3

10 5 4

15

\*/

var arr = [1, 2, 3, 4];

multiplue= arr.reduce(function(pre, cur, index, arr) {

console.log(pre, cur, index);

return pre \* cur - 2;

})

console.log(multiplue);

/\*结果：

1 2 1

0 3 2

-2 4 3

-10

\*/