冒泡排序：这个算法的名字由来是因为越大的元素会经由交换慢慢“浮”到数列的顶端（升序或降序排列）

```
 /*
 原理：数组中有n个数，比较相邻的两个数，如果后面的数小于前面的数，则两者交换。一轮完成后，此时最大的数据已经在最后，此时进行第二轮，确定倒数第二个数，依次几轮就可完成：
 */
 const arr = [12, 32, 54, 36, 74, 98, 3];//首先创建一组数组
         for (let i = 0; i < arr.length; i++) {//循环里面的每一项
             for (let j = 0; j < arr.length - 1 - i; j++) {//j代表其中一个项，然后循环，-i的目的是为了更精准，少循环几次，让代码运行速度更快
                 if(arr[j] < arr[j + 1]){//判断第一项与他下一项的大小
                    let num = arr[j];//抓取一个最小值
                    arr[j] = arr[j + 1];//这个可以理解为换位置，它符合判断的话，j 的位置就去了 j+1 的位置
                    arr[j + 1] = num;//所以j+1的位置就是 j 的数，也就是 num 抓取的这个值
                 }

             }
         }
         console.log(arr)//  arr = [98, 74, 54, 36, 32, 12, 3]
```

2. 剩下的一半里，有些写的比较好，会写成函数，有参数和返回值；多数经过提醒，也能写成函数。但是少数人一脸茫然，表示不明白，即使我直接说“参数用List”这种程度，也写不出参数。然后聊聊一些技术要点，和他做过的项目。
3. 如果通过了冒泡，就让写快排；写不出也不要紧，在纸上随便划划原理。还有余力的，再聊聊二叉树，能写二叉树的代码当然更好，写不出来也不要紧(如果连听说都没听说过二叉树的就算了)。然后是他做过的项目；技术要点；设计模式；项目架构。

冒泡排序

```
var array = [5, 4, 3, 2, 1];
var temp = 0;
for (var i = 0; i <array.length; i++){
for (var j = 0; j <array.length - i; j++){
if (array[j] > array[j + 1]){
temp = array[j + 1];
array[j + 1] = array[j];
array[j] = temp;
 }
}
```

字符串分割

```
const str ='data-of-list-cell'
function toCamel(str){
  //字符串分割
  let temp=str.split("-");
  //每一个的首字母转换
   for(let i=1;i<temp.length;i++){
           temp[i]=temp[i][0].toUpperCase()+temp[i].slice(1);
       }
    return temp.join("");
}
```

sort排序

```
/*
sort()函数 sort()函数主要是对数组进行正序排序,比如: 也可以使用含有参数的sort()方法进行排序,
比如下面的两个例子,二者的效果是一致的。例子1:
2.reverse()函数 reverse()函数主要用于倒叙排序
*/
var data = {
 "rows": [{
     "name": "张三",
     "time": "2011/4/1 0:00:00",
 },{
     "name": "李四",
     "time": "2015/5/6 12:30:00",
 },{
     "name": "王五",
     "time": "2012/10/1 22:10:00",
 },{
     "name": "赵六",
     "time": "2011/9/1 22:10:00",
 }]
};
var rows = data.rows;
rows.sort(function(a,b){
 return Date.parse(a.time) - Date.parse(b.time);//时间正序
});
for(var i =0,l=rows.length;i<l;i++){
 console.log(rows[i].name + " | " + rows[i].time);
}
```

js数组 start<==========>

js数组 start<==========>

```
map: 遍历数组，返回回调返回值组成的新数组forEach: 无法break，可以用try/catch中throw new Error来停止
filter: 过滤
some: 有一项返回true，则整体为true
every: 有一项返回false，则整体为false
join: 通过指定连接符生成字符串
push / pop: 末尾推入和弹出，改变原数组， 返回推入/弹出项【有误】
unshift / shift: 头部推入和弹出，改变原数组，返回操作项【有误】
sort(fn) / reverse: 排序与反转，改变原数组
concat: 连接数组，不影响原数组， 浅拷贝
slice(start, end): 返回截断后的新数组，不改变原数组
splice(start, number, value...): 返回删除元素组成的数组，value 为插入项，改变原数组
indexOf / lastIndexOf(value, fromIndex): 查找数组项，返回对应的下标
reduce / reduceRight(fn(prev, cur), defaultPrev): 两两执行，prev 为上次化简函数的return值，cur 为当前值(从第二项开始)
```

js 数组

forEach

```
const arr = [1,2,3,4,5]
arr.forEach((item)=>{
  console.log("item",item)
})
```

sort() 方法用于对数组的元素进行排序

```
const sortArr = [10, 5, 40, 25,1000,1]
console.log("sort第一种方案：",sortArr.sort()) //没有排序
console.log("sort第二种方案：",sortArr.sort((x,y)=>{
  return x-y
}))
// 2.1 sort 遍历对象数组
const arrObj = [{name:'zopp',age:0},{name:'gpp',age:18},{name:'yjj',age:8}]
const sortObj = arrObj.sort((a,b)=>{
  const age1 = a['age']
  const age2 = b['age']
  return age1 - age2
})
console.log("sort对象",sortObj)
```

实战:

```
//结构类似：{categoryCode: 2, keyWords: "电视", sort: 2}
//2020.5.11 新规则：顺序调整为：电视2-空调3-冰箱4-洗衣机5-工程机6 start
supplier.sort((a, b) => {
console.log('a--', a)
console.log('b--', b)
return a['sort'] - b['sort']
})
console.log('排序后：', supplier)
//2020.5.11 新规则：顺序调整为：电视-空调-冰箱-洗衣机-工程机end
```

array.filter：它用于把Array的某些元素过滤掉，然后返回剩下的元素

```
callback 被调用时传入三个参数：
1.元素的值
2.元素的索引
3.被遍历的数组本身

const fArr = [2,4.8]
const fArrTemp = fArr.filter((val,index,e)=>{
  return val>3
})
console.log("filter_",fArrTemp)
// 3.1所以可以巧妙地用来去重
const fArr2 = ['apple', 'strawberry', 'banana', 'pear', 'apple', 'orange',
'orange', 'strawberry']
const fArr2Temp = fArr2.filter((val,index,e)=>{
  return e.indexOf(val) === index
})
console.log("filter2_",fArr2Temp)
```

扩展去重：forEach+indexOf() ---->indexOf() 方法可返回某个指定的字符串值在字符串中首次出现的位置，如果没有找到返回-1

includes() 方法用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回 true，否则返回 false。

```
const array1 = [1, 2, 3];

console.log(array1.includes(2));
// expected output: true
```

```
function unique(arr){
   const tempArr = []
   arr.forEach(item=>{
     if(tempArr.indexOf(item)===-1){
       tempArr.push(item)
     }
   })
   return tempArr
}
```

filter的原理,也就是把函数传进去:fn,然后在内部执行就是了

```
Array.prototype.filter1 = function (fn) {
  if (typeof fn !== "function") {
  throw new TypeError(`${fn} is not a function`);
  }
  let newArr = [];
  for(let i=0; i< this.length; i++) {
    console.log("原理：",this[i])  //这里的this就是调用者，也就是数组
    // console.log("原理2",fn(this[i])) //返回false或则true
    fn(this[i]) && newArr.push(this[i]);
  }
  return newArr;
}
let arrFilter=[2,4,6,8];
```

```
let arrFilterTemp=arrFilter.filter1((item)=>{
    return item>5
})
console.log("原理",arrFilterTemp) //[6,8]
```

map: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

```
const array1 = [1, 4, 9, 16];
const map1 = array1.map(x => x * 2);
console.log("map__",map1);
//参数：
/*
callback
生成新数组元素的函数，使用三个参数：
  currentValue,callback 数组中正在处理的当前元素。
  index可选,callback 数组中正在处理的当前元素的索引。
  array可选,map 方法调用的数组。
*/
// 4.1.原理：
Array.prototype.newMap = function(fn) {
const newArr = [];
for(var i = 0; i<this.length; i++){
    newArr.push(fn(this[i],i,this))
  }
  return newArr;
}
```

find()方法返回数组中满足提供的测试函数的第一个元素的值。否则返回 undefined

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find(element => element > 10);
console.log(found); // expected output: 12
/*
callback
在数组每一项上执行的函数，接收 3 个参数：
  element当前遍历到的元素。
  index可选当前遍历到的索引。
  array可选数组本身。
*/
// 5.2.find()原理：
Array.prototype.find = function(fn) {
    if (typeof fn !== "function") {
  throw new TypeError(`${fn} is not a function`);
    }
    for (let i = 0; i < this.length; i++) {
        if (fn(this[i])) return this[i]
    }
}
```

findIndex()方法返回数组中满足提供的测试函数的第一个元素的索引,若没有找到对应元素则返回-1

```
/*
callback针对数组中的每个元素，都会执行该回调函数，执行时会自动传入下面三个参数:
  element当前元素。
  index当前元素的索引。
  array调用findIndex的数组。
*/
const array1 = [5, 12, 8, 130, 44];
const isLargeNumber = (element) => element > 13;
console.log(array1.findIndex(isLargeNumber)); //3
```

js数组 end<==========>

js数组 end<==========>