WEB前端必须掌握的一些算法题

Q1 判断一个单词是否是回文?

> 回文是指把相同的词汇或句子，在下文中调换位置或颠倒过来，产生首尾回环的情趣，叫做回文，也叫回环。比如 mamam redivider .

很多人拿到这样的题目非常容易想到用for 将字符串颠倒字母顺序然后匹配就行了。其实重要的考察的就是对于reverse的实现。其实我们可以利用现成的函数，将字符串转换成数组，这个思路很重要，我们可以拥有更多的自由度去进行字符串的一些操作。

```
1. function checkPalindrom(str) {
2.     return str == str.split('').reverse().join('');
3. }
4.
```

Q2 去掉一组整型数组重复的值

1. 比如输入：[1,13,24,11,11,14,1,2]
2. 输出：[1,13,24,11,14,2]
3. 需要去掉重复的11 和 1 这两个元素。

这道问题出现在诸多的前端面试题中，主要考察个人对Object的使用，利用key来进行筛选。

```
1.
2.
3.
4. let unique = function(arr) {
5.   let hashTable = {};
6.   let data = [];
7.   for(let i=0,l=arr.length;i<l;i++) {
8.     if(!hashTable[arr[i]]) {
9.       hashTable[arr[i]] = true;
10.      data.push(arr[i]);
11.    }
12.  }
13.  return data
14. }
15.
16. module.exports = unique;
17.
18.
```

Q3 统计一个字符串出现最多的字母

给出一段英文连续的英文字符窜，找出重复出现次数最多的字母

```
1. 输入 ： afjghdfraaaasdenas
2.
3. 输出 ： a
```

前面出现过去重的算法，这里需要是统计重复次数。

```
1. function findMaxDuplicateChar(str) {
2.   if(str.length == 1) {
3.     return str;
4.   }
5.   let charObj = {};
6.   for(let i=0;i<str.length;i++) {
7.     if(!charObj[str.charAt(i)]) {
8.       charObj[str.charAt(i)] = 1;
9.     }else{
10.      charObj[str.charAt(i)] += 1;
11.    }
12.  }
13.  let maxChar = '',
14.      maxValue = 1;
15.  for(var k in charObj) {
16.    if(charObj[k] >= maxValue) {
17.      maxChar = k;
18.      maxValue = charObj[k];
19.    }
20.  }
21.  return maxChar;
22. }
```

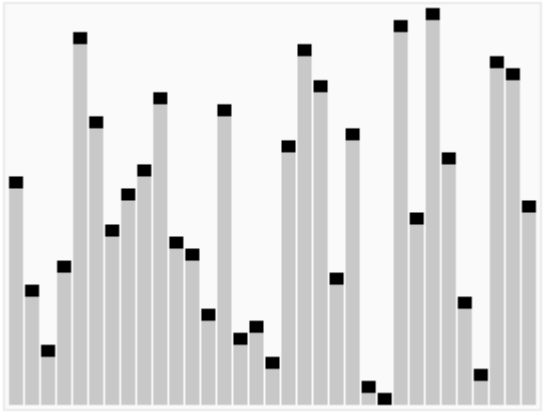
```
25. module.exports = findMaxDuplicateChar;
26.
```

Q4 排序算法

如果抽到算法题目的话，应该大多都是比较开放的题目，不限定算法的实现，但是一定要求掌握其中的几种，所以冒泡排序，这种较为基础并且便于理解记忆的算法一定需要熟记于心。冒泡排序算法就是依次比较大小，小的的大的进行位置上的交换。

```
1. function bubbleSort(arr) {
2.     for(let i = 0,l=arr.length;i<l-1;i++) {
3.         for(let j = i+1;j<l;j++) {
4.           if(arr[i]>arr[j]) {
5.                 let tem = arr[i];
6.                 arr[i] = arr[j];
7.                 arr[j] = tem;
8.             }
9.         }
10.     }
11.     return arr;
12. }
13. module.exports = bubbleSort;
14.
```

除了冒泡排序外，其实还有很多诸如 插入排序,快速排序，希尔排序等。每一种排序算法都有各自的特点。全部掌握也不需要，但是心底一定要熟悉几种算法。比如快速排序，其效率很高，而其基本原理如图(来自wiki)：



算法参考某个元素值，将小于它的值，放到左数组中，大于它的值的元素就放到右数组中，然后递归进行上一次左右数组的操作，返回合并的数组就是已经排好顺序的数组了。

```
function quickSort(arr) {

    if(arr.length<=1) {
        return arr;
    }
    let leftArr = [];
    let rightArr = [];
    let q = arr[0];
    for(let i = 1,l=arr.length; i<l; i++) {
        if(arr[i]>q) {
            rightArr.push(arr[i]);
        }else{
            leftArr.push(arr[i]);
        }
    }
    return [].concat(quickSort(leftArr),[q],quickSort(
```

```
}
module.exports = quickSort;

```

安利大家一个学习的地址，通过动画演示算法的实现。

[HTML5 Canvas Demo: Sorting Algorithms]

Q5 不借助临时变量，进行两个整数的交换

输入 a = 2, b = 4 输出 a = 4, b =2

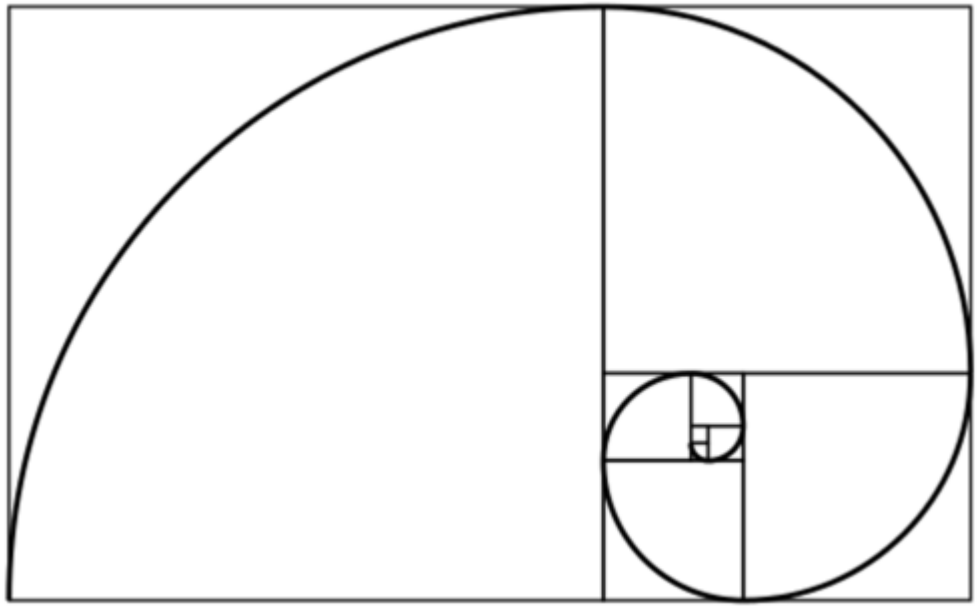
这种问题非常巧妙，需要大家跳出惯有的思维，利用 a , b进行置换。

主要是利用 + - 去进行运算，类似 a = a + (b - a) 实际上等同于最后 的 a = b;

```
function swap(a , b) {
  b = b - a;
  a = a + b;
  b = a - b;
  return [a,b];
}
module.exports = swap;

```

Q6 使用canvas 绘制一个有限度的斐波那契数列的曲线?



数列长度限定在9.

斐波那契数列，又称黄金分割数列，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……在数学上，斐波纳契数列主要考察递归的调用。我们一般都知道定义

```
1. fibo[i] = fibo[i-1]+fibo[i-2];
2.
```

生成斐波那契数组的方法

```
1. function getFibonacci(n) {
2.   var fibarr = [];
3.   var i = 0;
```

```
  while(i<n) {
    if(i<=1) {
      fibarr.push(i);
    }else{
      fibarr.push(fibarr[i-1] + fibarr[i-2])
    }
    i++;
  }
  return fibarr;
}

```

剩余的工作就是利用canvas arc方法进行曲线绘制了

DEMO

Q7 找出下列正数组的最大差值比如:

```
1. 输入 [10,5,11,7,8,9]
2.
3. 输出 6
```

这是通过一道题目去测试对于基本的数组的最大值的查找，很明显我们知道，最大差值肯定是一个数组中最大值与最小值的差。

```
1.  function getMaxProfit(arr) {
2.
```

```
    var minPrice = arr[0];
    var maxProfit = 0;

    for (var i = 0; i < arr.length; i++) {
        var currentPrice = arr[i];
        minPrice = Math.min(minPrice, currentPrice);
        var potentialProfit = currentPrice - minPrice;
        maxProfit = Math.max(maxProfit, potentialProfi
    }
    return maxProfit;
}

```

Q8 随机生成指定长度的字符串

实现一个算法，随机生成指制定长度的字符窜。

比如给定 长度 8 输出 4ldkfg9j

```
function randomString(n) {
  let str = 'abcdefghijklmnopqrstuvwxyz9876543210';
  let tmp = '',
      i = 0,
      l = str.length;
  for (i = 0; i < n; i++) {
    tmp += str.charAt(Math.floor(Math.random() * l));
  }
  return tmp;
```

```
}
module.exports = randomString;

```

Q9 实现类似getElementsByClassName 的功能

自己实现一个函数，查找某个DOM节点下面的包含某个class的所有DOM节点？不允许使用原生提供的 getElementsByClassName querySelectorAll 等原生提供DOM查找函数。

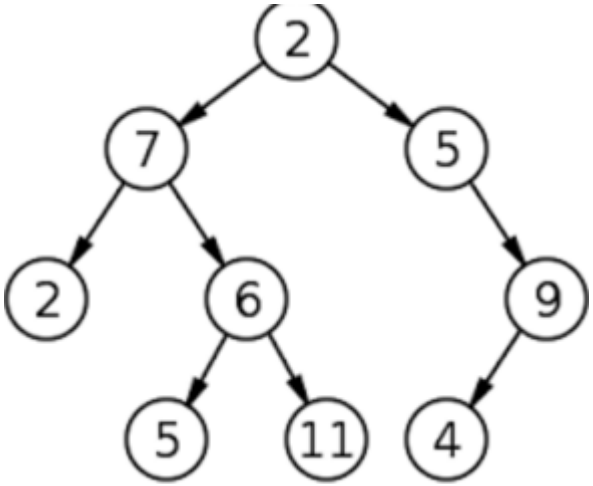
```
function queryClassName(node, name) {
  var starts = '(^|[ \n\r\t\f])',
       ends = '([ \n\r\t\f]|$)';
  var array = [],
       regex = new RegExp(starts + name + ends),
       elements = node.getElementsByTagName("*"),
       length = elements.length,
       i = 0,
       element;
    while (i < length) {
       element = elements[i];
       if (regex.test(element.className)) {
           array.push(element);
       }
       i += 1;
    }
    return array;
}

```

Q10 使用JS 实现二叉查找树(Binary Search Tree)

一般叫全部写完的概率比较少，但是重点考察你对它的理解和一些基本特点的实现。 二叉查找树，也称二叉搜索树、有序二叉树（英语：ordered binary tree）是指一棵空树或者具有下列性质的二叉树：

- 任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；
- 任意节点的右子树不空，则右子树上所有结点的值均大于它的根结点的值；
- 任意节点的左、右子树也分别为二叉查找树；
- 没有键值相等的节点。二叉查找树相比于其他数据结构的优势在于查找、插入的时间复杂度较低。为O(log n)。二叉查找树是基础性数据结构，用于构建更为抽象的数据结构，如集合、multiset、关联数组等。



在写的时候需要足够理解二叉搜素树的特点，需要先设定好每个节点的数据结构

```
1. class Node {
2.   constructor(data, left, right) {
```

```
    this.data = data;
    this.left = left;
    this.right = right;
  }
}

```

树是有节点构成，由根节点逐渐延生到各个子节点，因此它具备基本的结构就是具备一个根节点，具备添加，查找和删除节点的方法.

```
class BinarySearchTree {

  constructor() {
    this.root = null;
  }
  insert(data) {
    let n = new Node(data, null, null);
    if (!this.root) {
      return this.root = n;
    }
    let currentNode = this.root;
    let parent = null;
    while (1) {
      parent = currentNode;
      if (data < currentNode.data) {
        currentNode = currentNode.left;
        if (currentNode === null) {
          parent.left = n;
          break;
```

```
        }
      } else {
        currentNode = currentNode.right;
        if (currentNode === null) {
          parent.right = n;
          break;
        }
      }
    }
  }
  remove(data) {
    this.root = this.removeNode(this.root, data)
  }
  removeNode(node, data) {
    if (node == null) {
      return null;
    }
    if (data == node.data) {

      if (node.left == null && node.right == null) {
        return null;
      }
      if (node.left == null) {
        return node.right;
      }
      if (node.right == null) {
        return node.left;
      }
      let getSmallest = function(node) {
        if(node.left === null && node.right == null) {
```

```
55.           return node;
56.         }
57.         if(node.left != null) {
58.           return node.left;
59.         }
60.         if(node.right !== null) {
61.           return getSmallest(node.right);
62.         }
63.       }
64.       let temNode = getSmallest(node.right);
65.       node.data = temNode.data;
66.       node.right = this.removeNode(temNode.right,temNo
67.       return node;
68.     } else if (data < node.data) {
69.       node.left = this.removeNode(node.left,data);
70.       return node;
71.     } else {
72.       node.right = this.removeNode(node.right,data);
73.       return node;
74.     }
75.   }
76.   find(data) {
77.     var current = this.root;
78.     while (current != null) {
79.       if (data == current.data) {
80.         break;
81.       }
82.       if (data < current.data) {
83.         current = current.left;
84.       } else {
```

```
88.           current = current.right
89.         }
90.       }
91.       return current.data;
92.     }
93. }
94. module.exports = BinarySearchTree;
```

前言

> javascript是一门非常灵活的语言，实际的开发过程中我们也可以灵活的使用它而给我们的工作带来便利，这篇文章记录了自己平时学习过程中经常用到的一些小技巧，整理出来作为笔记，也希望对感兴趣的同学有所帮助。(持续更新...)

1 获取指定范围内的随机数

> 当我们需要获取指定范围(min,max)内的整数的时候，下面的代码非常适合。

```
function getRadomNum(min,max){
return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

测试

```
function getRadomNum(min,max){
    return  Math.floor(Math.random() * (max - min + 1)) + min;
}
var i=0;
while(i<10){
  console.log(getRadomNum(5,20));
  i++;
}
```

```
11
20
17
7
13
6
17
14
8
5
```

2 随机获取数组中的元素

```
function getRadomFromArr(arr){
return arr[Math.floor(Math.random()*arr.length)];
}
```

测试

3 生成从0到指定值的数字数组

```
var arr=[],length=100,i=1;
for(;arr.push(i++)<length;){}
console.log(arr)
```

测试

4 打乱数字数组的顺序

```
var arr = [1,2,3,4,5,6,7,'a','dsfs',8,9,'v'];
arr.sort(function(){return Math.random()-0.5});
```

测试

```
> var arr=[1,2,3,4,5,6,7,'a','dsfs',8,9,'v'];
  arr.sort(function(){return Math.random()-0.5});
  console.log(arr)
["v", 8, 6, 3, 4, "dsfs", 2, 7, 9, 1, "a", 5]
```

5 对象转换为数组

```
1
2  var obj={
3  0:'qian',
4  1:'long',
5  2:'chu',
6  3:'tian',
7  length:4
8  }
9  var _slice=[].slice;
10 var objArr=_slice.call(obj);
11
```

测试

```
> var obj={
      0:'qian',
      1:'long',
      2:'chu',
      3:'tian',
      length:4
  }
  var _slice=[].slice;
  var objArr=_slice.call(obj);
  console.log(objArr)
["qian", "long", "chu", "tian"]
```

6 验证是否为数组

```
1  function isArray(obj){
2  return Object.prototype.toString.call(obj) === '[object Array]' ;
```

```
3 }
```

测试

```
> function isArray(obj){
      return  Object.prototype.toString.call(obj) === '[object Array]' ;
  }
  console.log(isArray([1,2]));
  console.log(isArray({}));
  console.log(isArray(123));
  true
  false
  false
```

7 获取数组中最大或者最小值

```
1 function maxAndMin(arr){
2 return {
3 max:Math.max.apply(null,arr.join(',').split(',')),
4 min:Math.min.apply(null,arr.join(',').split(','))
5 }
6 }
```

该方法适合一维或者多维数组求最大最小值的情况

测试

```
> function maxAndMin(arr){
      return {
         max:Math.max.apply(null,arr.join(',').split(',')),
         min:Math.min.apply(null,arr.join(',').split(','))
      }
  }
  console.log(maxAndMin([1,2,3,4,5,6]));
  console.log(maxAndMin([1,2,3,[4,5,6,7,8],[9,10,11,12,12,[14,15,16]]]));
  Object {max: 6, min: 1}
  Object {max: 16, min: 1}
```

8 清空数组

```
1
2
3
4
5
6
7
8
9
10
11
```

```
var arr=[1,2,3,4,5];
arr.length=0;

var arr=[1,2,3,4,5];
arr.splice(0,arr.length);

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

9 保留指定小数位

```
1
2
```

```
var num =4.345678;
num = num.toFixed(4);
```

10 不要直接使用delete来删除数组中的元素

> 数组在js中也是对象，有时候我们可能会通过delete来删除数组中的元素，但是其实仅仅是将数组的元素的值赋值为了undefined。

```
1
2
3
```

```
var arr=[1,2,3,4,5,'谦龙','雏田'];
delete arr[5];
console.log(arr,arr[5],arr.length);
```

测试

```
var arr=[1,2,3,4,5,'谦龙','雏田'];
    delete arr[5];
    console.log(arr,arr[5],arr.length);
[1, 2, 3, 4, 5, 6: "雏田"] undefined 7
```

赋值为undefined 长度依然是7

可以通过splice来删除数组中的某一项

```
var arr=[1,2,3,4,5,'谦龙','雏田'];
arr.splice(5,1);
console.log(arr,arr[5],arr.length);
```

测试

```
var arr=[1,2,3,4,5,'谦龙','雏田'];
    arr.splice(5,1);
    console.log(arr,arr[5],arr.length);
[1, 2, 3, 4, 5, "雏田"] "雏田" 6
```

这个元素已经被删除 而且长度也变为了6

11 生成指定长度的随机字母数字字符串

```
function getRandomStr(len) {
var str = "";
for( ; str.length < len; str += Math.random().toString(36).substr(2));
return str.substr(0, len);
}
```

测试

```
function getRandomStr(len) {
    var str = "";
    for( ; str.length < len; str  += Math.random().toString(36).substr(2));
    return  str.substr(0, len);
}
console.log(getRandomStr(400));
console.log(getRandomStr(40));
console.log(getRandomStr(4));
13fv6fu02sdobt9bkzpsazdoc62bj4i98ekqv3scfas8aorig1imp3svww2ke29w554g3q2ss9py14i9yob1572f1hmpld10v0wmgyvsjmm42t9ghjx2tt7tjvjwcdic50mvxmolo1dcxrww71z1q3hyw4s4iv1f1eykkceze61orog3i6y4utur3sorzxnckv
9n8z11sjorg4muo41u5kp7rpb9yd24g7cc17gv61xr03bftpsf53jz1a4i7g8rfvrikb4vx6ro9373qm2kfublnmifd7whyj9g1g4x6rjynfdw4bc7v3rf6rtxu9m2nua3df5hfr2mvelt36sazia4iykawkot105wv1jorlb2hpqx182r9t3xrqjmsm6csdmj
qncdiv7e7y5m
pre3ydihth2g1dl2rlng149ov1xxbt9oif4gd6k8
Bukp
```

12 null 与 undefined

> null == undefined,null == null 返回true,有时候我们为了排除null 和 undefined可以使用如下的代码

```
function test(obj){
if(obj!=null){
....这里写代码逻辑
}
}
```

13 找出数组中出现次数最的元素，并给出其出现过的位置

```
function getMaxAndIndex( arr ){
var obj = {};
arr.forEach(function(item,index){
if(!obj[item]){
obj[item]= {indexs: [index]}
}else{
obj[item]['indexs'].push(index);
}
});
var num=0;
var str='';
var reArr;
for(var attr in obj){
var temp=obj[attr]['indexs'];
if(temp.length>num){
num=temp.length;
str=attr;
reArr=temp;
}
}
return {
```

```
22  maxStr:str,
23  indexs:reArr
24  }
25  }
```

测试结果