JS编程实战100面试题（https://github.com/minsion/）

面试题 1. Javascript 编写一个b继承a的方法?

推荐指数： ★★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

参考代码如下：

```
function A(name){
this.name = name;
this.sayHello = function(){alert(this.name+” say Hello!”);};
}
function B(name,id){
this.temp = A;
this.temp(name); //相当于new A();
delete this.temp;
this.id = id;
this.checkId = function(ID){alert(this.id==ID)};
}
```

面试题 2. JavaScript 如何消除一个数组里面重复的元素?

推荐指数： ★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

1.使用Set对象：将数组转换为Set对象，Set对象会自动去除重复值，然后将Set对象再转换为数组。例如：

```
let arr = [1, 2, 2, 3, 4, 4, 5];
let uniqueArr = Array.from(new Set(arr));
console.log(uniqueArr); // [1, 2, 3, 4, 5]
```

2.使用filter()方法：遍历数组，使用filter()方法筛选出第一次出现的元素。例如：

```
let arr = [1, 2, 2, 3, 4, 4, 5];
let uniqueArr = arr.filter((value, index, array) => {
return array.indexOf(value) === index;
});
console.log(uniqueArr); // [1, 2, 3, 4, 5]
```

3.使用reduce()方法：使用reduce()方法遍历数组，将第一次出现的元素存入一个新数组中。例如：

```
let arr = [1, 2, 2, 3, 4, 4, 5];
let uniqueArr = arr.reduce((prev, curr) => {
if (!prev.includes(curr)) {
prev.push(curr);
}
```

```
return prev;
}, []);
console.log(uniqueArr); // [1, 2, 3, 4, 5]
```

4.使用对象属性：遍历数组，将元素作为对象的属性，利用对象属性的唯一性去除重复值，然后将对象属性转换为数组。例如：

```
let arr = [1, 2, 2, 3, 4, 4, 5];
let obj = {};
arr.forEach((value) => {
obj[value] = true;
});
let uniqueArr = Object.keys(obj).map((value) => Number(value));
console.log(uniqueArr); // [1, 2, 3, 4, 5]
```

以上是一些去除数组中重复值的方法，每种方法都有其优缺点，应根据具体场景选择最适合的方法。

面试题 3. Javascript 实现数组快速排序?

推荐指数：★★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

关于快排算法的详细说明整个排序过程只需要三步：

（1）在数据集之中，选择一个元素作为"基准"（pivot）。
（2）所有小于"基准"的元素，都移到"基准"的左边；所有大于"基准"的元素，都移到"基准"的右边。
（3）对"基准"左边和右边的两个子集，不断重复第一步和第二步，直到所有子集只剩下一个元素为止。

参考代码：

```
var quickSort = function(arr) {
    if (arr.length <= 1) { return arr; }
    var pivotIndex = Math.floor(arr.length / 2);
    var pivot = arr.splice(pivotIndex, 1)[0];
    var left = [];
    var right = [];
    for (var i = 0; i < arr.length; i++){
        if (arr[i] < pivot) {
            left.push(arr[i]);
        } else {
            right.push(arr[i]);
        }
    }
    return quickSort(left).concat([pivot], quickSort(right));
};
```

面试题 4. JavaScript实现二分法查找

推荐指数：★★★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

二分法查找，也称折半查找，是一种在有序数组中查找特定元素的搜索算法。查找过程可以分为以下步骤：

（1）首先，从有序数组的中间的元素开始搜索，如果该元素正好是目标元素（即要查找的元素），则搜索过程结束，否则进行下一步。

（2）如果目标元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半区域查找，然后重复第一步的操作。

（3）如果某一步数组为空，则表示找不到目标元素。

参考代码：

```
// 非递归算法
function binary_search(arr, key) {
var low = 0,
high = arr.length - 1;
while(low <= high){
var mid = parseInt((high + low) / 2);
if(key == arr[mid]){
return mid;
}else if(key > arr[mid]){
low = mid + 1;
}else if(key < arr[mid]){
high = mid -1;
}else{
return -1;
}
}
};
var arr = [1,2,3,4,5,6,7,8,9,10,11,23,44,86];
var result = binary_search(arr,10);
alert(result); // 9 返回目标元素的索引值

// 递归算法
function binary_search(arr,low, high, key) {
if (low > high){
return -1;
}
var mid = parseInt((high + low) / 2);
if(arr[mid] == key){
return mid;
}else if (arr[mid] > key){
high = mid - 1;
return binary_search(arr, low, high, key);
}else if (arr[mid] < key){
low = mid + 1;
return binary_search(arr, low, high, key);
```

```
}
};
var arr = [1,2,3,4,5,6,7,8,9,10,11,23,44,86];
var result = binary_search(arr, 0, 13, 10);
alert(result); // 9 返回目标元素的索引值
```

面试题 5. 实现一个函数clone，可以对JavaScript中的5种主要的数据类型进行值复制?

推荐指数: ★★★★★ 试题难度: 高难 试题类型: 编程题

试题回答参考思路:

```
Object.prototype.clone = function(){
var o = this.constructor === Array ? [] : {};
for(var e in this){
o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];
}
return o;
}
```

面试题 6. 编写JavaScript方法，求一个字符串的长度（单位是字节）？

假设一个英文字符占用一字节，一个中文字符占用两字节

推荐指数: ★★★ 试题难度: 初级 试题类型: 编程题

试题回答参考思路:

```
function GetBytes（str）{
var len=str .length;
var bytes= len;
for（var i-0：i if （str. charcodeAt （i）>255） bytes++;
}
return bytes;
}
alert（ GetBytes（"hello 有课前端网！"））；
```

面试题 7. ["1，"2，"3"].map（ parseInt）的执行结果是多少?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

[1， NaN,NaN]，因为 parseInt需要两个参数（val, radix），其中 radix表示解析时用的基数（进制）；map传递了3个参数（item, index,aray），对应的radix不合法导致解析失败。

面试题 8. 如何统计字符串“ aaaabbbccccddfgh”中字母的个数或统计最多的字母数?

推荐指数: ★★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

具体代码如下

```
var str =aaaabbbecccddfgh";
function dealstr (str) {
var obj={};
for (var i= 0; i< str length: i++) {
var v=str.charAt (i) ;
if (obj[v] && obj [v].value === v) {
++obj[v]. count
} else {
obj[v] ={
count: 1,
va⊥ue:v
}
}
}
return obj ;
}
var obj= dealstr (str) ;
for (key in obj) {
console. log (obj[key] .value +'=' obj[ key].count)
}
```

面试题 9. 写JavaScript function，清除字符串前后的空格（兼容所有浏览器）

推荐指数: ★★★ 试题难度: 初级 试题类型: 编程题

试题回答参考思路:

具体代码如下

```
function trim (str) {
if (str && typeof str === "string") {
return str.replace (/^\s+1\s+$/g, "") ; //去除前后空白符。
```

面试题 10. 用 JavaScript实现一个数组合并的方法（要求去重）？

推荐指数: ★★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

代码如下。

```
var arrl =['a'];
var arr2 =['b',  'c'];
var arr3=['c', ['d'], 'e',  undefined, null];
var concat = ( function() {
//去重合并arr1和arr2
var _concat =function  (arrl, arr2)
for  (var i =0,  len= arr2.length;  i< len;  i++){
~ arrl. indexOf (arr2[i])|| arrl. push(arr2[i])
}
}
//返回数组去重合并方法
return function(){
var result =[];
for (var i=0,  len= arguments .length; i< len: i++){
_concat  (result, arguments [i])
return result
}
})()
```

执行concat（arrl,ar2，ar3）后，会返回['a',null, undefined，'e'，['d],'c','b']。

面试题 11. 说明正则表达式给所有string对象添加去除首尾空白符的方法（trim方法）？

推荐指数：★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

代码如下

```
prototype. trim= function(){
return this .replace (/^\s+l\s+$/g, " ) ;
};
```

面试题 12. 用 JavaScript实现一个提取电话号码的方法？

推荐指数：★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
var str="12345678901 021-12345678 有课前端网 0418-1234567 13112345678";
var reg=/ (1\d{0}) | (0\d{2,3}\-\d{7,8}) /g;
alert (str.match (reg)
```

面试题 13. JavaScript 运算3+2+“7”的结果是什么?

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

由于3和2是整数，它们将直接相加，同时由于“7”是一个字符串，将会被直连接，因此结果将是57

面试题 14. 简述下面的代码将输出什么到控制合？为什么？

```
unction (){
var a= b= 3;
})();
console. log（"a defined " +（typeof a ！ =='undefined'））；
console. log（"b defined "  （typeof b ！ == 'undefined'））
```

推荐指数：★★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

输出结果如下。
a defined false
b defined true
a和b都定义在函数的封闭范围内，并且都始于var关键字，因此大多数 JavaScript开发人员期望 typeof a和 typeof b在上面的例子中都是 undefined。
然而，事实并非如此。问题在于大多数开发人员将语句vara=b=3错误地理解为是以下声明的简写
var b= 3;
var a= b;
但事实上，var a=b=3实际是以下声明的简写
b= 3;
var a= b;
因此（如果你不使用严格模式），该代码段的输出如下。
a defined false
b defined true
但是，b如何定义在封闭函数的范围之外呢?
既然语句vara=b=3是语句b=3和vara=b的简写，b最终就成为一个全局变量
（因为它没有前缀var关键字），于是需要定义在封闭函数之外。
需要注意的是，在严格模式下（使用 use strict），语句var a=b=3将生成 Reference error；b is not defined的运行时错误，从而避免任何 headfakes或bug

面试题 15. 下面的JavaScript代码将输出什么内容到控制台?

```
var myobject = {
foo:"bar"
func:function(){
var self =this；
console. log("outer func:this foo =" this foo);
console. log(outer func:self. foo =" self foo);
(function(){
console. log（"inner func:this foo =" +this .foo)；
console. log（"inner func:self. foo ="+ self .foo);
```

```
}());
}
};
myobject func();
```

推荐指数：★★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

上面的代码将输出以下内容到控制台。
outer func:this. foo =bar
outer func:self.foo =bar
inner func:this. foo =undefined
inner func:self. foo s bar
在外部函数中，this和self都指向 myObject，因此两者都可以正确地引用和访问foo。
在内部函数中，this不再指向 myObject。结果是，this.foo没有在内部函数中定义，相反，指向到本地的变量self保持在范围内，并且可以访问（在 ECMAScript5之前，在内部函数中this将指向全局的 window对象；反之，内部函数中的this是未定义的）

面试题 16. 以下两个JavaScript函数会返回相同的结果吗？为什么相同或为什么不相同?

```
function fool (){
return {
bar:"hello"
};
}
function foo2(){
return
{
bar:"hello"
};
}
```

推荐指数：★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

这两个函数返回的内容并不相同。通过以下语句查看返回值。
console. log("fool returns:");
console. log(foo1());
console. log("foo2 returns：");
console. log(foo2());
输出结果如下。
fool returns:Object
{
bar:"hello"
}
foo2 returns:undefined
这不仅令人惊讶，而且让人困惑，因为foo2()返回 undefined，却没有任何错误抛出。
原因与这样一个事实有关，即分号在 JavaScript中是一个可选项（尽管省略它们通常是非

常糟糕的形式）。结果是，当碰到foo2()中包含 return语句的代码行时（代码行上没有其他任何代码），分号会立即自动插入返回语句之后，也不会抛出错误，因为代码的其余部分是完全有效的，即使它没有得到调用或做任何事情。
这也说明了在 JavaScript中大括号的位置应该放在语句后面的编程风格更符合 JavaScript的语法要求。

面试题 17. 写出函数 isInteger（x）的实现方法，用于确定x是否是整数？

ECMAScript6引入了一个新的方法，即 Number. isInteger()，它可以用来判断一个数字是否是整数。在 ECMAScript6之前， isInteger的实现会更复杂。在 ECMAScript规格说明中，整数只是在概念上存在，即，数字值总是存储为浮点数值

推荐指数： ★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

考虑到这一点，有如下几种实现方案
function isInteger （x）{
return (x^ 0)===x;
}
下面的解决方法虽然不如上面那个方法优雅，但也是可行的。
function isInteger(x){
return Math. round（x） === x;
请注意， Math. ceil()和 Math. floor()在上面的实现中等同于 Math. round()。
或者使用以下实现方案。
function isInteger (x){
return （typeof x ==='number '） && （x %1===0）；
}
一个比较普遍的错误解决方案如下。
function isInteger(x) {
return parseInt（x, 10）===x;
}
虽然这个以 parseInt函数为基础的方法在x取许多值时能良好地工作，但是一旦x取值相当大，它就会无法正常工作。问题在于 parseInt()在解析数字之前强制把第一个参数转换为字符串。
因此，一旦数目变得足够大，它的字符串就会表达为指数形式（例如1e+21）。
因此， parseInt()函数就会解析1e+21，但当解析到‘e'字符的时候，就会停止解析，因此只会返回1。

面试题 18. 下列代码行1~4如何排序，才能使之能够在执行代码时输出到控制台？为什么？

```
(function(){
console. log(1);
setTimeout(function(){ console. log(2)}， 1000);
setTimeout(function (){ console. log(3)}， 0);
console. log(4)
})();
```

推荐指数： ★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

序号如下
1
4
3
2
让我们先来解释比较显而易见的那部分。
1和4之所以放在前面，是因为它们只是简单调用 console.log（O，而没有任何延迟输出。
2之所以放在3的后面，是因为2是延迟了1000ms（即，ls）之后输出的，而3是延迟了0ms之后输出的。
浏览器有一个事件循环，它会检查事件队列和处理未完成的事件。例如，如果时间发生在后台（例如，脚本的 onload事件），浏览器正忙（例如，处理一个 onclick），那么事件会添加到队列中。当 onclick处理程序完成后，检查队列，然后处理该事件（例如，执行onload脚本）。
同样，如果浏览器正忙。setTimeout0也会把其引用的函数的执行放到事件队列中当setTimeout的第二个参数为0的时候，它代表“尽快”执行指定的函数。具体是指，函数的执行会从事件队列的下一个计时器开始。但是请注意，这不是立即执行：函数不会执行，除非下一个计时器开始计时，这就是为什么在上述例子中调用 console. log（4）
发生在调用 console. log（3）之前（因为调用 console. log（3）是通过 setTimeout完成的，因此会稍微延迟）。

面试题 19. 写一个简单的函数（少于80个字符），要求返回一个布尔值，指明字符串是否为回文结构？

```
下面这个函数在str是回文结构的时候返回true；否则，返回 false
function isPalindrome（str）{
str= str.replace（/\W/g, ' '）.toLowercase();
return （str = str .split（' '）.reverse().join(' ')'));
}
```

推荐指数：★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

例如：
console. log（isPalindrome（"level"））;//true
console. log（isPalindrome（"levels"））；// false
console. log（isPalindrome（"A car, a man, a maraca"））;// logs 'true

面试题 20. 写一个sum方法，在使用任意语法调用时，都可以正常工作？

```
console. log(sum(2,3))；//5console. log(sum(2)(3))；// Outputs 5
```

推荐指数：★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

(至少)有两种方法可以做到。
方法1，代码如下。

```
function sum(x){
if ( arguments. length === 2){
return arguments[0]+ arguments[1];
}
else {return function （y） { return x+ y;
}
}
}
```

在 JavaScript中，函数可以提供到 arguments对象的访问， arguments对象提供传递到函数的实际参数的访问。这时我们能够使用 length属性来确定在运行时传递给函数的参数数量。
如果传递两个参数，那么只须加在一起并返回。
否则，假设它以sum（2）（3）这样的形式调用，所以返回一个匿名函数，这个匿名函数合并了传递到sumO的参数和传递给匿名函数的参数。
方法2，代码如下。

```
function sum（x, y）
{ if（y ！ = undefined）
{ return x+ y ； }else{ return function （y）
{ return x +y}}}
```

当调用一个函数的时候， JavaScript不要求参数的数目匹配函数定义中的参数数量如果传递的参数数量大于函数定义中的参数数量，那么多余参数将简单地被忽略。
另一方面，如果传递的参数数量小于函数定义中的参数数量，那么在函数中引用缺少的参数时将会给一个 undefined值。
所以，在上面的例子中，简单地检查第2个参数是否未定义，就可以确定函数的调用方式

面试题 21. 简述请看下面的代码片段并回答以下问题 ？

```
for (var i = 0； i< 5； i++){
var btn= document.createElement ('button );
btn. appendchild(document createTextNode('Button +i));
btn.addEventListener('click', function(){ console. log（i） ;});
document .body. appendchild （btn） ；
```

（1）当用户单击“ Button4”的时候会输出什么到控制台，为什么?
（2）提供一个或多个可获取当前i的值的实现方案。

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

（1）无论用户单击什么按钮，数字5将总会输出到控制台。这是因为，当调用onclick方法（对于任何按钮）的时候，for循环已经结束，变量i已经获得了5的值。
（2）要让代码工作的关键是，通过传递到一个新创建的函数对象，在每次传递通过for循环时，捕捉到i值。下面是3种可能实现的方法。
可以通过闭包实现对循环变量i的存储

```
for （var i=0： i< 5:1++）{
var btn =document .createElement （'button'） ；
btn.appendchild（document createTextNode （ 'Button ' +i） ） ；
```

```
btn. addEventListener（'click',(function (i) { return function() { console.
1og(i)； }； }) (i) )；
document .body. appendChild(btn);}
或者，可以封装全部调用到新匿名函数中的btn.addEventListener。
for （var i = 0； i< 5:1++){
var btn= document createElement （'button');
btn. appendChild（document .createTextNode('Button '+i));
(function (i){ btn.addEventListener（'click', function(){ console. log(i);
})； })(i);
document .body. appendChild（btn）；
也可以调用数组对象的本地 for Each方法来替代for循环。
Array(5).fill(0). forEach（function （value, i）{
var btn = document createElement（'button '）；
btn.appendChild(document.createTextNode（'Button'+));
btn.addEventListener（'click',function(){ console. log（i）； });
document .body.appendchild （btn）；
```

面试题 22. 下面的JavaScript代码将输出什么到控制台？为什么？

```
var arrl ="john"split(' ');
var arr2= arrl. reverse();
var arr3 ="jones"，split（' ）；
arr2 push(arr3);
console. log（"array 1：length="+ arrl. length +" last="+ arrl. slice(-1));
console.log（"array 2：length="+ arr2 .length+" last="+ arr2. slice (-1));
```

推荐指数：★★★★ 试题难度： 中级 试题类型： 编程题 ▸

试题回答参考思路：

输出结果是如下
"array1:length=5 last=j,o,n,e,s"
“array2:length-5 last=j,o,n,e,s"
在上述代码执行之后，arr1和arr2相同，原因如下。
调用数组对象的 reverse()方法并不只是返回反顺序的阵列，它也反转了数组本身的顺序（即，在这种情况下，指的是arr1）。
reverse()方法返回一个对数组本身的引用（在这种情况下，即arr1）。其结果为，arr2仅仅是一个对arr1的引用（而不是副本）。因此，当对arr2做了任何事情（即当调用arr2 .push（arr3））时，arrl也会受到影响，因为arr1和arr2引用的是同一个对象

面试题 23. 下面的JavaScript代码将输出什么到控制台？

```
console，10g（1+"2"+"2"）；
console.1og（1+ +"2"+"2"）；
console.1og（1+-"1"+"2"）；
console. log（+"1"+"1"+"2");
console. log（"A"-B"+"2"）；
console. log（"A"- "B"+2）；
```

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

上面的代码将输出以下内容到控制台。
"122"
"32"
"02"
"112"
"NaN2"
NaN
JavaScript（ ECMAScript）是一种弱类型语言，它可对值进行自动类型转换，以适应正在执行的操作。
例1:1+"2”+"2”输出"122"。下面分析原因。1+"2”是执行的第一个操作，由于其中一个运算对象（"2"）是字符串， JavaScript会假设它需要执行字符串连接，因此会将1的类型转换为"1"，1+"2"结果就是"12"。然后，"12"+"2"就是"122"。
例2:1++"2"+"2”输出"32"。下面分析原因。根据运算的顺序，要执行的第一个运算是+"2”（第一个"2"前面的“+”被视为一元运算符），因此， JavaScript将"2"的类型转换为数字。其结果是，接下来的运算就是1+2，这当然是3.然后我们需要在个数字和一个字符串之间进行运算（即，3和"2"）。同样， JavaScript会将数值类型转换为字符串，并执行字符串的连接，产生"32”。
例3:1+-"1”+"2”输出"02"。下面分析原因。这里的解释和前一个例子相同，除了此处的一元运算符是“-”，当应用“-”时"1"变为了-1.然后将其与1相加，结果为0，再将其转换为字符串，连接最后的"2”，得到"02"。
例4：十"1″+"1”+"2”输出"12"。下面分析原因。虽然第一个运算对象”1"因为前缀的一元"艹"运算符类型转换为数值，但是当连接到第二个运算对象"I"的时候，立即转换回字符串。最后与"2"连接，产生了字符串"112”。
例5："A""B"+"2”输出"NaN2"。说明：由于运算符“-”对"A"和"B"处理的时候，都不能转换成数值，因此"A"-"B"的结果是NaN，然后再和字符串"2"连接，得到"NaN2”。
例6："A"-"B"+2输出NaN。说明：参见前一个例子，"A"-"B"结果为NaN，但是，应用任何运算符到NaN与其他任何的数字运算对象上，结果仍然是NaN

面试题 24. 下面的递归代码在数组列表偏大的情况下会导致堆栈溢出，在保留递归模式的基础上，怎么解决这个问题?

```
var list= readHugeList ();
var nextListItem function()l{
var item-=list. pop();
if (item){
nextListItem();
}
};
```

潜在的堆栈溢出可以通过修改 nextListItem函数避免。

```
var list readHugeList();
var nextListItem function(){
var item =list .pop();
if (item) {
setTimeout （ nextListItem, 0）；
```

```
}
};
```

推荐指数：★★★ 试题难度： 初级 试题类型： 编程题 ▸

试题回答参考思路：

堆栈溢出之所以会被消除，是因为事件循环操纵了递归，而不是调用堆栈。当nextlistItem运行时，如果item不为空， timeout函数（ nextlistItem）就会被推到事件队列，该函数退出，因此就清空调用堆栈。当事件队列运行其 timeout事件且进行到下个iem时，把定时器设置为再次调用 nextlistItem。因此，该方法从头到尾都没有直接的递归调用，所以无论迭代次数的多少，调用堆栈一直保持清空的状态

面试题 25. 下面JavaScript的代码将输出什么？闭包在这里能起什么作用?

```
for (var i =0； i < 5:1++){
(function()
{settimeout (function () {
console. log（i）; 1，i *1000 );
})();
}
```

推荐指数：★★★ 试题难度： 中级 试题类型： 编程题 ▸

试题回答参考思路：

上面的代码不会按预期显示值0、1、2、3和4，而是会显示5、5、5、5和5。
原因是，在循环中执行的每个函数将先整个循环完成之后执行，因此，将会引用存储在i中的最后一个值，那就是5。
闭包可以为每次迭代创建一个唯一的作用域，存储作用域内的循环变量。如下代码会按预期输出0、1、2、3和4到控制台。

```
for (var i= 0； i< 5:i++){
( function(x)
{
setTimeout（function(){console. log(x);), x*1000);
})(i);
}
```

面试题 26. 以下JavaScript代码行将输出什么到控制台?

```
console.log("0 ||1 = "+(0 ||1));
console.log("1 ||2 = "+(1 || 2));
console.log("0 && 1 = "+(0 && 1));
console.1og("1 && 2 = "+(1 && 2));
```

推荐指数：★★★★ 试题难度： 高难 试题类型： 编程题 ▸

试题回答参考思路：

该代码将输出以下结果。
0 || 1=1
1 || 2=1
0 & & 1=0
1 & & 2=2
详细分析如下。
在JavaScript中，|| 和 && 都是逻辑运算符，从左至右进行运算。

“或”（||）运算符。在形如X || Y的表达式中，首先，计算X并将其表示为一个布尔值。如果这个布尔值是true，那么返回true（1），不再计算Y，因为“或”的条件已经满足。如果这个布尔值为false，那么，我们仍然无法知道X || Y是真还是假，直到计算Y，并且也把它表示为一个布尔值。

因此，0 ||1的计算结果为true（1），同理计算1 || 2。

“与”（&&）运算符。在形如X&&Y的表达式中，首先，计算X并将其表示为一个布尔值。如果这个布尔值为false，那么返回false（0），不再计算Y，因为“与”的条件已经失败。如果这个布尔值为true，但是，我们仍然不知道X&&Y是真还是假，直到计算Y，并且也把它表示为一个布尔值。

不过，关于&&运算符最有趣的地方在于，当一个表达式的计算结果为“true”的时候，就返回表达式本身。这就解释了为什么1&&2返回2（而不是返回true或1）。

面试题 27. 下面的JavaScript代码将输出什么？请解释

```
console.log（false ==0'）
console.log（false ===10'）
```

推荐指数：★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

代码将输出以下结果：
true
false
在JavaScript中，有两种等于运算符。3个等于运算符===的作用类似于传统的等于运算符：如果两侧的表达式有相同的类型和相同的值，那么计算结果为true。而双等于运算符会只强制比较它们的值。因此，总体上而言，使用===而不是==的做法更好。!==与!=亦是同理。

面试题 28. 解释如下JavaScript代码将输出什么？解释你的答案

```
var a=()，b=(key：'b')，c=(key：'c')；
a[b]=123；
a[c]=456；
console.log(a[b])；
```

推荐指数：★★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

这段代码将输出456（而不是123）。

原因是，当设置对象属性时，JavaScript会隐式地将[ ]内的变量转换成字符串。在这种情况下，由于b和c都是对象，因此，它们都将被转换为"[object Object]"。结果就是， a[b]和a[c]均相当于a["[object Object]"]，并可以互换使用。因此，设置或引用a[c]和设置或引用a[b]完全相同

面试题 29. JavaScript 执行这段代码，输出什么结果 ？

```
function test() {
console.log(a);
console.log(foo());
var a = 1;
function foo() {
return 2;
}
}
test();
```

推荐指数： ★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

这段代码的结果是 undefined 和 2。

原因是，变量和函数的声明都被提前了（移到了函数的顶部），但变量不分配任何值。因此，在打印变量的时候，它在函数中存在（它被声明了），但它仍然是undefined

面试题 30. 考核this在JavaScript中如何工作的?

下面的代码会输出什么结果? 给出你的答案。

```
var fullname = 'John Doe';
var obj = {
fullname: 'Colin Ihrig',
prop: {
fullname: 'Aurelio De Rosa',
getFullname: function() {
return this.fullname;
}
}
};
console.log(obj.prop.getFullname());
var test = obj.prop.getFullname;
console.log(test());
```

推荐指数： ★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

答案是Aurelio De Rosa和John Doe。原因是，在一个函数中，this的行为，取决于JavaScript函数的调用方式和定义方式，而不仅仅是看它如何被定义的。

在第一个 console.log()调用中，getFullname()被调用作为obj.prop对象的函数。所以，上下文指的是后者，函数返回该对象的 fullname。与此相反，当getFullname()被分配到test变量时，上下文指的是全局对象（window）。这是因为test是被隐式设置为全局对象的属性。出于这个原因，该函数返回window的fullname，即定义在第一行的那个值

面试题 31. 简述0.1 + 0.2 === 0.3 嘛？为什么？

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

JavaScirpt 使用 Number 类型来表示数字（整数或浮点数），遵循 IEEE 754 标准，通过 64 位来表示一
个数字（1 + 11 + 52）
1 符号位，0 表示正数，1 表示负数 s
11 指数位（e）
52 尾数，小数部分（即有效数字）
最大安全数字：Number.MAX_SAFE_INTEGER = Math.pow(2, 53) – 1，转换成整数就是 16 位，所以
0.1 === 0.1，是因为通过 toPrecision(16) 去有效位之后，两者是相等的。
在两数相加时，会先转换成二进制，0.1 和 0.2 转换成二进制的时候尾数会发生无限循环，然后进行对阶
运算，JS 引擎对二进制进行截断，所以造成精度丢失。
所以总结：精度丢失可能出现在进制转换和对阶运算中

面试题 32. 实现Javascript函数能够深度克隆基本类型？

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

浅克隆：

```
function shallowClone(obj) {
let cloneObj = {};
for (let i in obj) {
cloneObj[i] = obj[i];
}
return cloneObj;
}
```

深克隆：
考虑基础类型
引用类型
RegExp、Date、函数 不是 JSON 安全的

会丢失 constructor，所有的构造函数都指向 Object
破解循环引用

```
function deepCopy(obj) {
if (typeof obj === 'object') {
var result = obj.constructor === Array ? [] : {};
for (var i in obj) {
result[i] = typeof obj[i] === 'object' ? deepCopy(obj[i]) : obj[i];
}
} else {
var result = obj;
}
return result;
}
```

面试题 33. 简述Js事件是如何实现的？？

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

基于发布订阅模式，就是在浏览器加载的时候会读取事件相关的代码，但是只有实际等到具体的事件触发的时候才会执行。
比如点击按钮，这是个事件（Event），而负责处理事件的代码段通常被称为事件处理程序（EventHandler），也就是「启动对话框的显示」这个动作。
在 Web 端，我们常见的就是 DOM 事件：
DOM0 级事件，直接在 html 元素上绑定 on-event，比如 onclick，取消的话，dom.onclick =null，同一个事件只能有一个处理程序，后面的会覆盖前面的。
DOM2 级事件，通过 addEventListener 注册事件，通过 removeEventListener 来删除事件，一个事件可以有多个事件处理程序，按顺序执行，捕获事件和冒泡事件DOM3级事件，增加了事件类型，比如 UI 事件，焦点事件，鼠标事件

面试题 34. 简述Javascript数组扁平化？

推荐指数：★★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
function flatten(arr) {
let result = [];
for (let i = 0; i < arr.length; i++) {
if (Array.isArray(arr[i])) {
result = result.concat(flatten(arr[i]));
} else {
result = result.concat(arr[i]);
}
}
return result;
```

```
}
const a = [1, [2, [3, 4]]];
console.log(flatten(a));
```

面试题 35. 简述Javascript实现柯里化？

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

预先设置一些参数
柯里化是什么：是指这样一个函数，它接收函数 A，并且能返回一个新的函数，这个新的函数能够处理
函数 A 的剩余参数

```
function createCurry(func, args) {
var argity = func.length;
var args = args || [];
return function () {
var _args = [].slice.apply(arguments);
args.push(..._args);
if (args.length < argity) {
return createCurry.call(this, func, args);
}
return func.apply(this, args);
}
}
```

面试题 36. 简述JS前序遍历判断回文链表？

推荐指数：★★★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

利用链表的后续遍历，使用函数调用栈作为后序遍历栈，来判断是否回文

```
/**
*
*/
var isPalindrome = function(head) {
let left = head;
function traverse(right) {
if (right == null) return true;
let res = traverse(right.next);
res = res && (right.val === left.val);
left = left.next;
return res;
}
return traverse(head);
```

```
};
```
通过 快、慢指针找链表中点，然后反转链表，比较两个链表两侧是否相等，来判断是否是回文链表

```
/**
*
*/
var isPalindrome = function(head) {
// 反转 slower 链表
let right = reverse(findCenter(head));
let left = head;
// 开始比较
while (right != null) {
if (left.val !== right.val) {
return false;
}
left = left.next;
right = right.next;
}
return true;
}
function findCenter(head) {
let slower = head, faster = head;
while (faster && faster.next != null) {
slower = slower.next;
faster = faster.next.next;
}
// 如果 faster 不等于 null，说明是奇数个，slower 再移动一格
if (faster != null) {
slower = slower.next;
}
return slower;
}
function reverse(head) {
let prev = null, cur = head, nxt = head;
while (cur != null) {
nxt = cur.next;
cur.next = prev;
prev = cur;
cur = nxt;
}
return prev;
}
```

面试题 37. 简述实现Javascript反转链表 ？

推荐指数： ★★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} head
* @return {ListNode}
*/
var reverseList = function(head) {
if (head == null || head.next == null) return head;
let last = reverseList(head.next);
head.next.next = head;
head.next = null;
return last;
};
```

面试题 38. 简述实现Javascript合并K个升序链表？

推荐指数：★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode[]} lists
* @return {ListNode}
*/
var mergeKLists = function(lists) {
if (lists.length === 0) return null;
return mergeArr(lists);
};
function mergeArr(lists) {
if (lists.length <= 1) return lists[0];
let index = Math.floor(lists.length / 2);
const left = mergeArr(lists.slice(0, index))
const right = mergeArr(lists.slice(index));
```

```
return merge(left, right);
}
function merge(l1, l2) {
if (l1 == null && l2 == null) return null;
if (l1 != null && l2 == null) return l1;
if (l1 == null && l2 != null) return l2;
let newHead = null, head = null;
while (l1 != null && l2 != null) {
if (l1.val < l2.val) {
if (!head) {
newHead = l1;
head = l1;
} else {
newHead.next = l1;
newHead = newHead.next;
}
l1 = l1.next;
} else {
if (!head) {
newHead = l2;
head = l2;
} else {
newHead.next = l2;
newHead = newHead.next;
}
l2 = l2.next;
}
}
newHead.next = l1 ? l1 : l2;
return head;
}
```

面试题 39. 简述实现Javascript K 个一组翻转链表 ？

推荐指数： ★★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} head
```

```
* @param {number} k
* @return {ListNode}
*/
var reverseKGroup = function(head, k) {
let a = head, b = head;
for (let i = 0; i < k; i++) {
if (b == null) return head;
b = b.next;
}
const newHead = reverse(a, b);
a.next = reverseKGroup(b, k);
return newHead;
};
function reverse(a, b) {
let prev = null, cur = a, nxt = a;
while (cur != b) {
nxt = cur.next;
cur.next = prev;
prev = cur;
cur = nxt;
}
return prev;
}
```

面试题 40. 简述实现Javascript 环形链表 ？

推荐指数： ★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} head
* @return {boolean}
*/
var hasCycle = function(head) {
if (head == null || head.next == null) return false;
let slower = head, faster = head;
while (faster != null && faster.next != null) {
slower = slower.next;
faster = faster.next.next;
```

```
if (slower === faster) return true;
}
return false;
};
```

面试题 41. 简述实现Javascript 排序链表 ?

推荐指数: ★★★★★ 试题难度: 高难 试题类型: 编程题

试题回答参考思路:

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} head
* @return {ListNode}
*/
var sortList = function(head) {
if (head == null) return null;
let newHead = head;
return mergeSort(head);
};
function mergeSort(head) {
if (head.next != null) {
let slower = getCenter(head);
let nxt = slower.next;
slower.next = null;
console.log(head, slower, nxt);
const left = mergeSort(head);
const right = mergeSort(nxt);
head = merge(left, right);
}
return head;
}
function merge(left, right) {
let newHead = null, head = null;
while (left != null && right != null) {
if (left.val < right.val) {
if (!head) {
newHead = left;
head = left;
} else {
```

```
newHead.next = left;
newHead = newHead.next;
}
left = left.next;
} else {
if (!head) {
newHead = right;
head = right;
} else {
newHead.next = right;
newHead = newHead.next;
}
right = right.next;
}
}
newHead.next = left ? left : right;
return head;
}
function getCenter(head) {
let slower = head, faster = head.next;
while (faster != null && faster.next != null) {
slower = slower.next;
faster = faster.next.next;
}
return slower;
}
```

面试题 42. 简述实现Javascript 相交链表 ？

推荐指数：★★★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for singly-linked list.
* function ListNode(val) {
* this.val = val;
* this.next = null;
* }
*/
/**
* @param {ListNode} headA
* @param {ListNode} headB
* @return {ListNode}
*/
var getIntersectionNode = function(headA, headB) {
let lastHeadA = null;
```

```
let lastHeadB = null;
let originHeadA = headA;
let originHeadB = headB;
if (!headA || !headB) {
return null;
}
while (true) {
if (headB == headA) {
return headB;
}
if (headA && headA.next == null) {
lastHeadA = headA;
headA = originHeadB;
} else {
headA = headA.next;
}
if (headB && headB.next == null) {
lastHeadB = headB
headB = originHeadA;
} else {
headB = headB.next;
}
if (lastHeadA && lastHeadB && lastHeadA != lastHeadB) {
return null;
}
}
return null;
};
```

面试题 43. 实现Javascript 最长回文子串【双指针】 ？

推荐指数： ★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {string} s
* @return {string}
*/
var longestPalindrome = function(s) {
if (s.length === 1) return s;
let maxRes = 0, maxStr = '';
for (let i = 0; i < s.length; i++) {
let str1 = palindrome(s, i, i);
let str2 = palindrome(s, i, i + 1);
if (str1.length > maxRes) {
maxStr = str1;
```

```
maxRes = str1.length;
}
if (str2.length > maxRes) {
maxStr = str2;
maxRes = str2.length;
}
}
return maxStr;
};
function palindrome(s, l, r) {
while (l >= 0 && r < s.length && s[l] === s[r]) {
l--;
r++;
}
return s.slice(l + 1, r);
}
```

面试题 44. 实现Javascript 最长公共前缀【双指针】？

推荐指数：★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* @param {string[]} strs
* @return {string}
*/
var longestCommonPrefix = function(strs) {
if (strs.length === 0) return "";
let first = strs[0];
if (first === "") return "";
let minLen = Number.MAX_SAFE_INTEGER;
for (let i = 1; i < strs.length; i++) {
const len = twoStrLongestCommonPrefix(first, strs[i]);
minLen = Math.min(len, minLen);
}
return first.slice(0, minLen);
};
function twoStrLongestCommonPrefix (s, t) {
let i = 0, j = 0;
let cnt = 0;
while (i < s.length && j < t.length) {
console.log(s[i], t[j], cnt)
if (s[i] === t[j]) {
cnt++;
} else {
return cnt;
```

```
}
i++;
j++;
}
```

面试题 45. 实现Javascript 无重复字符的最长子串【双指针】？

推荐指数：★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* @param {string} s
* @return {number}
*/
var lengthOfLongestSubstring = function(s) {
let window = {};
let left = 0, right = 0;
let maxLen = 0, maxStr = '';
while (right < s.length) {
let c = s[right];
right++;
if (window[c]) window[c]++;
else window[c] = 1
while (window[c] > 1) {
let d = s[left];
left++;
window[d]--;
}
if (maxLen < right - left) {
maxLen = right - left;
}
}
return maxLen;
};
```

面试题 46. 实现Javascript 最小覆盖子串【滑动窗口】？

推荐指数：★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* @param {string} s
* @param {string} t
* @return {string}
*/
```

```
var minWindow = function(s, t) {
let need = {}, window = {};
for (let c of t) {
if (!need[c]) need[c] = 1;
else need[c]++;
}
let left = 0, right = 0;
let valid = 0, len = Object.keys(need).length;
let minLen = s.length + 1, minStr = '';
while (right < s.length) {
const d = s[right];
right++;
if (!window[d]) window[d] = 1;
else window[d]++;
if (need[d] && need[d] === window[d]) {
valid++;
}
console.log('left – right', left, right);
while (valid === len) {
if (right – left < minLen) {
minLen = right – left;
minStr = s.slice(left, right);
}
console.lo
let c = s[left];
left++;
window[c]––;
if (need[c] && window[c] < need[c]) {
valid––;
}
}
}
return minStr;
};
```

面试题 47. 实现Javascript 俄罗斯套娃信封问题【排序+最长上升子序列】 ？

推荐指数： ★★★★ 试题难度： 高难 试题类型： 编程题 ▸

试题回答参考思路：

```
/**
* @param {number[][]} envelopes
* @return {number}
*/
var maxEnvelopes = function(envelopes) {
if (envelopes.length === 1) return 1;
```

```
envelopes.sort((a, b) => {
if (a[0] !== b[0]) return a[0] - b[0];
else return b[1] - a[1];
});
let LISArr = [];
for (let [key, value] of envelopes) {
LISArr.push(value);
}
console.log( LISArr);
return LIS(LISArr);
};
function LIS(arr) {
let dp = [];
let maxAns = 0;
for (let i = 0; i < arr.length; i++) {
dp[i] = 1;
}
for (let i = 1; i < arr.length; i++) {
for (let j = i; j >= 0; j--) {
if (arr[i] > arr[j]) {
dp[i] = Math.max(dp[i], dp[j] + 1)
}
maxAns = Math.max(maxAns, dp[i]);
}
}
return maxAns;
}
```

面试题 48. 实现Javascript 最长连续递增序列【快慢指针】 ？

推荐指数：★★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @return {number}
*/
var longestConsecutive = function(nums) {
if (nums.length === 0) return 0;
const set = new Set(nums);
const n = nums.length;
let globalLongest = 1;
for (let i = 0; i < n; i++) {
if (!set.has(nums[i] - 1)) {
let longest = 1;
let currentNum = nums[i];
```

```
while (set.has(currentNum + 1)) {
currentNum += 1;
longest++;
}
globalLongest = Math.max(globalLongest, longest);
}
}
return globalLongest;
};
```

面试题 49. 简述Javascript实现盛最多水的容器【哈希表】算法？

推荐指数：★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* @param {number[]} height
* @return {number}
*/
var maxArea = function(height) {
let n = height.length;
let left = 0, right = n - 1;
let maxOpacity = 0;
while (left < right) {
let res = Math.min(height[left], height[right]) * (right - left);
maxOpacity = Math.max(maxOpacity, res);
if (height[left] < height[right]) left++
else right--;
}
return maxOpacity;
};
```

面试题 50. JS寻找两个正序数组的中位数【双指针】？

推荐指数：★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var findMedianSortedArrays = function(nums1, nums2) {
let m = nums1.length, n = nums2.length;
let i = 0, j = 0;
```

```
let newArr = [];
while (i < m && j < n) {
if (nums1[i] < nums2[j]) {
newArr.push(nums1[i++]);
} else {
newArr.push(nums2[j++]);
}
}
newArr = newArr.concat(i < m ? nums1.slice(i) : nums2.slice(j));
const len = newArr.length;
console.log(newArr)
if (len % 2 === 0) {
return (newArr[len / 2] + newArr[len / 2 - 1]) / 2;
} else {
return newArr[Math.floor(len / 2)];
}
};
```

面试题 51. 实现Javascript 删除有序数组中的重复项【快慢指针】 ？

推荐指数： ★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @return {number}
*/
var removeDuplicates = function(nums) {
if (nums.length <= 1) return nums.length;
let lo = 0, hi = 0;
while (hi < nums.length) {
while (nums[lo] === nums[hi] && hi < nums.length) hi++;
if (nums[lo] !== nums[hi] && hi < nums.length) {
lo++;
nums[lo] = nums[hi];
}
hi++;
}
return lo + 1;
};
```

面试题 52. 简述Javascript 实现和为K的子数组【哈希表】 ？

推荐指数： ★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var subarraySum = function(nums, k) {
let cnt = 0;
let sum0_i = 0, sum0_j = 0;
let map = new Map();
map.set(0, 1);
for (let i = 0; i <= nums.length; i++) {
sum0_i += nums[i];
sum0_j = sum0_i - k;
console.log('map', sum0_j, map.get(sum0_j))
if (map.has(sum0_j)) {
cnt += map.get(sum0_j);
}
let sumCnt = map.get(sum0_i) || 0;
map.set(sum0_i, sumCnt + 1);
}
return cnt;
};
```

面试题 53. 编写Javascript实现 nSum问题【哈希表】 ？

推荐指数： ★★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @param {number} target
* @return {number[]}
*/
var twoSum = function(nums, target) {
let map2 = new Map();
for (let i = 0; i < nums.length; i++) {
map2.set(nums[i], i);
}
for (let i = 0; i < nums.length; i++) {
if (map2.has(target - nums[i]) && map2.get(target - nums[i]) !== i) return
[i, map2.get(target - nums[i])]
}
};
```

面试题 54. 【面试真题】接雨水【暴力+备忘录优化】 ？

推荐指数：★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} height
* @return {number}
*/
var trap = function(height) {
let l_max = [], r_max = [];
let len = height.length;
let maxCapacity = 0;
for (let i = 0; i < len; i++) {
l_max[i] = height[i];
r_max[i] = height[i];
}
for (let i = 1; i < len; i++) {
l_max[i] = Math.max(l_max[i - 1], height[i]);
}
for (let j = len - 2; j >= 0; j--) {
r_max[j] = Math.max(r_max[j + 1], height[j]);
}
for (let i = 0; i < len; i++) {
maxCapacity += Math.min(l_max[i], r_max[i]) - height[i];
}
return maxCapacity;
};
```

面试题 55. 计算二叉树的最近公共祖先？

推荐指数：★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {TreeNode} p
* @param {TreeNode} q
* @return {TreeNode}
*/
```

```
let visited;let parent;
var lowestCommonAncestor = function(root, p, q) {
visited = new Set();
parent = new Map();
dfs(root);
while (p != null) {
visited.add(p.val);
p = parent.get(p.val);
}
while (q != null) {
if (visited.has(q.val)) {
return q;
}
q = parent.get(q.val);
}
return null;
};
function dfs(root) {
if (root.left != null) {
parent.set(root.left.val, root);
dfs(root.left);
}
if (root.right != null) {
parent.set(root.right.val, root);
dfs(root.right);
}
}
```

面试题 56. 编写JS函数实现二叉搜索树中的搜索？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {number} val
* @return {TreeNode}
*/
var searchBST = function(root, val) {
```

```
if (root == null) return null;
if (root.val === val) return root;
if (root.val > val) {
return searchBST(root.left, val);
} else if (root.val < val) {
return searchBST(root.right, val);
}
};
```

面试题 57. 编写JS函数实现删除二叉搜索树中的节点？

推荐指数：★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @param {number} key
* @return {TreeNode}
*/
var deleteNode = function(root, key) {
if (root == null) return null;
if (root.val === key) {
if (root.left == null && root.right == null) return null;
if (root.left == null) return root.right;
if (root.right == null) return root.left;
if (root.left != null && root.right != null) {
let target = getMinTreeMaxNode(root.left);
root.val = target.val;
root.left = deleteNode(root.left, target.val);
}
}
if (root.val < key) {
root.right = deleteNode(root.right, key);
} else if (root.val > key) {
root.left = deleteNode(root.left, key);
}
return root;
};
function getMinTreeMaxNode(root) {
```

```
if (root.right == null) return root;
return getMinTreeMaxNode(root.right);
}
```

面试题 58. 计算完全二叉树的节点个数？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @return {number}
*/
var countNodes = function(root) {
if (root == null) return 0;
let l = root, r = root;
let lh = 0, rh = 0;
while (l != null) {
l = l.left;
lh++;
}
while (r != null) {
r = r.right;
rh++;
}
if (lh === rh) {
return Math.pow(2, lh) - 1;
}
return 1 + countNodes(root.left) + countNodes(root.right);
};
```

面试题 59. 编写Js代码实现二叉树的锯齿形层序遍历？

推荐指数：★★★★ 试题难度：高难 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
```

```
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @return {number[][]}
*/
let res;
var zigzagLevelOrder = function(root) {
if (root == null) return [];
res = [];
BFS(root, true);
return res;
};
function BFS(root, inOrder) {
let arr = [];
let resItem = [];
let node;
let stack1 = new Stack();
let stack2 = new Stack();
// 判断交换时机
let flag;
stack1.push(root);
res.push([root.val]);
inOrder = !inOrder;
while (!stack1.isEmpty() || !stack2.isEmpty()) {
if (stack1.isEmpty()) {
flag = 'stack1';
} else if (stack2.isEmpty()) {
flag = 'stack2';
}
// 决定取那个栈里面的元素
if (flag === 'stack2' && !stack1.isEmpty()) node = stack1.pop();
else if (flag === 'stack1' && !stack2.isEmpty()) node = stack2.pop();
if (inOrder) {
if (node.left) {
if (flag === 'stack1') {
stack1.push(node.left);
} else {
stack2.push(node.left);
}
resItem.push(node.left.val);
}
if (node.right) {
if (flag === 'stack1') {
```

```
stack1.push(node.right);
} else {
stack2.push(node.right);
}
resItem.push(node.right.val);
}
} else {
if (node.right) {
if (flag === 'stack1') {
stack1.push(node.right);
} else {
stack2.push(node.right);
}
resItem.push(node.right.val);
}
if (node.left) {
if (flag === 'stack1') {
stack1.push(node.left);
} else {
stack2.push(node.left);
}
resItem.push(node.left.val);
}
}
// 判断下次翻转的时机
if ((flag === 'stack2' && stack1.isEmpty()) || (flag === 'stack1' &&
stack2.isEmpty())) {
inOrder = !inOrder;
// 需要翻转了，就加一轮值
if (resItem.length > 0) {
res.push(resItem);
}
resItem = [];
}
}
}
class Stack {
constructor() {
this.count = 0;
this.items = [];
}
push(element) {
this.items[this.count] = element;
this.count++;
}
pop() {
if (this.isEmpty()) return undefined;
```

```
const element = this.items[this.count - 1];
delete this.items[this.count - 1];
this.count--;
return element;
}
size() {
return this.count;
}
isEmpty() {
return this.size() === 0;
}
}
```

面试题 60. 简述使用Js代码实现二分查找？

推荐指数： ★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var findMedianSortedArrays = function(nums1, nums2) {
let m = nums1.length, n = nums2.length;
let i = 0, j = 0;
let newArr = [];
while (i < m && j < n) {
if (nums1[i] < nums2[j]) {
newArr.push(nums1[i++]);
} else {
newArr.push(nums2[j++]);
}
}
newArr = newArr.concat(i < m ? nums1.slice(i) : nums2.slice(j));
const len = newArr.length;
console.log(newArr)
if (len % 2 === 0) {
return (newArr[len / 2] + newArr[len / 2 - 1]) / 2;
} else {
return newArr[Math.floor(len / 2)];
}
};
```

面试题 61. 编写Javascript代码实现判断子序列【二分查找】？

推荐指数： ★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {string} s
* @param {string} t
* @return {boolean}
*/
var isSubsequence = function(s, t) {
let hash = {};
for (let i = 0; i < t.length; i++) {
if (!hash[t[i]]) hash[t[i]] = [];
hash[t[i]].push(i);
}
let lastMaxIndex = 0;
for (let i = 0; i < s.length; i++) {
if (hash[s[i]]) {
const index = binarySearch(hash[s[i]], lastMaxIndex);
console.log('index', index, hash[s[i]]);
if (index === -1) return false;
lastMaxIndex = hash[s[i]][index] + 1;
} else return false;
}
return true;
};
function binarySearch(array, targetIndex) {
let left = 0, right = array.length;
while (left < right) {
let mid = left + Math.floor((right - left) / 2);
if (array[mid] >= targetIndex) {
right = mid;
} else if (array[mid] < targetIndex) {
left = mid + 1;
}
}
if (left >= array.length || array[left] < targetIndex) return -1;
return left;
}
```

面试题 62. 简述JS 在排序数组中查找元素的第一个和最后一个位置【二分搜索】 ？

推荐指数： ★★★ 试题难度： 高难 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @param {number} target
* @return {number[]}
*/
var searchRange = function(nums, target) {
const left = leftBound(nums, target);
const right = rightBound(nums, target);
return [left, right];
};
function leftBound(nums, target) {
let left = 0;
let right = nums.length - 1;
while (left <= right) {
let mid = Math.floor(left + (right - left) / 2);
if (nums[mid] === target) {
right = mid - 1;
} else if (nums[mid] < target) {
left = mid + 1;
} else if (nums[mid] > target) {
right = mid - 1;
}
}
if (left >= nums.length || nums[left] !== target) {
return -1;
}
return left;
}
function rightBound(nums, target) {
let left = 0;
let right = nums.length - 1;
while (left <= right) {
let mid = Math.floor(left + (right - left) / 2);
if (nums[mid] === target) {
left = mid + 1;
} else if (nums[mid] < target) {
left = mid + 1;
} else if (nums[mid] > target) {
right = mid - 1;
}
}
if (right < 0 || nums[right] !== target) {
return -1;
}
return right;
}
```

面试题 63. 简述JS实现最长递增子序列？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @return {number}
*/
var lengthOfLIS = function(nums) {
let maxLen = 0, n = nums.length;
let dp = [];
for (let i = 0; i < n; i++) {
dp[i] = 1;
}
for (let i = 0; i < n; i++) {
for (let j = 0; j < i; j++) {
if (nums[i] > nums[j]) {
dp[i] = Math.max(dp[i], dp[j] + 1);
}
}
maxLen = Math.max(maxLen, dp[i]);
}
return maxLen;
};
```

面试题 64. 【Javascript面试真题】 零钱兑换？

推荐指数：★★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* @param {number[]} coins
* @param {number} amount
* @return {number}
*/
var coinChange = function(coins, amount) {
if (amount === 0) return 0;
let dp = [];
for (let i = 0; i <= amount; i++) {
dp[i] = amount + 1;
}
dp[0] = 0;
for (let i = 0; i <= amount; i++) {
for (let j = 0; j < coins.length; j++) {
if (i >= coins[j]) {
```

```
dp[i] = Math.min(dp[i - coins[j]] + 1, dp[i])
}
}
}
return dp[amount] === amount + 1 ? -1 : dp[amount];
};
```

面试题 65. 【Javascript面试真题】 最长公共子序列？

推荐指数： ★★★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {string} text1
* @param {string} text2
* @return {number}
*/
var longestCommonSubsequence = function(text1, text2) {
let n1 = text1.length, n2 = text2.length;
let dp = [];
for (let i = -1; i < n1; i++) {
dp[i] = [];
for (let j = -1; j < n2;j++) {
dp[i][j] = 0;
}
}
for (let i = 0; i < n1; i++) {
for (let j = 0; j < n2; j++) {
if (text1[i] === text2[j]) {
dp[i][j] = Math.max(dp[i][j], dp[i - 1][j - 1] + 1);
} else {
dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1])
}
}
}
return dp[n1 - 1][n2 - 1];
};
```

面试题 66. 【Javascript面试真题】最长回文子序列？

推荐指数： ★★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
/**
* @param {string} s
```

```
* @return {number}
*/
var longestPalindromeSubseq = function(s) {
let dp = [];
for (let i = 0; i < s.length; i++) {
dp[i] = [];
for (let j = 0; j < s.length; j++) {
dp[i][j] = 0;
}
dp[i][i] = 1;
}
for (let i = s.length - 1; i >= 0; i--) {
for (let j = i + 1; j < s.length; j++) {
if (s[i] === s[j]) {
dp[i][j] = dp[i + 1][j - 1] + 2;
} else {
dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);
}
}
}
return dp[0][s.length - 1];
};
```

面试题 67. 【Javascript面试真题】最大子序和？

推荐指数：★★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* @param {number[]} nums
* @return {number}
*/
var maxSubArray = function(nums) {
let maxSum = -Infinity;
let dp = [], n = nums.length;
for (let i = -1; i < n; i++) {
dp[i] = 0;
}
for (let i = 0; i < n; i++) {
dp[i] = Math.max(nums[i], dp[i - 1] + nums[i]);
maxSum = Math.max(maxSum, dp[i]);
}
```

面试题 68. 简述使用JS实现二叉树的最小深度？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
/**
* Definition for a binary tree node.
* function TreeNode(val) {
* this.val = val;
* this.left = this.right = null;
* }
*/
/**
* @param {TreeNode} root
* @return {number}
*/
var minDepth = function(root) {
if (root == null) return 0;
let depth = 1;
let queue = new Queue();
queue.push(root);
while (!queue.isEmpty()) {
let size = queue.size();
for (let i = 0; i < size; i++) {
const node = queue.pop();
if (node.left == null && node.right == null) return depth;
if (node.left) {
queue.push(node.left);
}
if (node.right) {
queue.push(node.right);
}
}
depth++;
}
return depth;
};
class Queue {
constructor() {
this.items = [];
this.count = 0;
this.lowerCount = 0;
}
push(elem) {
this.items[this.count++] = elem;
}
pop() {
if (this.isEmpty()) {
return;
}
```

```
const elem = this.items[this.lowerCount];
delete this.items[this.lowerCount];
this.lowerCount++;
return elem;
}
isEmpty() {
if (this.size() === 0) return true;
return false;
}
size() {
return this.count - this.lowerCount;
}
}
```

面试题 69. 简述实现JS两个数组合并成一个数组？

请把两个数组 ['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1', 'D2'] 和 ['A', 'B', 'C', 'D']，合并为 ['A1', 'A2', 'A', 'B1', 'B2', 'B', 'C1', 'C2', 'C', 'D1', 'D2',

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
function concatArr (arr1, arr2) { const arr = [...arr1];
let currIndex = 0;
for (let i = 0; i < arr2.length; i++) {
const RE = new RegExp(arr2[i])
while(currIndex < arr.length) {
++currIndex
if (!RE.test(arr[currIndex])) {
arr.splice(currIndex, 0, a2[i])
break;
}
}
}
return arr } var a1 = ['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1',
'D2']
var a2 = ['A', 'B', 'C', 'D'] const arr = concatArr(a1, a2)
console.log(a1)
// ['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1', 'D2'] console.log(a2)
// ['A', 'B', 'C', 'D'] console.log(arr)
// ['A1', 'A2', 'A', B1', 'B2', 'B', C1', 'C2', 'C', D1', 'D2', 'D
```

面试题 70. 改造下面的代码，使之输出 0 - 9，写出你能想到的所有解法。？

```
for (var i = 0;
i< 10; i++){
setTimeout(() => {
```

```
console.log(i);
}, 1000)
```

推荐指数：★★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

```
// 解法：
for (let i = 0;
i< 10;
i++){
setTimeout(() => {
console.log(i);
}, 1000)}
// 解法二：for (var i = 0;
i< 10; i++){ ((i) => {
setTimeout(() => {
console.log(i);
}, 1000) })(i)}
```

面试题 71. 简述下面的代码打印什么内容，为什么？

```
var b = 10;
(function b(){
b = 20;
console.log(b);
})();
```

推荐指数：★★★★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

```
ƒ b(){
b = 20;
console.log(b);
}
```

首先函数声明比变量要高，其次 b = 20 没有 var 获取其他，说明是 window 最外层定义的变量。

js 作用域中，先找最近的 那就是 b fn ，直接打印了，如果 b = 20 有 var 那就是打印 20

面试题 72. 简述简单改造下面的代码，使之分别打印 10 和 20?

```
var b = 10;
(function b(){
b = 20;
console.log(b);
})();
```

推荐指数：★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

```
var b = 10;
(function (){
b = 20;
console.log(b);
// 20})();
var b = 10;
(function (){
console.log(b);
// 10 b = 20;
})();
```

面试题 73. 使用迭代的方式实现 flatten 函数？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
var arr=[1,2,3,[4,5],[6,[7,[8]]]]/** * 使用递归的方式处理 * wrap 内保
存结果 ret * 返回一个递归函数 * * @returns */function wrap(){
var ret=[];
return function flat(a){
for(var item of
a){ if(item.constructor===Array){
ret.concat(flat(item))
}else{
ret.push(item)
}
}
return ret
}}console.log(wrap()(arr));
```

面试题 74. 简述（京东）下面代码中 a 在什么情况下会打印 1？

```
var a = ?;
if(a == 1 && a == 2 && a == 3){
console.log(1);
}
```

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
var a = {
i: 1,
toString() {
return a.i++;
```

```
}}if( a == 1 && a == 2 && a == 3 ) {
console.log(1);
}
let a = [1,2,3];
a.toString = a.shift;if( a == 1 && a == 2 && a == 3 ) {
console.log(1);}
```

面试题 75. 简述下面JS代码输出什么？

```
var a = 10;(function () {
console.log(a)
a = 5
console.log(window.a)
var a = 20;
console.log(a)})()
```

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

分别为 undefined 10 20，原因是作用域问题，在内部声名 var a = 20;相当于先声明 var a;然后再执行赋值操作，这是在 I I F E 内形成的独立作用域，如果把 var a=20 注释掉，那么 a 只有在外部有声明，显示的就是外部的 A 变量的值了。结果 A 会是 10 5

面试题 76. 简述如何实现一个 sleep 函数？

比如 sleep(1000) 意味着等待 1000 毫秒，可从 Promise、Generator、Async/Await等角度实现

推荐指数：★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
const sleep = (time) => {
return new Promise(resolve => setTimeout(resolve,time))}sleep(1000).then(() =>
{
代码
```

面试题 77. 简述输出以下代码执行的结果并解释为什么？

```
var obj = {
'2': 3,
'3': 4,
'length': 2,
'splice': Array.prototype.splice,
'push':
Array.prototype.push}obj.push(1)obj.push(2)console.log(obj)
```

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

结果：[,,1,2], length 为 4
伪数组（ArrayLike）

面试题 78. 简述输出以下代码的执行结果并解释为什么？

```
var a = {n: 1};
var b = a;a.x = a = {n: 2};
console.log(a.x)
console.log(b.x)
```

推荐指数：★★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

结果:undefined{n:2}
首先，a 和 b 同时引用了{n:2}对象，接着执行到 a.x = a = {n：2}语句，尽管赋值是从右到左的没错，但是.的优先级比=要高，所以这里首先执行 a.x，相当于为a（或者 b）所指向的{n:1}对象新增了一个属性 x，即此时对象将变为{n:1;x:undefined}。之后按正常情况，从右到左进行赋值，此时执行 a ={n:2}的时候，a 的引用改变，指向了新对象{n：2},而 b 依然指向的是旧对象。之后执行a.x = {n：2}的时候，并不会重新解析一遍 a，而是沿用最初解析 a.x 时候的 a，也即旧对象，故此时旧对象的 x 的值为{n：2}，旧对象为 {n:1;x:{n：2}}，它被 b引用着。后面输出 a.x 的时候，又要解析 a 了，此时的 a 是指向新对象的 a，而这个新对象是没有 x 属性的，故访问时输出 undefined；而访问 b.x 的时候，将输出旧对象的 x

面试题 79. 实现某公司 1 到 12 月份的销售额存在一个对象？

如下：{1:222, 2:123, 5:888}，请把数据处理为如下结构：[222, 123, null, null, 888, null, null, null, null, null, null, null]

推荐指数：★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
let obj = {1:222, 2:123, 5:888};
const result = Array.from({ length: 12 }).map((_, index) => obj[index +1] || null);
console.log(result)
```

面试题 80. 简述要求设计 LazyMan 实现以下功能？

```
LazyMan('Tony');
// Hi I am Tony
LazyMan('Tony').sleep(10).eat('lunch');
// Hi I am Tony
// 等待了 10 秒...
// I am eating
lunchLazyMan('Tony').eat('lunch').sleep(10).eat('dinner');
```

```
// Hi I am Tony
// I am eating lunch// 等待了 10 秒...
// I am eating
dinerLazyMan('Tony').eat('lunch').eat('dinner').sleepFirst(5).sleep(1
0).eat('junk food');
// Hi I am Tony// 等待了 5 秒...
// I am eating lunch
// I am eating dinner
// 等待了 10 秒...
// I am eating junk food
```

推荐指数： ★★ 试题难度： 中级 试题类型： 编程题

试题回答参考思路：

```
class LazyManClass {
constructor(name) {
this.name = name
this.queue = []
console.log(`Hi I am ${name}`)
setTimeout(() => {
this.next()
},0)
}
sleepFirst(time) {
const fn = () => {
setTimeout(() => {
console.log(`等待了${time}秒...`)
this.next()
}, time)
}
this.queue.unshift(fn)
return this
}
sleep(time) {
const fn = () => {
setTimeout(() => {
console.log(`等待了${time}秒...`)
this.next()
},time)
}
this.queue.push(fn)
return this
}
eat(food) {
const fn = () => {
console.log(`I am eating ${food}`)
this.next()
}
```

```
this.queue.push(fn)
return this
}
next() {
const fn = this.queue.shift()
fn && fn() }}function LazyMan(name) {
return new LazyManClass(name)}
```

面试题 81. 简述给定两个数组，写一个方法来计算它们的交集？

例如：给定 nums1 = [1, 2, 2, 1]，nums2 = [2, 2]，返回 [2, 2]。
var nums1 = [1, 2, 2, 1], nums2 = [2, 2, 3, 4];

推荐指数：★★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
[NaN].indexOf(NaN) === -1var newArr1 = nums1.filter(function(item) {
return nums2.indexOf(item) > -1;
});
console.log(newArr1);
// 2.
var newArr2 = nums1.filter((item) => {
return nums2.includes(item);
});
console.log(newArr2);
```

面试题 82. 简述如何设计实现无缝轮播？

简单来说，无缝轮播的核心是制造一个连续的效果。最简单的方法就是复制一个轮播的元素，当复制元素将要滚到目标位置后，把原来的元素进行归位的操作，以达到无缝的轮播效果

推荐指数：★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
useEffect(() => {
const requestAnimationFrame =
window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame
const cancelAnimationFrame =
window.cancelAnimationFrame ||
window.webkitCancelAnimationFrame ||
window.mozCancelAnimationFrame
const scrollNode = noticeContentEl.current
const distance = scrollNode.clientWidth / 2
```

```
scrollNode.style.left = scrollNode.style.left || 0
window.__offset = window.__offset || 0
let requestId = null
const scrollLeft = () => {
const speed = 0.5
window.__offset = window.__offset + speed
scrollNode.style.left = -window.__offset + 'px'
// 关键行：当距离小于偏移量时，重置偏移量
if (distance <= window.__offset) window.__offset = 0
requestId = requestAnimationFrame(scrollLeft)
} requestId = requestAnimationFrame(scrollLeft)
if (pause) cancelAnimationFrame(requestId)
return () => cancelAnimationFrame(requestId)
}, [notice, pause])
```

面试题 83. 简述模拟实现一个 Promise.finally ?

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
Promise.prototype.finally = function (callback) {
let P = this.constructor;
return this.then(
value => P.resolve(callback()).then(() => value),
reason => P.resolve(callback()).then(()
=> { throw reason })
);
};
```

面试题 84. 实现Javascript数组编程题 ?

随机生成一个长度为 10 的整数类型的数组，例如 [2, 10, 3, 4, 5, 11, 10, 11, 20]，将其排列成一个新数组，要求新数组形式如下，例如 [[2, 3, 4, 5], [10, 11], [20]]。

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
function formArray(arr: any[]) {
const sortedArr = Array.from(new Set(arr)).sort((a, b) => a - b);
const map = new Map();
sortedArr.forEach((v) => {
const key = Math.floor(v / 10);
const group = map.get(key) || [];
group.push(v);
map.set(key, group);
}); return [...map.values()];}// 求连续的版本 function
```

```
formArray1(arr: any[]) {
const sortedArr = Array.from(new Set(arr)).sort((a, b) => a - b);
return sortedArr.reduce((acc, cur) => {
const lastArr = acc.slice().pop() || [];
const lastVal = lastArr.slice().pop();
if (lastVal!=null && cur-lastVal === 1)
{
lastArr.push(cur);
} else {
acc.push([cur]);
}
return acc;
}, []);}function genNumArray(num: number, base = 100) {
return Array.from({length: num}, () =>
Math.floor(Math.random()*base));
}const arr = genNumArray(10, 20);
//[2, 10, 3, 4, 5, 11, 10, 11, 20];
const res = formArray(arr);console.log(`res
${JSON.stringify(res)}`);
```

面试题 85. 实现一个字符串匹配算法，从长度为 n 的字符串 S 中，查找是否存在字符串 T，T 的长度是 m，若存在返回所在位置？

推荐指数：★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
const find = (S, T) => {
if (S.length < T.length) return -1
for (let i = 0; i < S.length; i++) {
if (S.slice(i, i + T.length) === T) return i
}
return -1
```

面试题 86. 使用 JavaScript Proxy 实现简单的数据绑定？

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
<div id="app"><br />
<input type="text" id="input"><br />
<div><br />
TODO:<br />
<span id="text"></span><br />
</div> <div id="btn">Add To Todo List</div><br />
<ul id="list"></ul> </div><br />
```

```
const input = document.getElementById('input')<br />
const text = document.getElementById('text')<br />
const list = document.getElementById('list')<br />
const btn = document.getElementById('btn')<br />
let render<br />
const inputObj = new Proxy({}, {<br />
get (target, key, receiver) {<br />
return Reflect.get(target, key, receiver)<br />
},<br />
set (target, key, value, receiver) {<br />
if (key === 'text') {<br />
input.value = value<br />
text.innerHTML = value<br />
}<br />
return Reflect.set(target, key, value,<br />
receiver)<br />
}<br />
})<br />
class Render {<br />
constructor (arr) {<br />
this.arr = arr<br />
}<br />
init () {<br />
const fragment = document.createDocumentFragment()<br />
for (let i = 0; i < this.arr.length; i++) {<br />
const li = document.createElement('li')<br />
li.textContent = this.arr[i]<br />
fragment.appendChild(li)<br />
}<br />
list.appendChild(fragment)<br />
}<br />
addList (val) {<br />
const li = document.createElement('li')<br />
li.textContent = val<br />
list.appendChild(li)<br />
}<br />
}<br />
const todoList = new Proxy([], {<br />
get (target, key, receiver) {<br />
return Reflect.get(target, key, receiver)<br />
},<br />
set (target, key, value, receiver) {<br />
if (key !== 'length')<br />
{<br />
render.addList(value)<br />
}<br />
return Reflect.set(target, key, value,<br />
```

```
receiver)<br />
}<br />
})<br />
window.onload = () => {<br />
render = new Render([])<br />
render.init()<br />
}<br />
input.addEventListener('keyup', e => {<br />
inputObj.text = e.target.value<br />
})<br />
btn.addEventListener('click', () =><br />
{<br />
todoList.push(inputObj.text)<br />
inputObj.text = ''<br />
})<br />
```

面试题 87. 简述输出以下JS代码运行结果？

```
// example 1
var a={}, b='123', c=123;
a[b]='b';a[c]='c';
console.log(a[b]);
----------------------
// example 2var a={}, b=Symbol('123'), c=Symbol('123');
a[b]='b';
a[c]='c';
console.log(a[b]);
----------------------
// example 3var a={}, b={key:'123'}, c={key:'456'};
a[b]='b';a[c]='c';
console.log(a[b]);
```

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

1. 对象的键名只能是字符串和 Symbol 类型。
2. 其他类型的键名会被转换成字符串类型。
3. 对象转字符串默认会调用 toString 方法。

```
// example 1
var a={}, b='123', c=123;a[b]='b';
// c 的键名会被转换成字符串'123'，这里会把 b 覆盖掉。a[c]='c';
// 输出 cconsole.log(a[b]);
// example 2var a={}, b=Symbol('123'), c=Symbol('123');
// b 是 Symbol 类型，不需要转换。a[b]='b';
// c 是 Symbol 类型，不需要转换。任何一个 Symbol 类型的值都是不相等的，
所以不会覆盖掉 b。a[c]='c';
// 输出 bconsole.log(a[b]);
// example 3var a={}, b={key:'123'}, c={key:'456'};
// b 不是字符串也不是 Symbol 类型，需要转换成字符串。
```

```
// 对象类型会调用 toString 方法转换成字符串 [object Object]。a[b]='b';
// c 不是字符串也不是 Symbol 类型，需要转换成字符串。
// 对象类型会调用 toString 方法转换成字符串 [object Object]。这里会把
b 覆盖掉。a[c]='c';
// 输出 cconsole.log(a[b]);
```

面试题 88. 实现JS算法题「旋转数组」？

给定一个数组，将数组中的元素向右移动 k 个位置，其中 k 是非负数。
示例 1:
输入: [1, 2, 3, 4, 5, 6, 7] 和 k = 3 输出: [5, 6, 7, 1, 2, 3, 4] 解释: 向右旋转 1 步: [7, 1, 2, 3, 4, 5, 6] 向右旋转 2 步: [6, 7, 1, 2, 3, 4, 5] 向右旋转 3 步: [5, 6, 7, 1, 2, 3, 4]
示例 2:
输入: [-1, -100, 3, 99] 和 k = 2 输出: [3, 99, -1, -100] 解释: 向右旋转 1 步: [99, -1, -100, 3] 向右旋转 2 步: [3, 99, -1, -100]

推荐指数: ★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

```
function rotate(arr, k) {
const len = arr.length const step = k % len return
arr.slice(-step).concat(arr.slice(0, len - step))}// rotate([1, 2, 3, 4,5, 6], 7) =>
[6, 1, 2, 3, 4, 5]
```

面试题 89. 简述实现打印出 1 - 10000的对称数 ？

例如：121、1331 等

推荐指数: ★★ 试题难度: 初级 试题类型: 编程题

试题回答参考思路:

```
[...Array(10000).keys()].filter((x) => {
return x.toString().length > 1 && x
===Number(x.toString().split('').reverse().join(''))
})
```

面试题 90. 实现JS算法题之「移动零」 ？

给定一个数组 nums，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。
示例:
输入: [0,1,0,3,12] 输出: [1,3,12,0,0]
复制代码说明: 必须在原数组上操作，不能拷贝额外的数组。 尽量减少操作次数

推荐指数: ★★★★ 试题难度: 初级 试题类型: 编程题

试题回答参考思路:

```
function zeroMove(array) {
let len = array.length;
let j = 0;
for(let
i=0;i if(array[i]===0){
array.push(0);
array.splice(i,1);
i --;
j ++;
}
}
return array;
}
```

面试题 91. 简述请实现一个 add 函数，满足以下功能？

```
add(1);
// 1add(1)(2);
// 3add(1)(2)(3);
// 6add(1)(2, 3);
// 6add(1, 2)(3);
// 6add(1, 2, 3);
// 6
```

推荐指数：★★★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

实现 1：

```
function currying(fn, length) {
length = length || fn.length; // 注释 1
return function (...args) { // 注释 2 return
args.length >= length // 注释 3
? fn.apply(this, args) // 注释 4
: currying(fn.bind(this, ...args), length - args.length) // 注释
5 }}
```

实现 2：

```
const currying = fn =>
judge = (...args) =>
args.length >= fn.length
? fn(...args)
: (...arg) => judge(...args, ...arg)
```

其中注释部分

注释 1：第一次调用获取函数 fn 参数的长度，后续调用获取 fn 剩余参数的长度

注释 2：currying 包裹之后返回一个新函数，接收参数为 ...args

注释 3：新函数接收的参数长度是否大于等于 fn 剩余参数需要接收的长度

注释 4：满足要求，执行 fn 函数，传入新函数的参数

注释 5：不满足要求，递归 currying 函数，新的 fn 为 bind 返回的新函数（bind 绑定了 ...args 参数，未执行），新的 length 为 fn 剩余参数的长度

面试题 92. 实现JS算法题之「两数之和」？

给定一个整数数组和一个目标值，找出数组中和为目标值的两个数。
你可以假设每个输入只对应一种答案，且同样的元素不能被重复利用。
示例：
给定 nums = [2, 7, 11, 15], target = 9 因为 nums[0] + nums[1] = 2 + 7 = 9 所以返回 [0, 1]

推荐指数：★★★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
function anwser (arr, target) {
let map = {} for (let i = 0; i < arr.length; i++) {
map[arr[i]] = i }
for (let i = 0; i < arr.length; i++) {
var d = target – arr[i]
if (map[d]) {
return [i, map[d]]
}
}
return new Error('404 not found')
```

面试题 93. 简述在输入框中如何判断输入的是一个正确的网址？

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

```
function isUrl(url) {
const a = document.createElement("a");
a.href = url;
return (
[
/^(http|https):$/.test(a.protocol),
a.host,
a.pathname !== url,
a.pathname !== `/${url}`
].find(x => !x) === undefined
);
}
```

面试题 94. 简述实现 convert 方法，把原始 list 转换成树形结构， 要求尽可能降低时间复杂度？

以下数据结构中，id 代表部门编号，name 是部门名称，parentId 是父部门编号，为 0 代表一级部门，现在要求实现一个 convert 方法，把原始 list 转换成树形结构，parentId 为

多少就挂载在该 id 的属性 children 数组下，结构如下：

```
// 原始 list 如下 let list =[
{id:1,name:'部门 A',parentId:0},
{id:2,name:'部门 B',parentId:0},
{id:3,name:'部门 C',parentId:1},
{id:4,name:'部门 D',parentId:1},
{id:5,name:'部门 E',parentId:2},
{id:6,name:'部门 F',parentId:3},
{id:7,name:'部门 G',parentId:2},
{id:8,name:'部门 H',parentId:4}];
const result = convert(list, ...);// 转换后的结果如下 let result =
[
{
id: 1,
name: '部门 A',
parentId: 0,
children: [
{
id: 3,
name: '部门 C',
parentId: 1,
children: [
{
id: 6,
name: '部门 F',
parentId: 3
}, {
id: 16,
name: '部门 L',
parentId: 3
}
]
},
{
id: 4,
name: '部门 D',
parentId: 1,
children: [
{
id: 8,
name: '部门 H',
parentId: 4
}
]
}
]
}, ···];
```

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

```
function convert(list) {
const res = []
const map = list.reduce((res, v) => (res[v.id] = v, res), {})
for (const item of list) {
```

```
if (item.parentId === 0) {
res.push(item)
Continue
}
if (item.parentId in map) {
const parent = map[item.parentId]
parent.children = parent.children || []
parent.children.push(item)
}
}
return res}
```

面试题 95. 简述设计并实现 Promise.race() ?

推荐指数: ★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

```
Promise._race = promises => new Promise((resolve, reject) => {
promises.forEach(promise => {
promise.then(resolve, reject)
})})Promise.myrace = function(iterator) {
return new Promise ((resolve,reject) => {
try {
let it = iterator[Symbol.iterator]();
while(true) {
let res = it.next();
console.log(res);
if(res.done) break;
if(res.value instanceof Promise){
res.value.then(resolve,reject);
} else{
resolve(res.value)
}
}
} catch (error) {
reject(error)
}
})
}
```

面试题 96. 简述已知数据格式，实现一个函数 fn 找出链条中所有的 父级 idconst value = '112 ?

```
const fn = (value) => {...}fn(value) // 输出 [1,  11,  112]
```

推荐指数: ★★★★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路：

```
const data = [
{
id: "1",
name: "test1",
children: [
{
id: "11",
name: "test11",
children: [
{
id: "111",
name: "test111"
},
{
id: "112",
name: "test112"
}
]
},
{
id: "12",
name: "test12",
children: [
{
id: "121",
name: "test121"
},
{
id: "122",
name:
"test122"
}
]
}
]
}
];
const find = value => {
let result = [];
let findArr = data;
let skey = "";
for (let i = 0, l = value.length; i < l; i++) {
skey += value[i];
let item = findArr.find(item => {
return item.id == skey;
```

```
});
if (!item) {
return [];
}
result.push(item.id);
if (item.children) {
findArr = item.children;
} else {
if (i < l - 1) return [];
return result;
}
}
};
//调用看结果
function testFun() {
console.log("1,11,111:", find("111"));
console.log("1,11,112:", find("112"));
console.log("1,12,121:", find("121"));
console.log("1,12,122:", find("122"));
console.log("[]:", find("113"));
console.log("[]:", find("1114"));
}
```

面试题 97. 通过实现JS函数实现以下逻辑目标？

给定两个大小为 m 和 n 的有序数组 nums1 和
nums2。请找出这两个有序数组的中位数。要求算法的时间复杂
度为 O(log(m+n))。
示例 1:
nums1 = [1, 3] nums2 = [2]
中位数是 2.0
示例 2:
nums1 = [1, 2] nums2 = [3, 4]
中位数是(2 + 3) / 2 = 2.5

推荐指数: ★★ 试题难度: 中级 试题类型: 编程题

试题回答参考思路:

```
const findMedianSortedArrays = function(
nums1: number[],
nums2: number[]
) {
const lenN1 = nums1.length;
const lenN2 = nums2.length;
const median = Math.ceil((lenN1 + lenN2 + 1) / 2);
const isOddLen = (lenN1 + lenN2) % 2 === 0;
const result = new Array(median);
let i = 0; // pointer for nums1
```

```
let j = 0; // pointer for nums2
for (let k = 0; k < median; k++) {
if (i < lenN1 && j < lenN2) {
// tslint:disable-next-line:prefer-conditional-expression
if (nums1[i] < nums2[j]) {
result[i + j] = nums1[i++];
} else {
result[i + j] = nums2[j++];
}
} else if (i < lenN1) {
result[i + j] = nums1[i++];
} else if (j < lenN2) {
result[i + j] = nums2[j++];
}
}
if (isOddLen) {
return (result[median - 1] + result[median - 2]) / 2;
} else {
return result[median - 1];
}
};
```

面试题 98. 简述模拟实现一个深拷贝，并考虑对象相互引用以及 Symbol 拷贝的情况？

推荐指数：★★ 试题难度：中级 试题类型：编程题

试题回答参考思路：

一个不考虑其他数据类型的公共方法，基本满足大部分场景

```
function deepCopy(target, cache = new Set()) {
if (typeof target !== 'object' || cache.has(target)) {
return target
}
if (Array.isArray(target)) {
target.map(t => {
cache.add(t)
return t
})
} else {
return
[...Object.keys(target), ...Object.getOwnPropertySymbols(target)].red
uce((res, key) => {
cache.add(target[key])
res[key] = deepCopy(target[key], cache)
return res
}, target.constructor !== Object ?
Object.create(target.constructor.prototype) : {})
```

```
}
}
```

主要问题是

symbol 作为 key，不会被遍历到，所以 stringify 和 parse 是不行的

有环引用，stringify 和 parse 也会报错

我们另外用 getOwnPropertySymbols 可以获取 symbol key 可以解决问题

1，用集

合记忆曾经遍历过的对象可以解决问题 2。当然，还有很多数据类型要独立去拷贝。比如拷贝一个 RegExp，lodash 是最全的数据类型拷贝了，有空可以研究一下

另外，如果不考虑用 symbol 做 key，还有两种黑科技深拷贝，可以解决环引用的问题，比 stringify 和 parse 优雅强一些

```
function deepCopyByHistory(target) {
const prev = history.state
history.replaceState(target, document.title)
const res = history.state
history.replaceState(prev, document.title)
return res
}
async function deepCopyByMessageChannel(target) {
return new Promise(resolve => {
const channel = new MessageChannel()
channel.port2.onmessage = ev => resolve(ev.data)
channel.port1.postMessage(target)
}).then(data => data)}
```

无论哪种方法，它们都有一个共性：失去了继承关系，所以剩下的需要我们手动补上去了，故有 Object.create(target.constructor.prototype)的操作

面试题 99. 简述写出如下JS代码的打印结果？

```
function changeObjProperty(o) {
o.siteUrl = "http://www.baidu.com"
o = new Object()
o.siteUrl = "http://www.google.com"
}
let webSite = new Object();
changeObjProperty(webSite);
console.log(webSite.siteUrl);
```

推荐指数：★★ 试题难度：初级 试题类型：编程题

试题回答参考思路：

webSite 属于复合数据类型，函数参数中以地址传递，修改值会影响到原始值，但如果将其完全替换成另一个值，则原来的值不会受到影响

面试题 100. 简述通过JS实现编程算法题？

用 JavaScript 写一个函数，输入 int 型，返回整数逆序后的字符串。如：输入整型 1234，返回字符串“4321”。要求必须使用递归函数调用，不能用全局变量，输入函数必须只有一

个参数传入，必须返回字符串

推荐指数： ★★ 试题难度： 初级 试题类型： 编程题

试题回答参考思路：

```
function fun(num) {
let num1 = num / 10;
let num2 = num % 10;
if (num1 < 1) {
return num;
} else {
num1 = Math.floor(num1);
return `${num2}${fun(num1)}`;
}
}
var a = fun(12345);
console.log(a);
console.log(typeof a);
```