# 前端面试指南 — JavaScript 篇面试题

# 说明：javaScript在笔试题中只有一两题，基本都在与面试官沟通的时候考察，这是考察基础的重点，如果回答不好回给人印象基础不牢固， 只会用的印象，所以要牢牢掌握下面的考点。

# 考点 1：数据类型

### 问题：JavaScript 有几种数据类型，分别是什么？

有 7 种：

string 字符串类型number 数字类型object 对象类型boolean 布尔值类型null 空值

undefined 未定义

其中 ES6 还新增了一种：Symbol 类型，用于创建一个独一无二的值。

### 问题：判断数据类型的方式有那些

1. typeof

对于基本类型，除 null 以外，均可以返回正确的结果。

对于引用类型，除 function 以外，一律返回 object 类型。对于 null ，返回 object 类型。

对于 function 返回 function 类型。

1. instanceOf
2. Constructor
3. Object.prototype.toString.call

链接：https://juejin.cn/post/6919805736734162952

### 问题：null 和undefined 有什么区别？

undefined：声明了但还没有赋值；

null：****即该处不应该有值，作为原型链的终点。****主动释放一个变量引用的对象，表示一个变量不再指向任何对象地址。 当使用完一个比较大的对象时，需要对其进行释放内存时，设置为 null。

### 问题：列举三种强制类型转换和两种隐式类型转换？

强制类型转换：

Number("10") 转换为 number 类型parseInt("20.5") 转换为number 取整parseFloat("10.5") 转换为number 保留小数、String(10) 转换为 string 类型

Boolean(1) 转换为 Boolean 类型

隐式类型转换

1. 四则运算转换：

加法运算符+是双目运算符，只要其中一个是 string 类型，表达式的值便是一个String。对于其他的四则运算，只要其中一个是 Number 类型，表达式的便是一个 Number。

比如：

"10"+20，加号遇到字符串变成连接符，20 隐式转换为字符串；1+true， true 隐式转换为 1；

1. 判断语句转换：

判断语句中的判断条件需要是 Boolean 类型，所以条件表达式会被隐式转换为Boolean。其转换规则则同Boolean 的构造函数。比如:

10 == "10"，"10"隐式转换为 10

# 考点 2：传值 VS 传址

### 问题：请说出你对盒模型的理解阅读下面代码，说说运行的结果。

//代码 1： let a = 1; let b = a; b = 3;

console.log(a, b);

//代码 2：

let c = { name:"hello" }; let d = c;

c.name = "hi"; console.log(c, d);

代码 1 运行的结果是：1 3；

理由：基本类型的值是通过值复制的方式来赋值或是传递的，所以这里 b 的改变不会影响a 的。基本类型包括有 null、undefined、string、number、boolean 以及ES6 中的symbol。

代码 2 运行的结果是：{name:"Hi"} {name:"Hi"}；

理由：引用类型赋值并不会真正重新拷贝引用值，而是拷贝引用的地址，也就是传址。c 与 d 是引用同一个地址。所以如果修改了 c，d 的值也会被修改。

引用类型包括有数组、对象、函数等。

# 考点 3：作用域

### 问题：执行以下代码的结果是什么？为什么？

function test()

{ console.log(a);

console.log(foo()); var a = 1; function foo() {

return 2;

}

}

test();

这段代码的执行结果是 undefined 和 2。

原因是：变量和函数都被提升到了函数体的顶部。因此，当打印变量 a 时，它虽存在于函数体（因为 a 已经被声明），但仍然是 undefined。函数 foo 执行后得到的结果是 2。

此题考察是的变量提升的知识点，变量提升：可以理解为把变量声明提升到当前执行环境的最顶端。

### 问题：说说你对作用域链的理解？

当所需要的变量，在所在的作用域中查找不到的时候，它会一层一层向上查找，直到找到全局作用域还没有找到的时候，就会放弃查找。这种一层一层的关系，就是作用域链。

### 问题：执行以下代码的结果是什么？为什么？

var a = 1; function check() { return function()

{console.log(a); console.log(b);

}

}

var func = check(); func();

答案

这段代码的执行结果是：1 和 报错 b is not defined。

当作用域内找不到 a，会向上一层一层查找，最后找到了全局下的 a,输出结果为 1。同理，向上一层一层查找，最后找不到 b，所以输出"Uncaught ReferenceError: b is not defined"。

此题考察是的作用域和作用域链的知识点。

# 考点 4：闭包

### 问题：请讲一下你对闭包的认识和理解。

闭包是：函数和他能访问到所有变量就是一个闭包，目的是想让外界访问到函数内部的变量。

稍微详细一点的回答：

在有些情况下我们想让函数内的变量一直存放在内存里不被清除就可以使用闭包。

在 js 中变量的作用域属于函数作用域, 在函数执行完后，作用域就会被清理，内存也会随之被回收。但是由于闭包函数是建立在函数内部的子函数, 由于它可以访问上级作用域，即使上级函数执行完, 作用域也不会随之销毁,。

这时的子函数（也就是闭包），便拥有了访问上级作用域中变量的权限，即使上级函数执行完后作用域内的值也不会被销毁。

# 考点 5：原型链和继承

### 问题：说一下你的原型链和原型链继承的理解

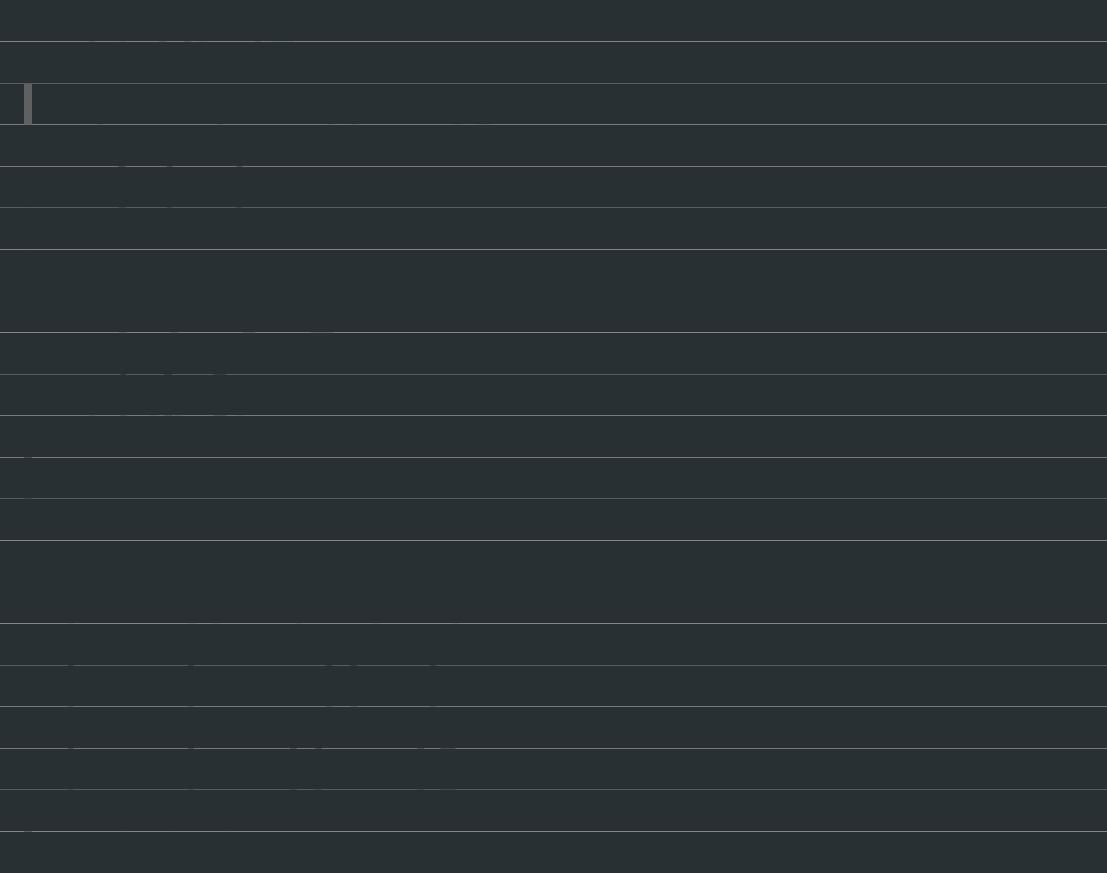
原型链：当对象查找一个属性的时候，如果没有在自身找到，那么就会查找自身的原型，如果原型还没有找到，那么会继续查找原型的原型，直到找到 Object.prototype 的原型时，此时原型为 null，查找停止。

这种通过通过原型链接的逐级向上的查找链被称为原型链。

原型链继承：一个对象可以使用其原型链上对象的属性或者方法，就称之为继承。具体是通过将这个对象的原型设置为另外一个对象，这样根据原型链的规则，如果查找一个对象属性且在自身不存在时，就会查找另外一个对象，相当于一个对象可以使用另外一个对象的属性和方法了。

### 问题：ES6 之前是如何使用 prototype 实现继承的？

先new一个要继承的新对象，然后把他放在子孙的原型链上。



function Father(name)

{this.name = name;

}

Father.prototype.myName = function ()

{return this.name;

}

// 继承属性，通过借用构造函数调用function Son(name, label) {

Father.call(this, name); this.label = label;

}

// 继承方法，创建备份

Son.prototype = Object.create(Father.prototype);

// 必须设置回正确的构造函数，要不然在会发生判断类型出错Son.prototype.constructor = Father;

Son.prototype.myLabel = function ()

{return this.label;

}

# 考点 6：BOM 与 DOM 操作

### 问题：什么是 BOM？什么是 DOM？

BOM 是即浏览器对象模型。 BOM 提供了独立于内容，是用来与浏览器窗口进行交互的对象； 由于 BOM 主要用于管理窗口与窗口之间的通讯，因此其核心对象是 window； BOM 由一系列相关的对象构成，并且每个对象都提供了很多方法与属性；

DOM 是文档对象模型。通过DOM 我们可以操作 HTML 中的元素，使得网页被下载到浏览器后，开发者可以根据需求进行页面内容的修改。

### 问题：说出几个你比较常用的 BOM 对象和方法？

window 对象，是 JS 的最顶层对象，其他的 BOM 对象都是 window 对象的属性； document 对象，文档对象；

location 对象，浏览器当前 URL 信息； navigator 对象，浏览器本身信息； screen 对象，客户端屏幕信息； history 对象，浏览器访问历史信息；

常用的方法： window.alert('提示信息') window.confirm("确认信息")

window.prompt("弹出输入框")

window.open("url 地址"，“打开的方式”，“新窗口的大小”） window.close() 关闭当前的网页

window.moveTo() - 移动当前窗口window.resizeTo() - 调整当前窗口的尺寸window.setTimeout(函数，时间) 只执行一次window.setInterval(函数，时间) 无限执行

# 考点 7：Ajax 与 跨域

### 问题：原生 js ajax 请求有几个步骤？分别是什么？

New对象

初始化open

发送请求send

Onreadystatechange回调里判断readystate和status

一共有 5 个步骤，如下：

1.创建 XMLHttpRequest 对象

var ajax = new XMLHttpRequest();

//2. 规定请求的类型、URL 以及是否异步处理请求。ajax.open('GET',url,true);

//3. 发送信息至服务器时内容编码类型

ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

//4. 发送请求ajax.send(null);

//5. 接受服务器响应数据ajax.onreadystatechange = function () {

if (obj.readyState == 4 && (obj.status == 200 || obj.status == 304)) {

}

};

### 问题：什么情况造成跨域？

同源策略限制，不同源会造成跨域。以下任意一种情况不同，都是不同源。

同源：1. 协议；2.域名；3.端口号三者要求全部相同，只要有一个不相同就是非同源策略。

### 问题：跨域解决方案有哪些？

1. 动态创建一个 script 标签，利用 script 标签的 src 属性不受同源策略限制。因为所有的 src 属性和 href 属性都不受同源策略限制。可以请求第三方服务器数据内容。

//去创建一个script 标签

var script = document.createElement("script");

//script 的src 属性设置接口地址 并带一个callback 回调函数名称script.src = "<http://xxxxx/index.php?callback=jsonpCallback>";

//插入到页面document.head.appendChild(script);

//通过定义函数名去接收后台返回数据function jsonpCallback(data){

//注意 jsonp 返回的数据是json 对象可以直接使用

//ajax 取得数据是 json 字符串需要转换成 json 对象才可以使用。

}

1. CORS：跨域资源共享

原理：服务器设置 Access-Control-Allow-OriginHTTP 响应头之后，浏览器将会允许跨域请求限制：浏览器需要支持 HTML5，可以支持 POST，PUT 等方法兼容ie9 以上

需要后台设置

Access-Control-Allow-Origin: \* //允许所有域名访问，或者Access-Control-Allow-Origin: [http://a.com](http://a.com/) //只允许所有域名访问

1. 使用一个代理

属于后端的解决方案，需要搭建一个中转 nginx 服务器，用于转发请求。也是最正规的解决方案

### 问题：请解释 JSONP 的工作原理？

在pageA中声明一个函数，在pageB写一个同名字的函数，并且把需要传递的数据以参数形式放在pageB中，然后在A中用script标签引入。

JSONP 是一个简单高效的跨域方式，HTML 中的 script 标签可以加载并执行其他域的 javascript，于是我们可以通过 script 标记来动态加载其他域的资源。

例如我要从域A 的页面 pageA 加载域 B 的数据，那么在域 B 的页面 pageB 中我以 JavaScript 的形式声明pageA 需要的数据，然后在 pageA 中用 script 标签把 pageB 加载进来，那么 pageB 中的脚本就会得以执行。

JSONP 在此基础上加入了回调函数，pageB 加载完之后会执行 pageA 中定义的函数，所需要的数据会以参数的形式传递给该函数。

JSONP 易于实现，但是也会存在一些安全隐患，如果第三方的脚本随意地执行，那么它就可以篡改页面内容，截获敏感数据。但是在受信任的双方传递数据，JSONP 是非常合适的选择。

# 考点 8：事件模型

### 问题：请说说什么是事件流？

事件是文档或者浏览器窗口中发生的，特定的交互瞬间。而事件流，描述的是从页面中接收事件的顺序。

### 问题：什么是事件冒泡？什么是事件捕获？

事件捕获阶段：事件从最上一级标签开始往下查找，直到捕获到事件目标(target)。事件冒泡阶段：事件从事件目标(target)开始，往上冒泡直到页面的最上一级标签。

### 问题：请说出阻止事件冒泡的几个办法？

1. event.stopPropagation();

事件处理过程中，阻止了事件冒泡，但不会阻击默认行为，比如：点击事件绑定在 a 标签的话，会执行了超链接的跳转。

1. return false;

事件处理过程中，阻止了事件冒泡，也阻止了默认行为。比如：点击事件绑定在 a 标签的话，不会执行超链接的跳转。

# 考点 9：this 关键字

### 问题：执行以下代码的结果是什么？为什么？

function

foo(){ console.log( this.a)

}

var a = 3; var obj = {

a: 2,

foo: foo

};

运行结果是 2，因为是 obj 调用的foo，所以 foo 的this 指向了obj，而 obj.a = 2。此题考察对 this 指向的理解。

### 问题：执行以下代码的结果是什么？为什么？

var name = "windows-name"; function test() {

var name = "cherry"; console.log(this.name);

}

test();

运行结果是 windows-name，因为 this 永远指向最后调用它的那个对象，最后调用 a 的地方 a()；前面没有调用的对象。那么就是全局对象 window，这就相当于是 window.a()；注意，这里我们没有使用严格模式，如果使用严格模式的话，全局对象就是 undefined，。

此题考察对 this 指向的理解。

# 考点 10：es6篇

### 问题：说说你知道那些es6特性（面试）

es6地址https://es6.ruanyifeng.com/

1. let const 2.变量的解构赋值 3.symbol 4.set和map的数据结构（set主要用于去重）5.proxy（vue3的重点） 6.promise对象（如何实现，与async，await的区别）7.async await（返回值是promise）8.generator（react使用比较多）9.class,class的继承（node，react主要使用）

主要看文档啃熟悉，与技术官面试才能游刃有余，可以先按上面的9点来啃熟悉，其他稳定里的可以认识就行，因为这9点是重点考察对象

### 问题：var、let、const之间的区别（笔试）

var声明变量可以重复声明，而let不可以重复声明var是不受限于块级的，而let是受限于块级

var会与window相映射（会挂一个属性），而let不与window相映射

var可以在声明的上面访问变量，而let有暂存死区，在声明的上面访问变量会报错const声明之后必须赋值，否则会报错

const定义不可变的量，改变了就会报错

const和let一样不会与window相映射、支持块级作用域、在声明的上面访问变量会报错

### 问题：介绍下 Set、Map的区别（比较少被问道，基本问的是set）

应用场景Set用于数据重组，Map用于数据储存Set

1. Map是键值对，Set是值得集合，当然键和值可以是任何的值；
2. Map可以通过get方法获取值，而set不能因为它只有值；
3. 都能通过迭代器进行for...of遍历；
4. Set的值是唯一的可以做数组去重，Map由于没有格式限制，可以做数据存储；

### 问题：介绍下promise，有用过all方法吗？（重要，百分之90会问到，除非到不了这里就被pass）

Promise 是异步编程的一种解决方案。Promise对象有以下两个特点，对象的状态不受外界影响和一旦状态改变，就不会再变，任何时候都可以得到这个结果，promise对象有三种状态：pending（进行中）、fulfilled

（已成功）和rejected（已失败）。Promise对象是一个构造函数，用来生成Promise实例，Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。promise的方法(有些人叫api)有then， catch，finally，resolve,reject等。

Promise.all()方法可以将多个Promise实例包装成一个新的Promise实例，用数组放一起同时执行。同时，成功和失败的返回值是不同的，成功的时候返回的是一个结果数组，而失败的时候则返回最先被reject失败状态的值

需要特别注意的是，Promise.all获得的成功结果的数组里面的数据顺序和Promise.all接收到的数组顺序是一致的，即p1的结果在前，即便p1的结果获取的比p2要晚。这带来了一个绝大的好处：在前端开发请求数据的过程中，偶尔会遇到发送多个请求并根据请求顺序获取和使用数据的场景，使用Promise.all毫无疑问可以解决这个问题。

### 问题：promise和async await的区别(用自己的话总结)

Promise的出现解决了传统callback函数导致的“地域回调”问题，但它的语法导致了它向纵向发展行成了一个回调链，遇到复杂的业务场景，这样的语法显然也是不美观的。而async await代码看起来会简洁些，使得异步代码看起来像同步代码，await的本质是可以提供等同于”同步效果“的等待异步返回能力的语法糖，只有这一句代码执行完，才会执行下一句。

async await与Promise一样，是非阻塞的。

async await是基于Promise实现的，可以说是改良版的Promise，它不能用于普通的回调函数。

### 问题：如何去重？（重要，百分之70，要不笔试，要不面试的时候问）

[...new set(arr)]

### 问题：for in 和for of的区别（面试）

for...in 语句用于遍历数组或者对象的属性（对数组或者对象的属性进行循环操作）。for in得到对对象的key或数组,字符串的下标

for of和forEach一样,是直接得到值for of不能对象用

### 问题：null和undefined的区别（笔试）

总结来说：null表示没有对象，即该处不应该有值。undefined表示缺少值，即此处应该有值，但没有定义

### 问题：操作数组，对象的常用方法有那些（重要）

数组：

1. push() 向数组最里面推一个或多个数据
2. unshift()在数组的开头添加一个或多个数据
3. pop()在数据的尾部删除一个数据
4. shift()在数组的开头删除一个数据
5. splice() arr.splice(参数1,参数2,参数3,参数4,参数5, 参数n)，参数1表示要删除的开始位置, 参数2表示删除的

个数,参数3,参数4,参数5, 参数n,表示在删除位置添加的字符

1. concat()把两个或多个数组连接成一个数组
2. sort()数组排序方法,默认是按照字符编码逐个字符进行排序
3. join()按照我们特定的方式（指定字符）把数组转化成字符串
4. split()把字符串转化成数组(按照指定的字符进行分割)
5. Array.isArray() 判断数据是否是数组
6. forEach() 用于遍历我们的数组,对数组中的每一个元素进行操作,没有返回值
7. Map()作用: 对数组进行遍历,进行某个操作,然后返回一个新的数组
8. filter() 作用: 用于进行过滤,筛选出符合条件的元素,组成一个新的数组返回
9. reduce()作用: 将前一项和后一项的值进行运算,返回累积的结果.
10. some()作用: 只要数组中的某一个元素符合指定的条件,就返回真,否则返回假,可以和逻辑运算符或(||)类比
11. find:用于查找某个元素,如果找不到undefined,findIndex:用于查找某个元素的索引,如果找不到就返回-1 对象：(面试直接问道对象的比较少，对于经常使用比如create，assign，freeze，is，keys要知道)



### 问题：异步加载的方式有那些（重要，笔试，面试）

数组：

1. Defer
2. HTML5为<script>元素定义的async属性3.动态创建<script>标签

# 考点 11：编程题

### 编程题：编写代码，实现数组扁平化，把[1, [2, [3, 4]]] 转换成 [1, 2, 3, 4]。

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));

### 编程题：编写代码，使用 ES6 特性，实现数组[1, 1, 2, 2]去重。

Array.from(new Set([1, 1, 2, 2]))

### 编程题：创建 10 个a 标签，点击弹出对应序号。

var a

for (let i = 0; i< 10; i++) {

a = document.createElement('a'); a.innerHTML = i + "<br>"; a.addEventListener("click", function(e) {

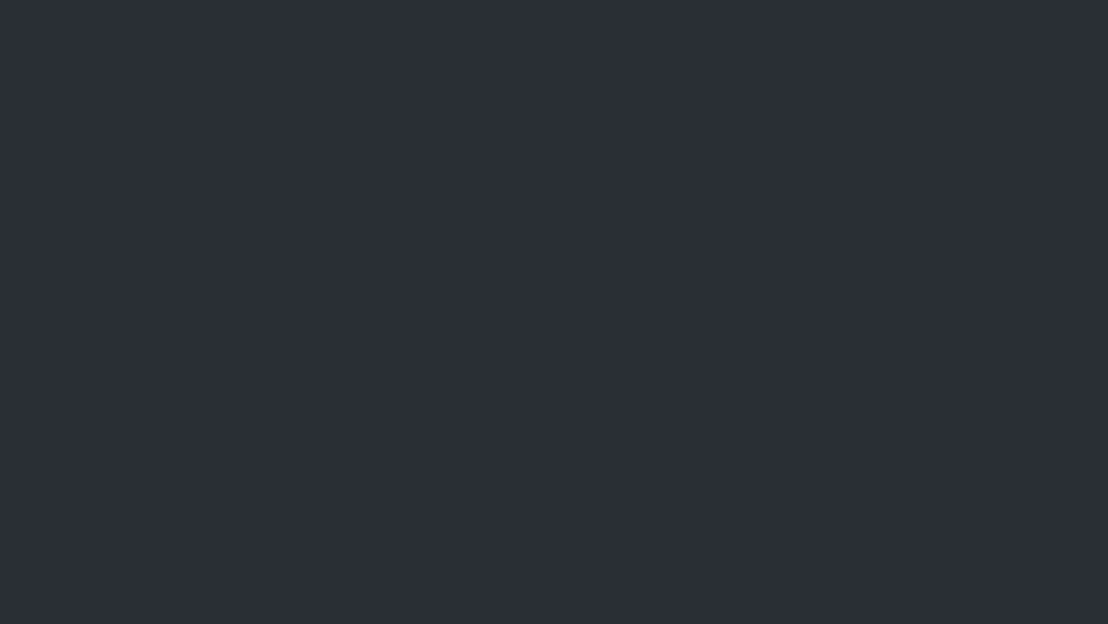
e.preventDefault(); alert(i);

})

document.body.appendChild(a)

}

### 编程题：封装一个冒泡排序的函数。对数组 [11,37,13,92,21,68] 进行排序



function sorts(arr){

for(var i = 0; i < arr.length-1;

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

if(arr[j] > arr[j+1]){ var temp = arr[j]; arr[j] = arr[j+1]; arr[j+1] = temp;

}

}

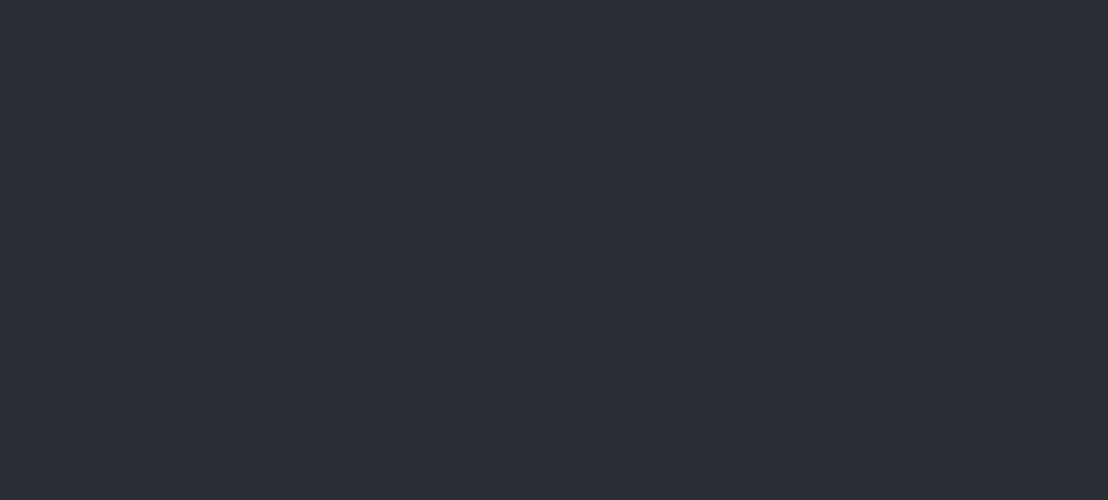
}

return arr;

}

var arr=[11,37,13,92,21,68];

### 编程题：编写一个函数，实现移除数组 arr 中的所有值与 item 相等的元素。不要直接修改数组 arr，结果返回新的数组。



function remove(arr, item)

{var newArr = []; for(var

i=0;i<arr.length;i++){if(arr [i]!=item){

newArr.push(arr[i]);

}

}

return newArr;

}

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

remove(arr,item);

### 编程题：编写一个函数，实现在数组 arr 中，查找值与 item 相等的元素出现的所有位置。

function find(arr, target)

{var temp = [];

arr.forEach(function(val,index){

val !== target || temp.push(index);

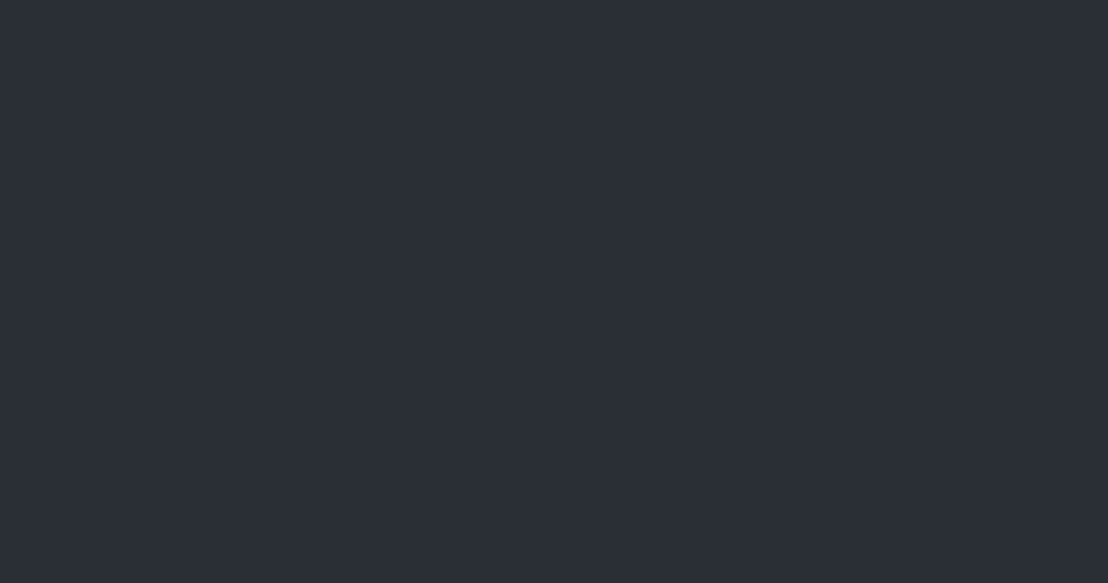
});

return temp;

}

find(['ab','b','a','ac','a'],'a');

### 编程题：var arr = [1,2,3,4,5] ; var arr2 = [3,4,9,5,6,7]; 将两数组比较，要求将 arr 里相同的部分与 arr2不同的部分合并得到新数组 [3,5,4,9,6,7]。



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

var arr2 = [3,4,9,5,6,7];

var temp1 = arr2.filter((x)=>{return arr.includes(x)

})

var temp2 = [];

for(var i = 0; i < arr2.length; i++)

{ if(temp1.indexOf(arr2[i]) === -1 )

{

temp2.push(arr2[i])

}

}

var last = temp1.concat(temp2);

### 编程题：用reduce 统计一个数组 ["apple","orange","apple","orange","pear","orange"] 中单词出现的次数。

var arr = ["apple","orange","apple","orange","pear","orange"]; function getWordCnt(){

return

arr.reduce(function(prev,next,index,arr){prev[ next] = (prev[next] + 1) || 1;

return prev;

},{});

}

### 编程题：分别使用es5的闭包和es6的let

