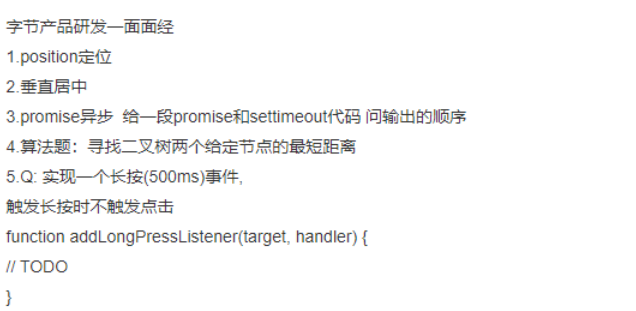
面试官你好，我叫许楷沂，是东莞理工学院一名大三的学生，前阵子很有幸参加了字节跳动的训练营，在学习react的情况下做了一个今日头条的简易版搜索项目，成绩也理想，和另一位同学并列第一，然后在学校大一大二做的是java后端的开发，后来大三才开始了前端的学习，也在同校的朋友开创的公司东莞绿灯科技有限公司中做项目学习了半年。后来因为觉得有些东西是在赶进度的过程中是学不到的，然后就没有继续做下去了。然后这次是我第一次投简历和面试，会有些紧张，同时我也觉得人生的每个阶段都是在不断地学习，这个面试也是一个学习的机会，希望面试官能够多多包容，遇到不会的问题可以解答一下。

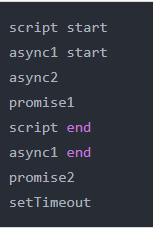
感激不尽。

这可能和多个方面的因素有关吧，第一个因为在学校里面教的都是java后端的知识，整个班级都是学的一样的知识而没有前端的课程，可能我不想和别人太一样吧，我就好奇的去看，为什么没有前端的课程？是太简单么？看完我觉得并不是这样，可能是之前的前端只是静态界面，处于一个不大重要的地位，但是现在不同了，大前端的趋势已经出来了，我们做前端的也可以自己写接口，做一个完整的项目了，所以这可能就是影响我的第二个要素，大前端的趋势吧。

在公司中有些东西是学不到的



3、



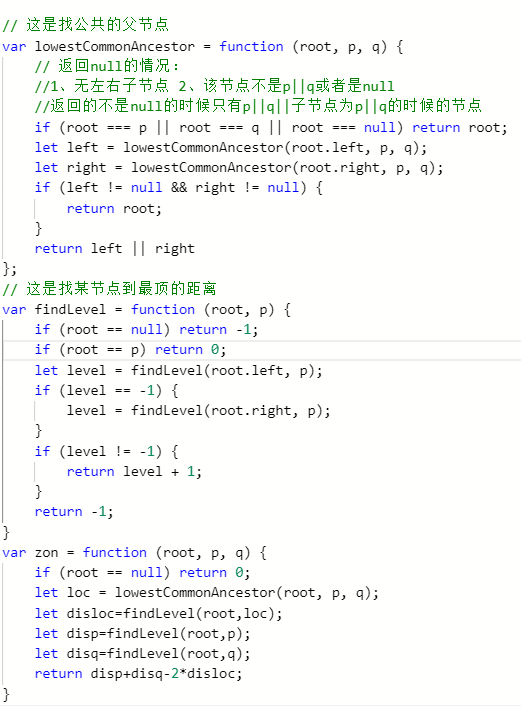
4、假设给定的节点为node1,node2,可以分为下面的两种情况：

1）node1是node2的祖先节点或孩子结点，可以理解为两个节点在一条线上。例如：Dist(2,4),Dist(6,1)

2）node1和node2没有直接或间接的父子关系。例如，Dist(4,3)，他们需要一个共同的祖先结点1连接起来

假设lca是两个节点的最低公共祖先节点：

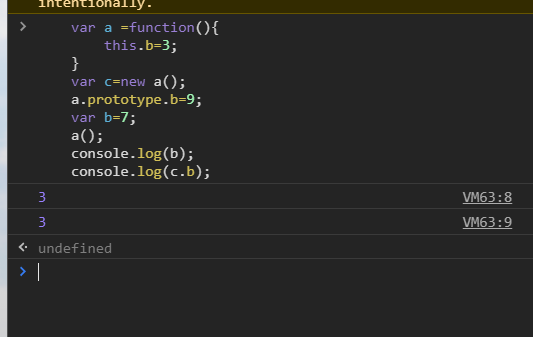
Dist(n1,n2)=Dist(root,n1)+Dist(root,n2)-2\*Dist(root,lca)



5、





1. JS数据类型：null undefined Number String Boolean ;;array object function
2. 

因为c是a的实例对象，a是构造函数。

一开始往a的原型中添加了b=9；所以c的原型\_\_proto\_\_中也会有一个b=9；而c中仍然还是会有a中有的this.b=3;

然后执行了a()；将全局的b变成了3；

然后打印b也就是3；

然后打印c.b由于c是a的实例对象，所以其实c.b=a(){this.b=3}=this.b=3所以打印3

1. Span设置宽高无效，因为他是行内元素，marginpadding的top与bottom无效
2. Position：relative相对父元素进行定位。Position:absolute 相对浏览器左上角进行定位；

有时候我们需要将absolute相对于父元素进行定位，那么我们在父元素中加relative即可。

1. 实现三角形：
2. 、border
3. 、transform旋转45度实现:



1. 、特殊菱形字符实现。

MVVM模式：Model逻辑层；View视图层；ViewModel视图模型层；

Model通过ajax通信，然后将数据发送给ViewModel

ViewModel也可以通过ajax进行通信，发送给model。

ViewModel进行数据改变，改变view上显示的内容。

View上的数据进行更新，ViewModel也会接收到改变的数据。

vue数据绑定：

Vue2.0的数据绑定是通过Object.defineProperty()这个方法重新定义了对象的get（）方法和set（）方法从而进行了数据劫持

defineProperty(对象，属性名，属性描述符)

其中：属性描述符有：

        writable:"",//是否可以修改

        configurable:"",//是否可以删除

        enumerable:"",//是否可以枚举

        value:"",//值



Object.defineProperty的缺点：无法监听数组的变化

所以vue3.0起用了Proxy（target,handler）

Target是需要添加代理的对象，handler是对象中的操作。



区别：proxy可以直接监听对象而非属性，就是对象的所有属性都可以进get和set；

Proxy有13种拦截方法，而defineProperty是不具备的。

1. 跨域：

（1）、Jsonp：在HTML标签里，一些标签比如script、img这样的获取资源的标签是没有跨域限制的，利用这一点，我们可以这样干：

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<script type='text/javascript'>

*// 后端返回直接执行的方法，相当于执行这个方法，由于后端把返回的数据放在方法的参数里，所以这里能拿到res。*

window.jsonpCb = **function** (res) {

console.log(res)

}

</script>

<script src='http://localhost:9871/api/jsonp?msg=helloJsonp&cb=jsonpCb' type='text/javascript'></script>

</body></html>

缺点：只能发送get请求

（2）、空iframe+form:

**const** requestPost = ({url, data}) => {

*// 首先创建一个用来发送数据的iframe.*

**const** iframe = document.createElement('iframe')

iframe.name = 'iframePost'

iframe.style.display = 'none'

document.body.appendChild(iframe)

**const** form = document.createElement('form')

**const** node = document.createElement('input')

*// 注册iframe的load事件处理程序,如果你需要在响应返回时执行一些操作的话.*

iframe.addEventListener('load', **function** () {

console.log('post success')

})

form.action = url

*// 在指定的iframe中执行form*

form.target = iframe.name

form.method = 'post'

**for** (**let** name **in** data) {

node.name = name

node.value = data[name].toString()

form.appendChild(node.cloneNode())

}

*// 表单元素需要添加到主文档中.*

form.style.display = 'none'

document.body.appendChild(form)

form.submit()

*// 表单提交后,就可以删除这个表单,不影响下次的数据发送.*

document.body.removeChild(form)

}*// 使用方式*

requestPost({

url: 'http://localhost:9871/api/iframePost',

data: {

msg: 'helloIframePost'

}

})

（3）、CORS:浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）

后端：

*// 处理成功失败返回格式的工具***const** {successBody} = require('../utli')**class** **CrossDomain** {

**static** **async** **cors** (ctx) {

**const** query = ctx.request.query

*// \*时cookie不会在http请求中带上*

ctx.**set**('Access-Control-Allow-Origin', '\*')

ctx.cookies.**set**('tokenId', '2')

ctx.body = successBody({msg: query.msg}, 'success')

}

}

module.exports = CrossDomain

前端啥都不用做

反向代理：vue啊，react啊，nignx啊等等都有。

1. http缓存：分为数据库缓存，服务器缓存，浏览器缓存

浏览器缓存分为：强缓存和协商缓存

强缓存：比如说我url重新回车的时候，对于强缓存就会直接从本地的缓存中进行数据对比，如果没有过期就调用这个缓存内容，不会去请求数据。

强缓存是通过http返回头中的cahce-control和expire这两个标识来控制的

Cache-control是资源有效期；expire是过期时间，cache-control优先级高于expire

协商缓存是如果没有命中强缓存，那么就会执行协商缓存

Last-modify这个在header中的变量，表示资源的最后修改时间。

发送请求时，请求头包含If-Modify-Since，这个值就是Last-modify,然后服务器判断是否命中缓存。命中返回304，否则返回200+资源。

Etag：是资源的一个校验码，资源发生变化，则etag发生变化。

服务器根据浏览器上的If-None-Match来判断是否命中缓存。

1. localstorage生命周期是永久，而sessionstorage是随着浏览器当前窗口或者标签页的关闭而清除的。不同浏览器无法共享localstorage和sessionstorage。

相同浏览器的不同页面可共享同域名端口的localstorage；

Iframe下可共享sessionstorage

Cookie可以设置失效时间，超过该时间就会自动删除。4kb

Cookie每次都会携带http请求头，如果使用cookie保存过多数据会带来性能问题。

11、

<!-- 版本号排序 -->

<script>

    // 选择排序法

    const chooseSort = arr => {

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

                let minindex = i;

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

                    if (arr[j] < arr[minindex]) {

                        minindex = j;

                    }

                }

                [arr[i], arr[minindex]] = [arr[minindex], arr[i]]

            }

            return arr;

        }

        //将if里面的内容替换

        //最终代码

    const versionsort = version => {

        const temp = version.map(v => v.split('.'));

        //变成[[1,45,0],[1,5],[1,6],[3,3,3,3,3,3]]

        //接下来是选择排序

        for (let i = 0; i < temp.length; i++) {

            let minindex = i;

            for (let j = i; j < temp.length; j++) {

                for (let k = 0; k < temp[j].length; k++) {

                    const current = +temp[j][k],

                        min = +temp[minindex][k];

                    if (current < min) {

                        minindex = j;

                    }

                    //只要不等,就立刻结束最内层for循环

                    if (current !== min) {

                        break;

                    }

                }

            }

            [temp[i], temp[minindex]] = [temp[minindex], temp[i]];

        }

        return temp.map(v => v.join('.'))

    }

    console.log(versionsort(['1.45.0', '1.5', '6', '3.3.3.3.3.3.3']))

    // 第二种解法sort

    function rule(str1, str2) {

        var arr1 = str1.split('.'), //去除'.'，将剩下的数字转换为数组

            arr2 = str2.split('.'),

            minLen = Math.min(arr1.length, arr2.length), //取出两个数组中的最小程度

            maxLen = Math.max(arr1.length, arr2.length); //最大长度

        //以最短的数组为基础进行遍历

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

            //这里需要转换后才进行比较，否则会出现'10'<'7'的情况

            if (parseInt(arr1[i]) > parseInt(arr2[i])) {

                return 1; //返回一个大于0的数，表示前者的index比后者的index大

            } else if (parseInt(arr1[i]) < parseInt(arr2[i])) {

                return -1; //返回一个小于0的数，表示前者的index比后者的index小

            }

            //因为不只进行一次计较，所以这里不对相等的两个数进行处理，否则有可能第一次比较就返回，不符合要求

            //这个是为了区分'4.8'和'4.8.0'的情况

            //在前面的比较都相同的情况下，则比较长度

            //位数多的index大

            if (i + 1 == minLen) {

                if (arr1.length > arr2.length) {

                    return 1;

                } else {

                    return -1;

                }

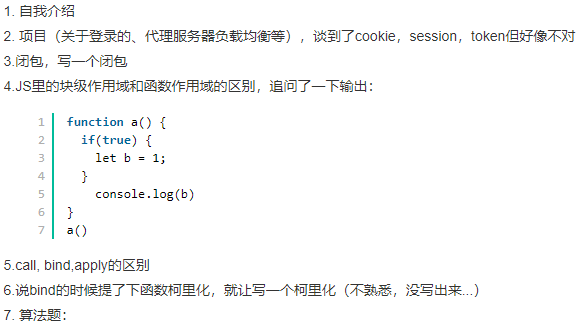
            }

        }

    }

    console.log(['1.45.0', '1.5', '6', '3.3.3.3.3.3.3'].sort(rule))

</script>



1. 闭包：在外部可以间接访问其他函数内部变量的函数，叫做闭包

    function a() {

        let flag = 2;

        function fa() {

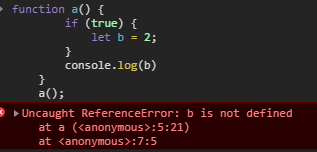
            console.log(flag);

        }

        return fa();

    }

    a();

4、块级作用域就是只在某一个快里面起作用，就像上面的if的{}里面，b是let声明的，b只在{}里面起作用，然后console.log(b)的b是函数作用域里面的，但是没有声明，所以报错了。

5、bind、call、apply的区别：

bind(this,参数1，参数2，……)

Call（this,参数1，参数2，……）

Apply(this,参数数组)

Apply和call会立即执行，

Bind不会立即执行，而是返回一个新函数，且函数里面的参数是原来的参数数组，不会被你改变而改变。

Demo

function add(a, b) {

return a + b;

}

const addBind = add.bind(null, 1);（指的是1，不会被改变）

addBind(2); // 3（2是b参数，也就是柯里化的概念）

其实也就相当于

6、function add(a){

return function (b){

return a+b;  
 }

}

add(1)(2).

也就是柯里化的概念：把接收的多个参数的函数变成单一参数的函数。

实现sum(2)(3)(4)和sum(2,3,4)

7、

给一棵二叉树和一个值，检查二叉树中的是否存在一条路径，这条路径上所有节点的值加起来等于给的那个初始值。

例如，对于下面的二叉树，如果初始值是 22，那么存在一条路径 5->4->11->2

      5

     / \

    4   8

   /   / \

  11   13  4

  / \      \

7  2      1

请实现如下这个函数

     // 路径总和返回true||false

    function hasPathSum(root, sum) {

        if (root == null) return false;

        if (root.left == null && root.right == null) {

            return sum = root.val == 0

        }

        return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, sum - root.val)

    }

    // 路径总和返回数组列表

    function pathSum(root, sum) {

        let res = [];

        let huisu = function(root, sum, arr) {

            if (root === null) return;

            arr.push(root.val);

            if (root.left === null && root.right === null && root.val === sum) {

                res.push([...arr])

            }

            huisu(root.left, sum - root.val, arr);

            huisu(root.right, sum - root.val, arr);

            // 回溯，要把arrpop

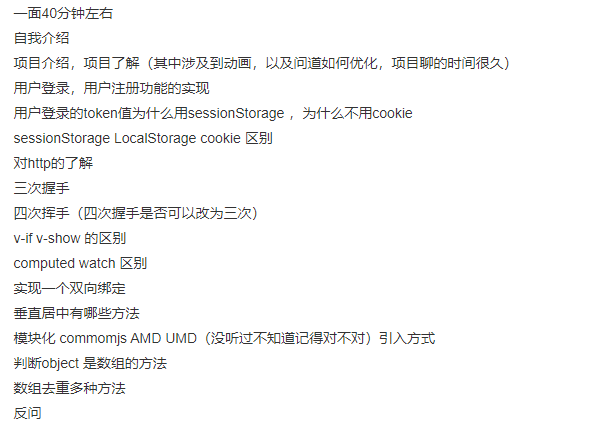
            arr.pop();

        }

        huisu(root, sum, []);

        return res;

    }



模块化：

CommonJS：

暴露出来,然后用require，复杂的数据类型是浅拷贝

Function1:……

module.exports={

Function1:Function1;

}

Let fun=require(‘./x.js’)

ES6模块：动态只读模块,不可以修改。

Export let counter={ count:1 }

Import {counter} from ‘./b.js’

AMD：一个定义，RequireJS则是它的实现框架。可以异步加载js文件，



CMD：SeaJs 同步模块定义，淘宝团队提供



判断Object是数组（函数）的方法：

一、Array.isArray();ES5中不支持IE8以下。

二、使用原型链：[].\_\_proto\_\_===Array.prototype //true

三、使用Object。Prototype.toString.call([])===‘[object Array]’

四、instanceof操作符:arr instanceof Array //true

(PS:instanceof是通过检测对象的原型链是否指向构造函数的prototype对象)

数组去重多种方法：

1. ES6中的set去重

**function** **unique** (arr) {

**return** Array.from(**new** Set(arr))

}**var** arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},{}];console.log(unique(arr))

1.2、

[...**new** Set(arr)]

1. ES5：利用for嵌套for，然后splice去重

**function** **unique**(arr){

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

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

**if**(arr[i]==arr[j]){ *//第一个等同于第二个，splice方法删除第二个*

arr.splice(j,1);

j--;

}

}

}**return** arr;

}**var** arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},{}];

console.log(unique(arr))

1. 先排序，然后for循环一遍，对比相邻元素。

**function** **unique**(arr) {

arr = arr.sort()

**var** arrry= [arr[0]];

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

**if** (arr[i] !== arr[i-1]) {

arrry.push(arr[i]);

}

}

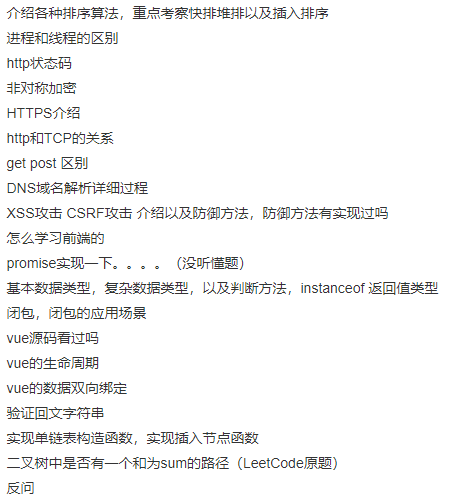
**return** arrry;

}

**var** arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},{}];

console.log(unique(arr))

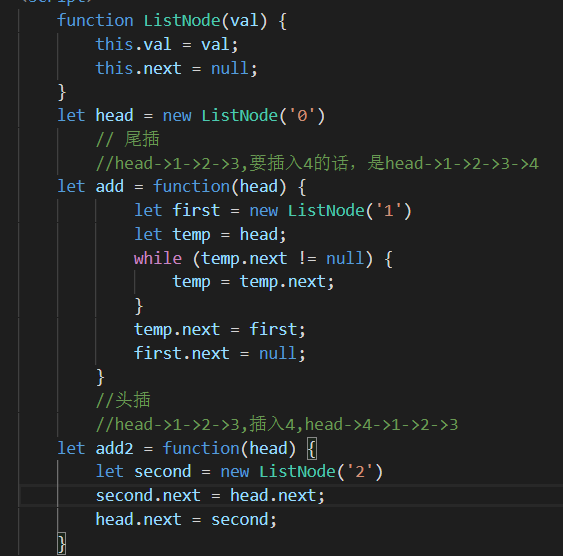
四、利用map，如果map.has()某个值，就忽略，然后转成数组。



回文字符串



实现单链表



二叉树中一个和为sum的路径



<script>

    function repeat(func, times, wait) {

        return function(str) {

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

                setTimeout(func(str), wait);

            }

        }

    }

    var repeatfun = repeat(console.log, 4, 3000);

    repeatfun('helloworld')

</script>

36进制转换

    // 36进制转换

    var zhuanhuan = function(num) {

        var s = '0123456789abcdefghijklmnopqrstuvwxyz'

        var res = [];

        while (num != 0) {

            res.push(s[num % 36]);

            num = parseInt(num / 36);

        }

        return res.reverse().join("");

    }

    console.log(zhuanhuan(1094));

原生ajax



数组降维，去重

已知如下数组：

var arr = [ [1, 2, 2], [3, 4, 5, 5], [6, 7, 8, 9, [11, 12, [12, 13, [14] ] ] ], 10];

编写一个程序将数组扁平化去并除其中重复部分数据，最终得到一个升序且不重复的数组

var arr = [ [1, 2, 2], [3, 4, 5, 5], [6, 7, 8, 9, [11, 12, [12, 13, [14] ] ] ], 10]

// 扁平化

let flatArr = arr.flat(4)

// 去重

let disArr = Array.from(new Set(flatArr))

// 排序

let result = disArr.sort(function(a, b) {

return a-b

})

console.log(result)

// [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

快排

    function Insertion(arr) {

        let len = arr.length;

        let preIndex, current;

        for (let i = 1; i < len; i++) {

            preIndex = i - 1;

            current = arr[i];

            while (preIndex >= 0 && current < arr[preIndex]) {

                arr[preIndex + 1] = arr[preIndex]

                preIndex--;

            }

            arr[preIndex + 1] = current;

        }

        return arr;

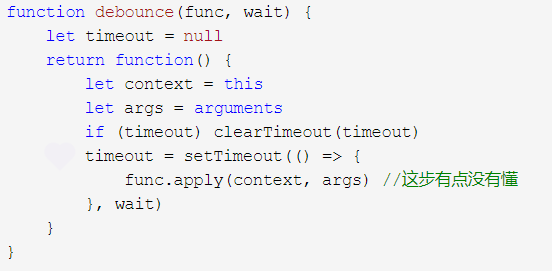
    }

    var arr = [3, 5, 7, 8, 9, 33, 55, 77, 22, 44, 56]

    console.log(Insertion(arr))



防抖：防抖，即短时间内大量触发同一事件，只会执行一次函数，实现原理为设置一个定时器，约定在xx毫秒后再触发事件处理，每次触发事件都会重新设置计时器，  
直到xx毫秒内无第二次操作，防抖常用于搜索框/滚动条的监听事件处理，如果不做防抖，每输入一个字/滚动屏幕，都会触发事件处理，造成性能浪费。



节流：防抖是延迟执行，而节流是间隔执行，函数节流即每隔一段时间就执行一次，实现原理为设置一个定时器，约定xx毫秒后执行事件，如果时间到了，那么执行函数并重置定时器，和防抖的区别在于，  
防抖每次触发事件都重置定时器，而节流在定时器到时间后再清空定时器

