XXS和csrf

ES6新特性

let与const声明的变量解决了这种问题,因为他们是块级作用域,。

For……of for……in for

ES6箭头函数=>有点像匿名函数的简写，this的值和函数外面的this的值是一样的.匿名函数指向全局。

回调函数是为了解决js异步操作的问题。为了保证异步操作的顺序，需要进行嵌套，当层数过多的时候就出现了回调地狱的问题。可以使用Promise解决这个问题。

异步操作：

读文件

Ajax

// 封装一个get请求的方法

function getJSON(url) {

    return new Promise(function(resolve, reject) {

        var XHR = new XMLHttpRequest();

        XHR.open('GET', url, true);

        XHR.send();

        XHR.onreadystatechange = function() {

            if (XHR.readyState == 4) {

                if (XHR.status == 200) {

                    try {

                        var response = JSON.parse(XHR.responseText);

                        resolve(response);

                    } catch (e) {

                        reject(e);

                    }

                } else {

                    reject(new Error(XHR.statusText));

                }

            }

        }

    })

}

getJSON(url).then(function(res){

Console.log(res)

});

定时器

事件绑定

var wait1000 = new Promise(function(resolve, reject) {

setTimeout(resolve, 1000);

}).then(function() {

console.log('Yay!');

});

闭包

获取到函数内部的局部变量，函数外部无法获取内部的局部变量，我定义一个函数内部的函数来和外部建立连接的桥梁。函数可以return这个闭包函数。

函数提升和变量提升

JavaScript 中，函数及变量的声明都将被提升到函数的最顶部。

**函数提升优先级比变量提升要高，且不会被变量声明覆盖，但是会被变量赋值覆盖。**

用css实现一个三角形

.con{

width: 0;

height: 0;

border-width: 40px;

border-style: solid;

border-color: red transparent transparent transparent;

}

http和https的区别

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，于是网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。

HTTPS和HTTP的区别主要如下：

　　1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

　　2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

　　3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

　　4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

http支持的方法

|  |  |  |
| --- | --- | --- |
| 1 | GET | 请求指定的页面信息，并返回实体主体。 |
| 2 | HEAD | 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| 3 | POST | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 |
| 4 | PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| 5 | DELETE | 请求服务器删除指定的页面。 |
| 6 | CONNECT | HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。 |
| 7 | OPTIONS | 允许客户端查看服务器的性能。 |
| 8 | TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |

跨域

跨域问题是由于的同源策略的限制导致，不同源的文件限制交互。

现在流行前后端分离的开发模式，前端和后端可能部署在不同的服务器上，端口也不同。必然出现跨域问题。

解决方案有一些：

jQuery的ajax解决跨域问题

jQuery插件-[jquery-jsonp](https://github.com/congmo/jquery-jsonp)

使用cors

回文算法(api实现和底层原理)

function palindrome(str) {

var str1 = str.toLowerCase();

var reg = /[\W\\_]/g; //删除所有非字母数字字符和下划线

var str2 = str1.replace(reg, "");

var str3 = str2.split("");

var str4 = str3.reverse();

var str5 = str4.join("");

return str2 === str5;

}

var out=palindrome("eye");

console.log(out);

输入一个url敲下回车  过程

<https://blog.csdn.net/xdzhouxin/article/details/81262087>

应用层DNS解析域名

应用层客户端发送HTTP请求

传输层TCP传输报文

网络层IP协议查询MAC地址

数据到达数据链路层

服务器接收数据

服务器响应请求

服务器返回相应文件

解析HTML以构建DOM树 –> 构建渲染树 –> 布局渲染树 –> 绘制渲染树。

cookie和session知道的都说说(面试官原话)  
闭包  
状态码

|  |  |
| --- | --- |
| 1\*\* | 信息，服务器收到请求，需要请求者继续执行操作 |
| 2\*\* | 成功，操作被成功接收并处理 |
| 3\*\* | 重定向，需要进一步的操作以完成请求 |
| 4\*\* | 客户端错误，请求包含语法错误或无法完成请求 |
| 5\*\* | 服务器错误，服务器在处理请求的过程中发生了错误 |

200 - 请求成功

301 - 资源（网页等）被永久转移到其它URL

404 - 请求的资源（网页等）不存在

500 - 内部服务器错误

## **301重定向和302重定向的区别**

　　302重定向只是暂时的重定向，搜索引擎会抓取新的内容而保留旧的地址，**因为服务器返回302，所以，搜索搜索引擎认为新的网址是暂时的。**

**而301重定向是永久的重定向，搜索引擎在抓取新的内容的同时也将旧的网址替换为了重定向之后的网址。**

三次握手过程：

所谓的“三次握手”即对每次发送的[数据](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE/5947370" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)量是怎样跟踪进行协商使[数据段](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%AE%B5/5136260" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)的发送和接收同步，根据所接收到的数据量而确定的数据确认数及数据发送、接收完毕后何时撤消联系，并建立虚连接。

第一次握手：建立连接时，[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)发送[syn](https://baike.baidu.com/item/syn" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)包（syn=j）到[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)，并进入[SYN\_SENT](https://baike.baidu.com/item/SYN_SENT" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)状态，等待服务器确认；SYN：同步序列编号。

[第二次握手](https://baike.baidu.com/item/%E7%AC%AC%E4%BA%8C%E6%AC%A1%E6%8F%A1%E6%89%8B" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)：[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)收到[syn](https://baike.baidu.com/item/syn" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)包，必须确认客户的SYN（[ack](https://baike.baidu.com/item/ack" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入[SYN\_RECV](https://baike.baidu.com/item/SYN_RECV" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)状态；

第三次握手：[客户端](https://baike.baidu.com/item/%E5%AE%A2%E6%88%B7%E7%AB%AF" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)收到[服务](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)器的SYN+ACK包，向[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)发送确认包ACK([ack](https://baike.baidu.com/item/ack" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)=k+1），此包发送完毕，客户端和服务器进入[ESTABLISHED](https://baike.baidu.com/item/ESTABLISHED" \t "https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B/_blank)（TCP连接成功）状态，完成三次握手。

浮点数相加   得啥  为啥？

上面我们发现0.1和0.2转化为二进制之后，变成了一个无限循环的数字，这在现实生活中，无限循环我们可以理解，但计算机是不允许无限循环的，对于无限循环的小数，计算机会进行舍入处理。进行双精度浮点数的小数部分最多支持 52 位，所以两者相加之后得到这么一串 0.0100110011001100110011001100110011001100110011001100 因浮点数小数位的限制而截断的二进制数字，这时候，我们再把它转换为十进制，就成了 0.30000000000000004。

把需要计算的数字升级（乘以10的n次幂）成计算机能够精确识别的整数，等计算完毕再降级（除以10的n次幂），这是大部分编程语言处理精度差异的通用方法。

给一个字符串  把重复的字母都用一个对象整出来  找出出现次数最多的和是什么数字

var str=" shu sad a asd"

var str1=str.replace(/\s/g, "")

var str\_shuzu=str1.split("");

var nc=[];

var nc\_num=[];

for(var i=0;i<str\_shuzu.length;i++){

var index=nc.indexOf(str\_shuzu[i]);

if(index!=-1){

nc\_num[index]++;

}else{

nc.push(str\_shuzu[i])

nc\_num.push(1)

}

}

var index\_max=nc\_num.indexOf(Math.max.apply(Math, nc\_num))

console.log(nc\_num[index\_max]+" "+nc[index\_max])

常见数组操作

https://blog.csdn.net/wang729506596/article/details/83019131

1、JSON.stringify的实现，要考虑null、undefined，循环引用的情况下需要报错

这个想了半天，最后说遍历+字符串的拼接。

至于循环引用，因为写深拷贝的时候处理过，所以这里也拿过来讲，思路一样，代码在这：[https://github.com/pflhm2005/JS-components/blob/master/deepClone.js](https://github.com/pflhm2005/JS-components/blob/master/deepClone.js" \t "_blank)

2、嵌套数组的环形遍历，简单讲就是[[1,2,3][4,5,6][7,8,9]]按照123698745遍历，题目是m\*n的矩阵，这个写起来烦，我就讲了思路

3、驼峰化字符串，考虑\_、-、at符号等多种分隔符，可能有a--b这种连续分隔符、a-b\_c组合分隔符，实现的函数只接受一个参数

4、问最有成就感的东西，自己之前用纯原生写了个Datepicker插件，所以问了很多很多的细节，源代码在这：[https://github.com/pflhm2005/JS-components/blob/master/vanilla-picker/index.js](https://github.com/pflhm2005/JS-components/blob/master/vanilla-picker/index.js" \t "_blank)

5、懒加载讲下原理，写点伪代码。那几个top没分清，被嘲笑了……不过前几天练习写过一个vue指令，类似于懒加载，代码在这：[https://github.com/pflhm2005/JS-components/blob/master/lazyLoad.js](https://github.com/pflhm2005/JS-components/blob/master/vanilla-picker/index.js" \t "_blank)

6、sum(1)(2).value()、sum(1,2).value()都要输出3，就是个柯里化。不过手写真的不习惯，实现了，但是性能不好，面试官看我有点痛苦就没继续问优化了

最后问的一些工程问题我一个人都不会，真的是惨，比如部署cdn、webpack工程化的注意点、从项目角度来讲讲用户点击页面响应过慢的处理办法等等，因为以前呆的都是小公司，撸撸业务代码……

1. 任意元素垂直水平居中（多种方法）

<div class="container">Hello World！</div>

.container {

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%); // 自身宽度和高度的一半

border: 1px solid red;

}

.container {

width: 300px;

margin: 50vh auto 0;

transform: translateY(-50%);

border: 1px solid red;

}

1. bind实现

Function.prototype.mybind = function (context) {

if (typeof this !== "function") {

throw new Error(this + "is not a function");

}

var self = this;

var args = [];

for (var i = 1, len = arguments.length; i < len; i++) {

args.push(arguments[i]);

}

var fbound = function () {

var bindArgs = Array.prototype.slice.call(arguments);

self.apply(this instanceof self ? this : context, args.concat(bindArgs));

}

fbound.prototype = Object.create(self.prototype);

//返回的函数不仅要和 被调函数的函数体相同，也要继承人家的原型链

return fbound;

}

4.排序算法

5.说出你知道的所有vue组件通信相关方法

1. props和$emit   
父组件向子组件传递数据是通过prop传递的，子组件传递数据给父组件是通过$emit触发事件来做到的。

2.vuex处理组件之间的数据交互   
如果业务逻辑复杂，很多组件之间需要同时处理一些公共的数据，这个时候才有上面这一些方法可能不利于项目的维护，vuex的做法就是将这一些公共的数据抽离出来，然后其他组件就可以对这个公共数据进行读写操作，这样达到了[解耦](https://www.baidu.com/s?wd=%E8%A7%A3%E8%80%A6&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/zhoulu001/article/details/_blank)的目的。

3. v-model   
父组件通过v-model传递值给子组件时，会自动传递一个value的prop属性，在子组件中通过this.$emit(‘input’,val)自动修改v-model绑定的值。

4.中央事件总线   
上面两种方式处理的都是父子组件之间的数据传递，而如果两个组件不是父子关系呢？这种情况下可以使用中央事件总线的方式。新建一个Vue事件bus对象，然后通过bus.$emit触发事件，bus.$on监听触发的事件。

5、$parent和$children

1. 禁止浏览器缓存的三种方法

html只要加在头部就可以了.

<META HTTP-EQUIV="Pragma" CONTENT="no-cache">   
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">   
<META HTTP-EQUIV="Expires" CONTENT="0">

1. css3新特性，transform

CSS3新添加的Transform可以改变元素在可视化区域的坐标,以及形状,还有些3D形变.

旋转(rotate):

缩放(scale):

倾斜(skew):

平移(translate):

1. localStorage和sessionStorage存在哪里

webstorage是本地存储，存储在客户端，包括localStorage和sessionStorage。

localStorage生命周期是永久，这意味着除非用户显示在浏览器提供的UI上清除localStorage信息，否则这些信息将永远存在。存放数据大小为一般为5MB,而且它仅在客户端（即浏览器）中保存，不参与和服务器的通信。

sessionStorage仅在当前会话下有效，关闭页面或浏览器后被清除。存放数据大小为一般为5MB,而且它仅在客户端（即浏览器）中保存，不参与和服务器的通信。

11.js this相关（this，不想输出undefined）

12.vue生命周期都做了什么事情

13.new vue里面具体发生了什么

14.vue dep watcher observer一些细节，怎么发布订阅的

1. css中position哪些？  
   2.http和https区别  
   https的劣势  
   具体又问了对称加密和非对称加密区别  
   3.setTimeOut和promise区别  
   4.实现一定时间内事件只触发一次  
   5.两道编程  
   实现队列函数（先进先出），以实现一次100秒后打印出1，200秒后打印2，300秒后打印3这样  
   实现类似于模板字符串的功能

一面

写代码，查找两个链表的交叉节点

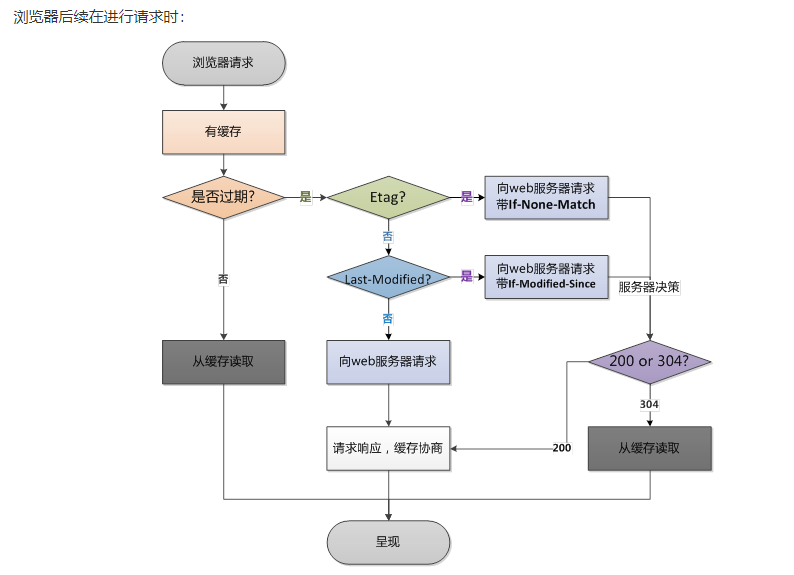
从输入url到显示页面都发生了什么

http缓存知道多少

**对于强制缓存，服务器通知浏览器一个缓存时间，在缓存时间内，下次请求，直接用缓存，不在时间内，执行比较缓存策略。  
对于比较缓存，将缓存信息中的Etag和Last-Modified通过请求发送给服务器，由服务器校验，返回304状态码时，浏览器直接使用缓存。**

**Etag：**  
服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。

**If-None-Match：**  
再次请求服务器时，通过此字段通知服务器客户段缓存数据的唯一标识。  
服务器收到请求后发现有头If-None-Match 则与被请求资源的唯一标识进行比对，  
不同，说明资源又被改动过，则响应整片资源内容，返回状态码200；  
相同，说明资源无新修改，则响应HTTP 304，告知浏览器继续使用所保存的cache。



强缓存和协商缓存的缓存，是从哪里拿的？

from memory cache：

from disk cache

资源本身大小数值（200服务器（报文大小），304本地）

打包工具都用什么，会写简单webpack配置吗

平时怎么学习

各种垂直居中方法

flex 有兼容性怎么办

从出需求到上线，中间经历了什么

箭头函数和普通函数的区别，什么时候有(没有)必要用箭头函数

二面

写代码，随机打乱一个数组

知道 indexOf 的实现方式吗？

TCP 三握四挥

http缓存， cache-control 的各个值

响应式和自适应的区别，如何做

transform、animation、translation 各种乱七八糟属性

redux 单向数据流

let const var

浮动是什么，举个栗子，为什么需要清除浮动，有没有不需要清除的情况

三面

写代码，实现懒加载，做节流处理

VDOM给开发者带来了什么

react diff 算法，某层节点从 ABCD 变成了 BDAC，说下整个 diff 流程，能写出代码吗？

https 加密数据

斐波那契数列，如何在使用递归的情况下，不爆栈，怎么优化

http2.0 多路复用使用的技术叫什么

http 强缓存是什么，流程、对应 http 头有哪些

redux中间件怎么写，怎么用

为什么会有单向数据流，解决了什么

Redux-saga 对比 redux-thunk 和 redux-promise 的优势，为什么要引一个这么大的包

https 数据传输加密逻辑

shouldComponentUpdate 是做什么的，在什么情况下会选择 return false，举个具体的例子

作者：Touma\_\_Kazusa  
链接：[https://www.nowcoder.com/discuss/109068](https://www.nowcoder.com/discuss/109068" \t "_blank)  
来源：牛客网

1. 说说函数节流

2. cookie、sessionStorage、LocalStorage区别，项目中的使用

3. new操作都做了什么

4. 如何实现链式调用

编程题：

1. 合并两个有序数组

2. 一棵二叉树找到是否有一条路径（从根节点到子节点），节点值的和为N（leetcode原题）

3. 忘了。。。

一面：

说说html5新增哪些标签

css垂直居中

vue的计算属性怎么实现的

vue的双向绑定实现原理

https过程

函数节流

清除浮动

ajax能不能封装一个类似axios的工具

……还有一些问题记不太清，一面50多分钟问了好多问题

二面：

说说各种协议吧，想说哪个说哪个（udp、tcp、http、https简单的介绍了一遍）

tcp为什么三次握手，两次有什么问题，怎么防止

tcp四次挥手

http缓存（介绍http状态码的时候自己加的戏）

给代码说结果（setTimeout，大家应该都懂）

算法题，拿着面试官电脑现场撸= =

有什么问题要问

作者：cupMen  
链接：[https://www.nowcoder.com/discuss/92523](https://www.nowcoder.com/discuss/92523" \t "_blank)  
来源：牛客网

Script标签异步特性  
Web动画的实现方式  
跨域相关  
Http options请求  
Http缓存控制，包括后端，结合304状态  
算法，数组去重  
Cookie的属性  
Xss,csrf  
正向代理，反向代理  
写一个router class  
语义化的理解  
异步

作者：信小呆本呆  
链接：[https://www.nowcoder.com/discuss/128106](https://www.nowcoder.com/discuss/128106" \t "_blank)  
来源：牛客网

1. 给你一个数字n(n < 1e9), 再给你一个数字k(k < n), 要求你找到1, 2, 3, ... n按照字典序排序后, 第k大的数字;

如, n = 15, k = 7;那1 ~ 15按照字典序排序为: 1, 10, 11, 12, 13, 14, 15, 2, 3, 4, 5, 6, 7, 8, 9;则答案为15;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | function get(n, k) {      let arr = [];      for (let i = 1; i <= n; i++) {          arr.push(i);      }      arr.sort(); //默认按字典序排序      return arr[k - 1];  } |

2. promise封装ajax请求，如何判断ajax超时

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | let ajax = (obj) => {      return new Promise((resolve, reject) => {          let method = obj.method || 'GET';          let xhr = null;          if (window.XMLHttpRequest) {              xhr = new XMLHttpRequest();          } else {              xhr = new ActiveXObject('Microsoft.XMLHTTP');          }          xhr.onReadyStateChange = () => {              if (xhr.readyState == 4) {                  if (xhr.status >= 200 && xhr.status < 300 || xhr.status == 304) {                      resolve(xhr.responseText);                  } else {                      reject(xhr.statusText);                  }              }          }          if (method == 'POST') {              xhr.open('POST', obj.url, true);              xhr.responseType = "json";              xhr.setRequestHeader("Accept", "application/json");              xhr.send(obj.data);          } else {              let query = '';              for (let key in obj.data) {                  query += '&' + encodeURIComponent(key) + "=" + encodeURIComponent(obj.data[key]);              }              query.substring(1);              xhr.open('GET', obj.url + '?' + query, true);              xhr.send();          }      })  } |

3. ES6的模块化标准相对AMD和CommonJS最本质的区别是什么

CommonJS和AMD是运行时加载，ES6模块是编译时加载

4. 设计一个简单的任务队列, 要求分别在 1,3,4 秒后打印出 "1", "2", "3"

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | class Queue {      constructor() {          this.queue = [];          this.time = 0;      }      task(t, fn) {          this.time += t;          this.queue.push([fn, this.time]);          return this;      }      start() {          this.queue.forEach((item) => {              setTimeout(() => {                  item[0]();              }, item[1]);          })      }  }    new Queue()      .task(1000, () => {          console.log(1)      })      .task(2000, () => {          console.log(2)      })      .task(1000, () => {          console.log(3)      })      .start() |

5. 一行代码判断一个字符串是否是回文字符串，回文字符串是对称字符串的形式，例如：did，eve, dad, level

|  |  |
| --- | --- |
| 1 | str.split('').reverse().join('') == str; //大佬们有其他的方法吗？ |

6.  script延迟加载方法 //不阻塞DOM渲染

//这个有一张简洁明了的图：[https://segmentfault.com/q/1010000000640869](https://segmentfault.com/q/1010000000640869" \t "_blank)

1. async：异步加载，加载完成后立即执行
2. defer：异步加载，脚本在元素解析完成之后，DOMContentLoaded 事件触发之前执行
3. **同时使用defer和async,如果浏览器两个属性都支持，则会忽视defer属性,按照async执行**

7. 求一个二叉树所有根节点到叶子节点的路径所组成数字的和

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | function sumTree(root,sum=0) {      if(!root) {          return 0;      }      sum = sum\*10+root.val;      if(!root.left && !root.right) {          return sum;      }      return sumTree(root.left,sum) + sumTree(root.right,sum);  } |

8. requestAnimationFrame理解：[http://web.jobbole.com/91578/](http://web.jobbole.com/91578/" \t "_blank)

9. 什么是类数组对象？

JS中，对于一个普通的对象来说，如果它的所有键名均为正整数，同时拥有length属性，则为“类数组对象”。例如：(函数中) arguments / nodeList

类数组 => 数组：

|  |  |
| --- | --- |
| 1  2  3 | [].slice.call(arguments);  [].splice.call(arguments,0);  [].concat.apply([],arguments); |

10.  0.1 + 0.2 > 0.3  why?

0.1和0.2都是近似表示的，在他们相加的时候，两个近似值进行了计算，导致最后得到的值是0.30000000000000004，此时对于JS来说，其不够近似于0.3

[https://segmentfault.com/a/1190000012175422](https://segmentfault.com/a/1190000012175422" \t "_blank)

11.  实现一个Event类，具有on\off\once\trigger方法

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | class Event {      constructor() {          this.tasks = {};      }      on(name,cb) {          if(!this.tasks[name]) {              this.tasks[name] = [];          }          this.tasks[name].push(cb);          return this;      }      off(name,cb) {          let item = this.tasks[name];          if(item) {              for(let i=0;i<item.length;i++) {                  if(item[i] == cb) {                      item.splice(i,1);                      break;                  }              }          }          return this;      }      once(name,cb) {          if(!this.tasks[name]) {              this.tasks[name] = [];          }          cb.tag = 'once';          this.tasks[name].push(cb);          return this;      }      trigger(name) {          let tasklist = this.tasks[name],              args = [].slice.call(arguments,1);          if(tasklist) {              for(let i=0;i<tasklist.length;i++) {                  tasklist[i].apply(this,args);                  if(tasklist[i].tag && tasklist[i].tag == 'once') {                      tasklist.splice(i,1);                      i--;                  }              }          }          return this;      }  }    let myEvent = new Event();  myEvent.on('console',() => {      console.log('on1');  }).once('console',() => {      console.log('once');  }).on('console',() => {      console.log('on2');  }).trigger('console').trigger('console');  //on1 once on2 on1 on2 |

12. CSS实现三角形

|  |  |
| --- | --- |
| 1  2 | div {  width: 0;  height: 0;  border: 100px solid black;  border-top-color: transparent;  border-right-color: transparent;  } |

13. 绝对定位和transform:translate的区别

绝对定位会引起页面重排（relayout）和重绘（repaint）

translate平移只会触发重绘（repaint）

14. 进程间通信的方式

1. 管道通信
2. 信号量机制
3. socket
4. 消息队列
5. socket套接字

15. node加载第三方模块的原理

16. git rebase和merge区别

* git merge：合并分支，并保留修改内容的历史记录
* git rebase：合并分支，在原有的历史记录后添加差异

17. React生命周期

略...

18. 归并排序

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | function mergeSort(arr) {//分解      if (arr.length <= 1) {          return arr;      }      let mid = Math.floor(arr.length / 2);      return merge(mergeSort(arr.slice(0, mid)), mergeSort(arr.slice(mid)));  }    function merge(left, right) {//合并      let rs = [];      while (left.length && right.length) {          if (left[0] < right[0]) {              rs.push(left.shift());          } else {              rs.push(right.shift());          }      }      return rs.concat(left, right);  } |