1. setTimeout(fn,0)会延迟执行吗

会延迟 js是单线程的，所有任务按照任务队列（eventLoop）顺序执行，任务队列分为宏任务队列和微任务队列 eventLoop的执行顺序:script主线程任务（属于宏任务）-> 微任务->下一个宏任务 setTimeout属于宏任务 setTimeout(fn,0)的含义是:尽可能早的执行，html5标准规定了时间的最小值不得低于4毫秒，如果低于这个值，则自动增加

2. 浏览器事件循环

告诉我们js代码的执行顺序，为了解决js单线程运行不会阻塞的机制，异步的原理 js有一个主线程和调用栈，所有的任务都会被放到调用栈中等待主线程执行。 js任务分为同步任务和异步任务： 同步任务：在调用栈中按照顺序排队等待主线程执行 异步任务：有了结果之后将注册的回调函数添加到任务队列，调用栈被清空之后被读取到调用栈被主线程执行

js单线程的任务可分为宏任务和微任务 宏任务：script代码，setTimeout，setInterval，setImmediate I/O UI rendering http回调 事件

微任务：promise await 执行主线script代码 -> 执行所有的微任务 -> 执行一个宏任务

浏览器视图渲染发生在本轮事件循环的微任务队列被执行完之后，也就是执行任务的耗时会影响视图渲染的时机，所以要让用户觉得流畅，单个宏任务以及它相关的微任务最好能在16.7ms内完成

但也不是每轮事件循环都会执行视图更新，浏览器有自己的优化策略，例如把前几次的视图更新累积到一起重绘，重绘之前通知requestAnimationFrame执行回调函数，也就是说requestAnimationFrame的回调的执行时机是在一次或多次事件循环的UI render阶段

3. js脚本加载问题 async，defer

async异步的：js加载不阻塞html解析，js加载完之后，html暂停，js执行，执行完之后继续加载html defer延迟：js加载，加载完之后等待html加载，html解析完之后继续执行js

script一直阻塞

async可能阻塞

defer不阻塞

- script 会阻塞

浏览器在解析html的时候，如果遇到一个没有任何属性的script标签就会暂停解析，先发送网络请求获取该js脚本的代码内容，然后让js引擎执行该代码，当代码执行完毕后恢复解析html。

- async script

async表示异步 可能阻塞也可能不阻塞 当浏览器遇到带async属性的script时，请求该脚本的网络请求是异步的，不会阻塞浏览器解析html，一旦网络请求回来之后，如果此时html还没有解析完，浏览器会暂停解析，js引擎执行代码，执行完之后再进行解析

如果js脚本请求回来之后，html已经解析完毕了，那么直接执行js代码。 所以async是不可控的，因为执行时间不确定，如果在js异步脚本中获取某个DOM元素，有可能获取到也有可能获取不到。如果存在多个async，他们之间的执行顺序也不确定，完全取决于网络传输结果，谁先到先执行谁。

- defer script 不阻塞

defer表示延迟 当浏览器遇到带有defer属性的script时，获取该脚本的网络请求也是异步的，不会阻塞浏览器解析html，一旦网络请求回来之后，如果此时html还没有解析完，浏览器不会暂停解析，而是等html解析完成后执行js代码

如果存在多个defer script标签，浏览器ie9以下除外，会保证他们按照在html中出现的顺序执行，不会破坏js脚本之间的依赖关系。

- 如果当前js文件依赖其他脚本和DOM结果，使用defer，defer能保证顺序，保证DOM，不会破坏依赖关系
- 如果当前js文件和DOM，其他脚本文件依赖性不强，使用async，因为async会阻塞html解析，请求完js后需要先执行，不能保证执行顺序，执行顺序取决于网络传输结果，谁先到谁执行。

4. 判断一个对象是不是空对象

Object.keys(obj).length

5. script 引入的外部js先加载还是onload先执行为什么

onload事件是在外部js被加载完并且执行完成之后才被触发的

window.onload是 html中所有的文档都加载完毕后，执行里面的内容，因此加了window.onload的js不论在页面中哪个位置引入，都不会出现html找不到的情况。

6. 数组方法

改变数组： push pop unshift shift sort reverse splice

其他 map 得到一个新数组 filter筛选 reduce 把一个数组中的一堆值计算成一个值 let result = rest.reduce(function(val, item, index, origin) { return val + item },0) // 0初始值 如果没有给初始值 则初始值为第一个元素的值 item从第二个元素开始

Array.isArray()

findindex

indexof 如果存在就返回当前下标，如果不存在就返回-1

7. 类数组转数组

Array.from Array.prototype.slice.call(arguments) ...

8. Array.from, [...likeArr],Array.prototype.slice.call(arr)

Array.from()方法从一个类似数组或可迭代对象创建一个新的，浅拷贝的数组实例

Array.from还可以接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，处理后的值放入返回的数组。 Array.from([1, 2, 3], (x) => x * x)// [1, 4, 9]

[...likeArr]扩展运算符背后调用遍历器接口，如果一个对象没有遍历器接口，无法转换

let items = Array.of(1, 2); console.log(items.length); // 2 console.log(items[0]); // 1 console.log(items[1]); // 2 items = Array.of(2); console.log(items.length); // 1 console.log(items[0]); // 2