书：

一、引入css样式的方式有哪几种？

①内联方式

②直接写在style里面

③link引入

④@import引入

@import会在整个网页加载完成之后再渲染样式，导致事先显示无样式页面，闪烁以后再显示网页样式

二、position的所有值

①absolute ②fixed ③relative ④static默认 ⑤inherit继承父元素

三、box-sizing属性

①content-box border和padding不计算进width的宽度

②padding-box padding计算进width的宽度

③border-box padding和border都不计算进width

ie8+浏览器支持content-box和border-box

火狐支持三个

四、三列布局的css，左右两侧宽固定，中间自适应

中间元素设置margin-left=左边元素宽和margin-right=右边元素宽度

中间的元素需要放在浮动两个元素后面写，否则右浮动元素会被挤下去

五、box-shadow写单边阴影

box-shadow: h-shadow   v-shadow   blur   spread   color   inset;

(box-shadow: 水平阴影 垂直阴影 模糊距离 阴影大小 阴影颜色  内部阴影;)

单边阴影需要舍弃模糊距离，元素有多大，阴影色块就位于下方有多大，h-shadow仅是将他移动而已

六、var a = {n:4};

var b = a;

b.x = a = {n: 10};

console.log(a.x);//undefined

console.log(b.x);//{n:10}

七、cookie、sessionStorage、localStorage的区别

①kookie：储存在本地终端上的数据，主要是利用session判断客户登录状态

②sessionStorage 储存在客户端，浏览器器窗口关闭后。数据删除

③localstorage 储存在客户端，没有时间限制，长久储存

kookie数据始终在同源的http请求中携带，固数据不会超过4k

另外两个不会把数据发送给服务端，仅保存在本地，可保存容量5M或更大

sessionStorage不在不同的浏览器窗口中共享，即使是同一个页面，另外两个在同源窗口都是共享的

八、如何让一个元素相对于父元素居中（考虑到块级，内联，不定宽高）

水平居中：父元素display：flex just-content:space-around

垂直居中：父元素display：flex flex-flow：column just-content：space-around

九、如何利用rem写自适应网页

rem是文字的单位，可以根据屏幕大小自适应

十、call/apply/bind的区别

call和apply区别在于call传参时单个跟参数， apply传参可将多个参数放在一个数组里直接传

bind和其余两个的区别，bind是复制一个函数副本，执行副本函数，而另外两个是立即执行当前函数

一、伪数组、类数组转为数组

①遍历类数组放入新数组中

②arr=Array.Prototype.slice.call(类数组) 让类数组强制调用数组的方法

③arr\_\_proto\_\_=Array.prototype ie是没有提供\_\_proto\_\_

二、判断一个对象是否为数组

①判断原型对象

obj.\_\_proto\_\_==Array.prototype

Array.protetype.isPrototypeOf(obj) 数组的原型对象是否为obj原型对象

②判断构造函数

obj.constractor==Array obj的构造函数等于数组构造函数

obj instanceof Array obj是数组的一个实例吗

以上两种不严谨，可能存在改\_\_proto\_\_属性

③判断隐藏的class属性

Object.prototype.toString().call(obj) 强行调用最顶级的tostring方法 object object 对象 object array 数组

④Array.isArray(obj)

三、promise用法

new promise （ function（open，err）{ } ）.then/.catch 创建对象的同时，自动执行内部的函数

如果open则.then 如果err则catch

Es7

one two three都是promise的实例

(async function(){ try{ await one;await two ;await three } }).catch(err){ }

promise.all([ one ,two,three ])

四、原生实现事件委托

①直接在标签中onclick=“处理函数”

②document获取元素.onclick=处理函数 仅绑定一个

③document获取元素.addEventListener(“click”，处理函数) 可添加多个

五、状态码

2开头 请求成功，并处理了请求的状态

3 重定向

4 请求错误 妨碍了服务器处理

5 服务器错误

六、get post的区别

①传递数据量 post没限制 get 1kb

②get在浏览器回退的时候是无害的，post会再次提交

③get比post更不安全，参数会直接暴露在URL上

④get通过url传参，post通过request body传参

⑤对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

七、页面性能优化

①尽量减少http的请求次数

将所有的背景图片放到一个图片文件中，利用background来显示不同的背景图片

②使用ajax实现异步更新

八、vue的生命周期

①beforeCreate() el:undefined data:undefined

②created() el:undefined data:{}

③beforeMount el:空的dom对象 data:{}

④mounted el:完整的dom对象 data:{}

⑤beforeUpdate

⑥undate() data里的值发生变化是触发

⑦beforeDistroy 摧毁前 提示你确认要删除xx

⑧distroyed 删除成功

九、this指向问题

this指向在函数创建时时决定不了的，this永远指向最后调用他的对象，也就是看他执行的时候是谁调用的

①如果这个函数被上一级对象所调用，那么this就指向上一级对象

②如果函数调用时是包含多个对象的，this也只是指向他的上一级对象

③构造函数中的this会因为关键词new而改变this的指向 this会指向实例

③另外我们也可以自行改变this的指向，利用call apply bind的方法

④如果一个函数中有return，假设return返回的是一个对象，那么this就指向这个对象；如果返回的不是一个对象，那么就指向函数的实例；注意：函数就是对象；null也是对象，但是null比较特殊，this不指向null

十、new的作用

①创建一个空的新对象

②让子对象继承构造函数的原型对象

③用新对象调用构造函数 这时如果构造函数中有this，自然就指向新对象

一、for循环的各种情况

①for(var i=0;i<=arr.length;i++) 最灵活

②arr.forEach((elem, I ,arr)=>{}) 无法控制遍历的顺序和步伐

.map 每个元素执行相同的操作后，放入到新数组中

.filter 将符合条件的元素放入新数组中

.reduce((prev,elem,I,arr)=>{ },初始值) 每次计算的值保存在prev中

③for(var elem of arr) 会以此获得元素值，保存在elem中，无法获得下标

④for(var key in arr) 专门遍历自定义下标名的关联数组和对象

二、js造成内存泄露的情况

内存泄露：已分配的堆内存由于某种原因程序未释放或无法释放，造成的系统内存浪费

①闭包 外层函数的局部作用域无法被释放

②意外全局变量 定义在函数内的全局变量，函数调用后，变量依然存在

解决：可用‘user strict’

③定时器 当不需要定时器时，定时器并没有被clear，里面的回调函数以及所依赖的变量都不能被回收

解决：clearTimeout(\*\*\*) clearInterval(\*\*\*)

④死循环

三、ES5新特性

① 严格模式 user strict

禁止给未声明的变量赋值

静默失败升级为错误 如修改只读属性，不报错也为成功

this不指向window而是指向undefined

②保护对象

value：属性值

writable：是否可修改value

enumerable：是否可遍历

configurable：是否可删除该属性 不可逆

③修改this指向的三种方法

④遍历API

forEach 每个元素执行相同的操作后再返回到原数组中

map 每个元素执行相同的操作后，放入到新数组中

filter 将符合条件的元素放入新数组中

reduce((prev,elem,I,arr)=>{ },初始值) 每次计算的值保存在prev中

四、ES6新特性

①遍历API forOf

②let 避免声明提前 添加块级作用域

③spread 打散数组专用function(…[ ])

④箭头函数 this的指向内外相同

⑤解构 利用关联数组或对象做形参

⑥Promise 保证多个异步函数按顺序执行 采用new Promise封装异步任务open则执行then err则执行catch

⑦class 将构造函数和他的原型对象封装为一个整体

⑧模板字符串

五、懒加载

原理：将要页面中所有图片一使用占位图进行占位，把正真的图片地址保存在一个自定义属性data-url中，根据滚动事件判断图片是否出现在可视区，如出现则将data-url里的值给src

①元素到文档顶部的高度：elem.offsetTop

②页面滚动的距离：document.body.scrollTop

③可视窗口高度：document.body.clientHeight

④滚动事件：window.onscroll

六、清除浮动

①：after IE6-7不支持

②父元素设置overflow：auto

③添加子元素，设置属性clear：both

七、let、const、var的区别？

①const 常量，不可修改，且必须赋值

②var 变量，可修改，未赋值未undefined

③let 块级作用域，可修改

八、闭包

定义：能够读取其他函数内部变量的函数

原因：外层函数调用以后，局部作用域无法释放

优缺：重用变量，又不造成全局污染

九、打开一个网页，中间的过程

①向DNS服务器获取ip地址并返回给浏览器

②浏览器通过ip地址向web服务器发请求

③服务器获取所需的问件和数据，响应給浏览器

十、typeof返回的类型有哪些

string、number、boolearn、undefined、object、function

一、nodeJs的优缺点

NodeJS设计思想中以事件驱动，异步，非阻塞I/O密集型为核心，非常适合高并发请求，编写方式与js一致，因此客户端和服务器可用同一种语言。

缺点：不适合CPU密集型任务的开发。

①阻塞和非阻塞

进程访问的数据如果尚未就绪，进程是否需要等待

②同步和异步

线程发送一个数据请求，线程是否需要等待

③CPU密集型：一些进程绝大多数时间是在计算上

I/O密集型：大多数时间是花在请求数据和响应数据上

二、js的本地对象、内置对象、宿主对象

本地对象：Obj、的点点滴滴多多多ct Date Array 等官方定义好的方法

内置对象：Global Math

宿主对象：所有的Dom和Bom对象

三、影藏元素的方式

①display:none

搜索引擎可能会认为被隐藏的文字属于垃圾信息而被忽略

屏幕阅读器会忽略隐藏的文字

②visibility:hidden opcity:0

占据页面宽度，opcity可添加事件

③overflow:hidden 设置width和height都为0

屏幕阅读器可读取隐藏的内容

四、给Array添加一个数组去重的方法

①for(var i=0;i<arr.length;i++){

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

if(arr[i]==arr[j]){

arr.splice(j,1)

j--

}

}

}

②利用indexof去重 array[i].indexOf(arr[i])===-1,则push进array

③利用对象属性去重 obj[arr[i]]==undefined 则obj[arr[i]]=1；array.push([arr[i]])

④利用数组的方法filter arr.filter(function(item ,index,arr){ return arr.indexOf(item,0)===index })

五、对象的合并

var o1 = { a: 1 };

var o2 = { b: 2 };

var o3 = { c: 3 };

var obj = Object.assign(o1, o2, o3);

是给o1对象末尾添加o2和o3对象

六、获取指定范围内的随机数

Math.random()\*(max-min)+min

七、阻止事件的冒泡和默认行为

e.preventDefault()

e.stopPropregation()

八、JSON

json是一种用于数据交换的文本格式

json.stringify() json.parse()

json格式

数字、字符串、逻辑值(true,false)、数组、对象、null

eval( ‘(’+str+’)’ )代表直接执行str字符串，返回计算后的值

不严谨，eval不会判断str的格式

九、Ajax

①function createXhr(){

if(window.XMLHttpRequest){ xhr=new XMLHttpRequest()}

else{xhr=new ActiveXObject(“Microsoft.XMLHttp”)

②为xhr添加监听事件xhr.onreadystatechange=

③发送请求

get:xhr.open(“get”,”/完整路径”,true)

post:xhr.open(“post,”、路径”，true)

设置响应头：xhr.setRequestHeader()

④发送完成

get:xhr.send()

post:xhr.send(查询字符串)

十、异步是如何实现的

①JavaScript是单线程的，负责解释和执行JavaScript代码的线程只有一个，主线程

另外还有一些像处理ajax,定时器这些异步请求的工作线程

②主线程发起一个异步请求，相应的工作线程接收请求，并告知主线程已收到

③主线程可继续执行后面的代码，同时工作线程也在执行异步任务

④工作线程完成工作后，通知主线程

⑤主线程收到通知后，接收结果并处理

一、vue组件间的通讯

父>子：是在父组件内给子组件标签上 :value=值 子组件内 props:[“value”]

子>父：在父组件内给子组件标签上 自定义事件名=事件处理函数 ；另外父组件内定义事件处理函数，根据参数做处理 子组件：this.$emit(父组件的自定义事件名，参数)

兄弟：首先创建一个bus.js 里面引入vue 将new vue()抛出 兄弟组件import Bus from ‘bus.js’

组件A Bus.$emit(“事件名”，参数) 发送

组件B Bus.on(“事件名”，(data)=>{ data }) 接收

二、路由传参

接收组件需要path:/:id

router-link to

this.$router.push

三、SPA 单页面应用

注：vue-router是专门用于实现单页面应用的组件

相对于多页面：

①请求一次拿到所有内容

②跳转时只替换局部的dom树节点对象

③共有资源只下载一次

④很容易实现页面切换动画

问题：首屏加载慢

解决：利用懒加载

原理：定义一个完整的HTML文件，里面利用router-view标签占位

定义多个template，并配置路由字典

利用url中的路径，利用路由字典查找对应的template替换占位标签

四、Vuex

state：数据储存

mutations：定义修改数据的方法

getters：获取数据的方法 function（state）return state.count

修改调用：this.$store.commit(‘方法名’，第二参数)

获取调用：this.$store.getters.方法名

五、websocket

可以看做是http协议的一个补充，实现了服务端主动向客户端发起信息

使用：

①new一个websocket对象ws = new WebSocket("ws://

②ws.onopen 连接成功触发事件

③.send() 发送消息

④.onmessage 接收消息

六、跨越

cors: origin：url Crendentails:true

axios请求携带kookei withCrendentails

jsonp:利用script的src属性

七、文档片段

①frag=document.createDocumentFragment()

②frag.appendChild(child)

①document.createElement (“a”)

八、jQuery的事件委托方法bind 、live、delegate、on之间有什么区别？

①bind：只能给已经存在的元素绑定事件，不能给未来新增的元素绑定时间

②live：新增元素不需要再次绑定事件，但是live只能放在直接元素的后面如$(body).find(“ul”).live就不行

③delegate：绑定在就近的父元素上，可以动态添加到新元素上，冒泡时间更快，性能优于live

④on:推荐使用，整合以上三种方法的绑定机制

九、正则表达式

①reg=new RegExp(“表达式”.”ig”)

②reg.test(str) 验证

③reg.exec(str) 内容加下标，一次只返回一个，找不到返回null 其中有lastIndex属性

十、重载

JS不支持重载，因为多个同名函数存在，之前的会被覆盖

但是我们可以通过一个函数传入参数的个数或者性质来实现重载

注：重写就是子对象本地定义同名函数，来覆盖父对象继承给他的这个函数

一、