面经总结：

HTTP：

1. HTTP1.0
   1. 浏览器与服务器之间只保持短暂的连接。浏览器的每次请求都需要与服务器建立一个TCP连接，请求处理后立刻断开；
2. HTTP1.1
   1. 支持持久连接，在一个TCP连接上可以进行多次HTTP的请求和响应，减少了建立和关闭连接的开销。
   2. 允许客户端无需等待上次请求结果返回，就可以进行下一次请求；
   3. request和response头中可能出现一个头connection，含义是浏览器和服务器在通信时遇到长链接怎么如何处理；
      1. 如果是close，正在使用的tcp链接在请求处理结束后断开；
      2. Keep-alive，客户端通知服务器返回本次结果后保持连接；
3. 介绍Http2，优点和存在什么坑或问题（必考）
   1. 允许多路复用；多路复用允许通过单一的HTTP2连接发起多重的请求-响应消息
   2. 二进制分帧
   3. 首部压缩；HPACK算法
   4. 服务器端推送；
4. HTTP报文的请求和返回会有几个部分（请求行、请求头、请求体）；每部分具体都有什么（常见的请求头）
   1. 请求报文
      1. 请求行
         1. 包括请求方法、URL、HTTP版本
      2. 请求头
         1. User-Agent：产生请求的浏览器类型
         2. Accept：客户端可识别的数据类型列表
         3. Host：主机地址
      3. 请求数据
         1. post方法，会把数据会以key-value的方式发送
      4. 空行
         1. 回车和换行符
   2. 响应报文
      1. 状态行
         1. 协议版本、状态码以及描述
      2. 响应报头
         1. 用于描述服务器的基本信息
         2. Allow服务器支持那些请求方式
         3. Expires 设置过期时间
      3. 响应正文，消息体
5. HTTP状态码：
   1. 1xx, 提示信息，表示信息已接收，继续处理
   2. 2xx，成功 表示信息已经被成功接收、理解、处理
   3. 3xx，重定向 要完成请求必须进行更进一步的操作
   4. 4xx，客户端错误 请求有语法错误或者请求无法实现
   5. 5xx，服务器端错误 服务器未能实现合法的请求
6. 介绍HTTPS；HTTP和HTTPS的区别（必考）
   1. HTTPS是更为安全的HTTP通道，加入了SSL层对HTTP的传输内容进行了加密。HTTPS在传输数据之前需要客户端和服务器进行一次握手，确立双方的加密密钥；
7. HTTPS加密过程
   1. 加密和解密用同一个密钥的方式叫做对称加密-->破解了一个用户的密钥，所有用户的密钥都被得到；
   2. 所以采用非对称加密，加密过程：
      1. 客户端发送HTTPS请求
      2. 服务器将证书（公钥）发送给客户端
      3. 客户端解析证书，并且验证公钥是否有效。如果有效，生成一个随机数，用公钥加密，将信息传送给服务器；
      4. 服务器接收到之后，用私钥解密后，得到随机数。
      5. 服务器和客户端用这个生成的随机数进行加密通信
8. HTTP缓存控制（强缓存、弱缓存）；缓存相关的HTTP请求头（必考）
   1. 强缓存：在本地缓存中查找该资源，如果其他限制也没有问题。返回200，直接使用强缓存；
   2. 弱缓存： 在本地缓存中找到该资源，发送给一个http请求到服务器，如果服务器判断该资源没有被修改过，返回304，使用本地缓存中的资源；
   3. 如果该资源被修改过，或者本地缓存中没有找到该缓存资源，服务器则返回该资源的数据。
9. GET和POST的区别
   1. Get用于向服务器请求制定资源，并且参数会出现在URL上程度的限制；
   2. POST用于向服务器提交一定的数据进行处理；
   3. Get提交数据会出现在URL上，不同的浏览器对URL的长度有不同；
   4. POST安全性更高
10. 输入url到页面加载全过程（必考）
    1. 输入URL回车
    2. 本地缓存查找当前URL是否存在缓存，并比较缓存是否过期
    3. DNS解析URL对应的IP
    4. 根据IP建立TCP连接（三次握手）
    5. HTTP发送请求
    6. 服务器处理请求，浏览器接受HTTP响应
    7. 渲染页面，构建DOM树
    8. 关闭TCP连接
11. DNS域名解析
    1. 检查本地hosts文件是否有该网址映射关系
    2. 查找本地DNS解析器缓存
    3. 查找本地DNS服务器
    4. 迭代查询
12. 对跨域的了解，跨域怎么解决（必考）
    1. 跨域就是浏览器在一个域名的网页请求另一个网页的数据（协议、主机好、端口号不同）
    2. Cors解决跨域
13. cors的返回头、cors预请求，什么时候会出发预请求（必考）
    1. cors：跨域资源共享，它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而克服Ajax只能同源使用的限制；
       1. 简单请求：
          1. 请求方式是HEAD、GET、POST之一；
          2. 或者HTTP的头信息不超过五种字段：Accept、Accept-language、content-language、Last-Event-ID、content-type;
       2. 非简单请求
    2. 对于简单请求，浏览器直接发出cors请求。
       1. 具体来说就是在头信息中添加一个origin字段，用来说明本次请求来自哪个源；
       2. 返回头：
          1. Access-Control-Allow-Origin(必须)，它的值又可能是请求里origin的值，也有可能是\*，代表接受任意域名的请求；
          2. Access-Control-Allow-Credentials（可选），布尔值，表示是否允许发送cookie
          3. Access-Control-Expose-Headers（可选）
    3. 对于非简单请求，在正式通信之前会先发出一次预请求
       1. 浏览器先向服务器询问，当前的网页是否在该服务器的许可名单中，如果得到肯定回复，再发送XMLHttpRequest请求；
       2. 预请求的头信息包括origin、Access-Control-Request-Method、Access-Control-Request-Headers
       3. 返回头Access-Control-Allow-Origin字段，如果返回的没有任何与cors有关的头，则失败；
       4. 其他头与简单请求返回头类似
14. CSRF跨站请求伪造
    1. 攻击者盗用用户身份，发送恶意请求
15. tcp的长连接和http2多路复用相关
    1. 长连接指的是在一个TCP连接上可以连续发送多个数据包；短连接是指通信双方有数据交互时，就建立一个TCP连接，数据发送完成后，就断开TCP连接；
    2. http2多路复用允许通过单一的HTTP连接，发送多重http请求；
16. tcp拥塞控制和流量控制
    1. 拥塞控制包括四部分：慢启动、拥塞避免、快速重传、快速恢复
    2. 流量控制：是通过大小可变的滑动窗口实现的
17. 三次握手四次挥手，为什么三次和四次
    1. 三次握手：需要让双方确定自己能收到对方的消息，而且自己的消息也能被对方收到；SYN、seq、ACK，ack
       1. 客户端设置SYN = 1，生成随机数seq发送给服务器
       2. 服务器将SYN = 1，ACK=1，ack = seq+1， 生成随机数seq返回给客户端，请求连接
       3. 返回ACK = 1，ack = seq+1；
    2. 四次挥手
       1. 客户端发送FIN，用来请求关闭连接，进入FIN\_WAIT\_1状态；
       2. 服务器收到FIN之后，发送ACK，进入CLOSE\_WAIT状态；
       3. 服务器发送一个FIN请求关闭连接，进入LAST\_ACK状态；
       4. 客户端收到FIN后，客户端进入到TIME\_WAIT状态，接着发送ACK到服务器，确认序号是收到序号加1，服务器进入CLOSE状态
18. 前端性能优化（必考）

CSS：

1. 盒模型
   1. 标准盒模型
   2. IE盒模型
2. position的值，都是根据什么定位（注意下还有sticky）
   1. absolute
      1. 相对于第一个非static的父元素定位
   2. relative
      1. 相对于元素本来的位置定位
   3. fixed
      1. 相对于浏览器窗口定位
   4. Static
      1. 默认定位，元素处于正常文档流中，会忽略top、bottom、等属性
   5. Inherit
      1. 继承定位
   6. sticky
3. 栅格布局的原理
4. 垂直居中布局
   1. Flex布局，align-items：center；justify-content:center;
5. 上中下布局，中间自适应
   1. Flex布局，flex-direction:垂直排序；中间块flex-grow：1；
6. 左中右布局，中间自适应
   1. 左右float
   2. Flex
7. 重绘和重排：
   1. 当DOM的变化改变影响了元素的几何属性，浏览器需要重新计算元素的几何属性，其他元素的几何属性和位置也收到影响，浏览器会让渲染树中收到影响的部分失效，重新构造渲染树叫做重排；重排发生后，浏览器会重新绘制受影响的部分到屏幕，这个过程叫做重绘；
   2. 引起重绘的行为(重排)
      1. 添加或者删除可见的DOM元素
      2. 元素位置改变
      3. 元素尺寸改变
      4. 元素内容改变
      5. 浏览器窗口尺寸改变
   3. 如何减少重绘
      1. css transform
      2. css opacity
      3. css filter
8. 动画相关属性
   1. transition: 需要触发一个事件才能改变属性
   2. animation：不需要触发任何事件的情况下就能改变元素
9. 移动端适配方案（必考）
   1. @media screen and(min-width:)查询设备的宽度执行不同的css代码
      1. 优点：方法简单成本低，特别是对移动和PC维护同一套代码的时候；
      2. 缺点：代码量大，影响各自的移动端和PC端各自的交互；
   2. flex弹性布局
      1. viewpoint固定
      2. 高度定死，宽度自适应
   3. rem+viewpoint
   4. rem
10. 移动端适配1px的问题
11. lineheight属性1.5和150%区别
    1. 区别在与子元素继承时：
       1. 父元素设置line-height：1.5；会直接继承给子元素，子元素根据自己的font-size去计算自己的line-height；
       2. 父元素设置line-height：150%，会父元素计算好了line-height值，然后再把这个计算值给子元素继承，子元素的font-size，与继承的line-height值没有影响
12. px,em,rem
    1. px,像素，相对长度单位。像素px是相对于显示屏分辨率而言的；
    2. em是相对长度单位，是相对于当前对象内文本的字体尺寸；
    3. rem相对长度单位，相对于html根元素。
    4. 对于只需要适配少部分手机设备，且分辨率对页面影响不大的，使用px即可；对于需要适配各种移动设备，使用rem。例如只需要适配iphone和ipad等分辨率差别大的设备；

JS：

1. 判断数组方法

方法一：

a.isArray();

方法二：

var a = [];

a instanceof Array;

方法三：

prototype + toString +call();

Object.prototype.toString.call(a).indexOf();

方法四：

isPrototypeOf()函数，用于指示一个对象是否存在于一个对象的原型链中

Array.isPrototypeOf(a)；

1. Object.assign和Object.create相关
2. 深拷贝和浅拷贝

假设B复制了A，如果修改A，B也同时改变，则说明是浅拷贝；如果修改A，B并不会改变，则是深拷贝。

深浅拷贝的主要区别就是：复制的是引用(地址)还是复制的是实例。

1. let、const和var的区别

let的作用域是块级作用域，不存在变量声明提前，不能重复定义，存在暂时性死区（即就是，不论在块中的任何一个地方声明了该变量，该名称的变量就指的是块中声明的变量，不是任何同名的全局变量。但是在块内声明前不可获取）

const表示该变量是只读的，不可以被修改，与let类似它的作用域是块机作用域，不存在变量声明提前，不能重复定义，存在暂时性死区；

var的作用域是函数作用域，在函数外不可以访问函数内声明的变量，以及存在变量声明提前（但是赋值不提前）

1. 基本类型和引用类型相关（堆栈）

基本类型包括：Number String Boolean Null undefined Symbol, 基本类型的名值对储存在栈内存中，当变量复制基本类型时，会在栈内存中开辟新的空间存储名值对；

引用类型包括 Object， Function， Array 引用类型的名称存储在栈内存中，值存储在堆内存中，栈内存中会提供一个引用地址指向堆内存中的值；当变量复制引用类型时，会在栈内存中开辟一个空间来存储名和值的引用地址；

1. bind, call, apply相关（自己要会实现）

bind(), call(), apply() 是Function.prototype下的方法，都是用于改变函数运行作用域的；

call（），第一个参数传入作用域，后几次依次传入参数；

apply（）与call（）类似，但是参数的传入可以通过arguments或者其他数组；

bind（）区别是call()和apply()在调用函数之后会立即执行，而bind()方法调用并改变函数运行时上下文后，返回一个新的函数，供我们需要时再调用。

1. 介绍事件循环（宏任务微任务，要知道执行顺序）
2. 定时器为什么不准（有可能会让你实现一个倒计时）
3. 原型介绍

每个函数都有prototype属性，这个属性是一个指针，指向原型对象，这个对象包含特性类型的所有实例共享的属性和方法；即原型对象是给一类实例对象提供共享属性和方法的

1. 继承方法（手写）

让一个原型对象等于另一个实例；

1. 闭包介绍和它的作用（手写）

闭包是指有权访问另一个函数作用域中的变量的函数；创建闭包的常见方法是在一个函数内部创建另一个函数；

作用：一个是可以读取函数内部的变量，另一个是让这些变量值始终保持在内存中。

function f1(){

var n = 999;

function f2(){

alert(n);}

return f2;

}

f1();

1. new实现过程
   1. 创建一个空对象，将它的引用赋给this，继承函数的原型
   2. 通过this将属性和方法添加到这个对象
   3. 最后返回这个新建立的对象，也就是实例；
2. 防抖和节流（手写）
3. 数组去重、扁平化、柯里化
   1. 数组去重
      1. indexOf + for循环
      2. 排序后相邻去除法
      3. ES6 set
   2. 扁平化
   3. 柯里化
4. this指向
5. es6特性
   1. let 和 const
   2. 模板字符串 ${}
   3. 函数（在定义函数时初始化这个参数，以及箭头函数）
   4. 解构
   5. Class， extends， super
   6. Promise
   7. Set
6. promise介绍（最好会写）

一个promise封装了一个操作的结果及其状态，用于将数据请求和数据处理清晰的分开的情况下；

Promise有三种状态：pending，rejected，fulfilled

当创建一个promise对象时，首先会将其状态设置为pending，同时初始化resolved和reject的回调函数。只有当当前状态是pending时，才会执行resolved和reject的代码，更改promise的状态。

1. promise.all和promise.race介绍（手写）
   1. 当一个线程依赖于多个线程的返回结果时，应该使用promise.all. Promise.all接收一个Promise对象组成的数组作为参数，当这个数组里的所有对象的状态都变成rejected或者fulfilled的时候，才会调用then方法；
   2. 与Promise.all相似的是，Promise.race都是以一个Promise对象组成的数组作为参数，不同的是，只要当数组中的其中一个Promsie状态变成resolved或者rejected时，就可以调用.then方法了。而传递给then方法的值也会有所不同
2. async介绍（了解原理 generator+自执行器）
3. 异步的串行（可以用promise也可以async）
4. 箭头函数指向
5. set、map介绍；和数组、对象的区别
   1. 关于set
      1. 元素不能重复，如果要求唯一性，则选择set
   2. 关于map
      1. Object只提供“字符串-值”的对应，Map提供了“值-值”的对应
   3. Set，map都不能通过下标来访问
6. 模块化介绍(commnjs amd cmd umd export)
7. localstorage sessionstorage cookie
   1. localstroage存储在本地，过期自动删除
   2. sessionstrage当会话关闭，则被删除
   3. cookie在浏览器和服务器之间传递，并且具有路径的概念
8. 事件委托
9. Proxy和Object.defineProperty