**阿里、腾讯、美团、网易等一线互联网公司面试总结**

上一篇文章记录了我认为的所谓的[二线互联网公司的面试总结](http://mp.weixin.qq.com/s?__biz=MzI3ODU4MzQ1MA==&mid=2247487441&idx=1&sn=b25c45be669bd6688e459e0e416e8779&chksm=eb558e24dc22073260291073809d5de2a45d62c4a1825e230cf5d4ac2ba05cc079be1d1beff5&scene=21" \l "wechat_redirect" \t "_blank)，本篇主要记录更火的一些公司的面试

记录。其实很多题目都有重复，基础相关的、框架相关的其实都是大同小异，所以记录有删减并不是每道题都进行记录了，很多题目面试的时候回答的并不好。

**理解与问答**

**1. 你有做过Hybrid APP开发吗？说说你的经验？**

在公司移动端和PC端都做过相应的开发，然后就介绍了移动端开发的一些东西，然后说了下H5是如何和客户端进行交互的，最后说了下自己参与的一些weex开发项目，说了下weex与H5的区别。

**2. 能聊一聊weex么，原理是啥？**

简单介绍了下weex出现的原因与好处，然后说了下weex的开发流程，最后介绍了下weex的实现原理，具体可以参考下这篇文章：https://zhuanlan.zhihu.com/p/71064826

**3. 了解flutter么，和weex有啥区别呢？**

说实话之前只是了解过flutter，对于他的开发方式不是怎么了解，反正就把我知道的说了一下，面试官也是似懂非懂吧

**4. 了解微信小程序开发么，能说一说么，和常规的Hybrid有什么区别吗？**

再好的文档也比不上官方文档，微信官方文档介绍了很清楚，https://developers.weixin.qq.com/miniprogram/dev/framework/quickstart/#%E5%B0%8F%E7%A8%8B%E5%BA%8F%E7%AE%80%E4%BB%8B 小程序的主要开发语言是 JavaScript ，小程序的开发同普通的网页开发相比有很大的相似性。对于前端开发者而言，从网页开发迁移到小程序的开发成本并不高，但是二者还是有些许区别的。网页开发渲染线程和脚本线程是互斥的，这也是为什么长时间的脚本运行可能会导致页面失去响应，而在小程序中，二者是分开的，分别运行在不同的线程中。网页开发者可以使用到各种浏览器暴露出来的 DOM API，进行 DOM 选中和操作。而如上文所述，小程序的逻辑层和渲染层是分开的，逻辑层运行在 JSCore 中，并没有一个完整浏览器对象，因而缺少相关的DOM API和BOM API。

**5. 微信长按识别二维码的原理你知道么？**

说实话这个真的不清楚，瞎扯了一堆，后来找到了这篇文章：https://mp.weixin.qq.com/s/4ct3au\_BsUHJb0wrXSKSHQ 看了下是截屏处理的，现在怎么处理的也不是很清楚了，要不扫一下试试

**6. 平时我们浏览器打开一个链接会提示打开一个APP，知道是什么原理么，微信中貌似有很多限制你们有尝试过解决么？**

这其实是一个很头疼的问题，国产浏览器有各种限制，微信限制就更多了，所以很多方案是没有办法实施的 最常规的方案还是通过Scheme进行唤起，微信中没有白名单的用户是没法用的，一般是引导打开默认浏览器，或者跳转应用宝 没有完美的方法更多方案可以看callapp库作者写的文章：https://suanmei.github.io/2018/08/23/h5\_call\_app/

**7. 谈一谈你知道的前端性能优化方案有哪些？**

太多了，这边不展开了，有机会写新的文章来聊一聊

**8. 离线方案你们是怎么做的，主要是客户端做还是前端来做啊？**

前端客户端配合实现，通过提供前端资源离线包，和离线配置，同时客户端通过配置hook h5的请求来选择匹配本地资源或者请求线上，做到离线。

**9. 为什么要有请求预加载，具体怎么实现的？**

请求预加载就是在启动webview的时候，客户端同时进行接口数据的请求

**10. GraphQL能聊一聊么，为啥能够提升页面性能？**

GraphQL能够根据页面展示需求请求所需要的数据，不会有冗余的数据，传输效率上会更高

**11. 能聊一聊SSR么，你们项目中是怎么用的？**

服务端渲染，之前写过vue的ssr，更好的首屏展示，更好的SEO，对服务器的压力会更大

**12. 边缘计算是啥，他是如何提升页面性能的？**

前面提到性能优化中提到了边缘计算，也来自之前看到的文章中说的，文章在这里：https://juejin.im/post/6844904173788479502

**13. 聊了这么多你能用几个关键词对整个前端性能提升大致分类么？**

页面是否可以快速加载 是否允许用户快速开始与之交互 滚动和动画是否流畅

**14. 请写下你对协商缓存和强缓存的理解？**

网上一大把，含义的理解，关键头的理解

**15. 知道 ETag 是怎么生成的么？**

nginx里面，是由Last-Modified和content-length的十六进制组合而成 不同 Web 服务器或者 CDN 的 ETag 生成方式不一样。

**16. 聊一聊监控，你们是怎么做的？**

用户行为监控 性能监控 异常监控

**17. 如何处理异常捕获？**

监听error事件处理常规js错误 监听unhandledrejection事件处理当Promise 被 reject 且没有 reject 处理器的时候 魔改ajax请求处理接口异常

**18. 能聊一聊Script error出现的原因么？**

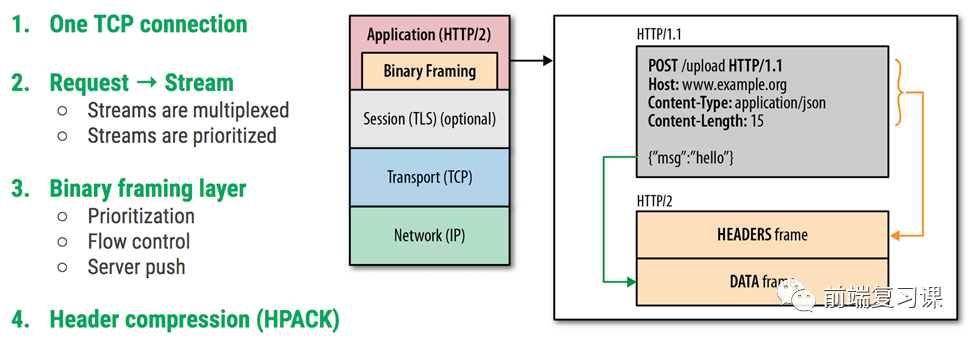
跨域脚本的异常是捕获不到的，只是抛出来Script error，增加crossorigin="anonymous"属性就好了，当然脚本cdn那边跨域请求头也是需要的

**19. 通常在发送数据埋点请求的时候使用的是 1x1 像素的图片？**

不存在跨域问题 执行过程无阻塞 体积较小 不占用ajax的请求限额

##### 20. http2了解么，能简单聊一下么？

多路复用、首部压缩、服务器推送

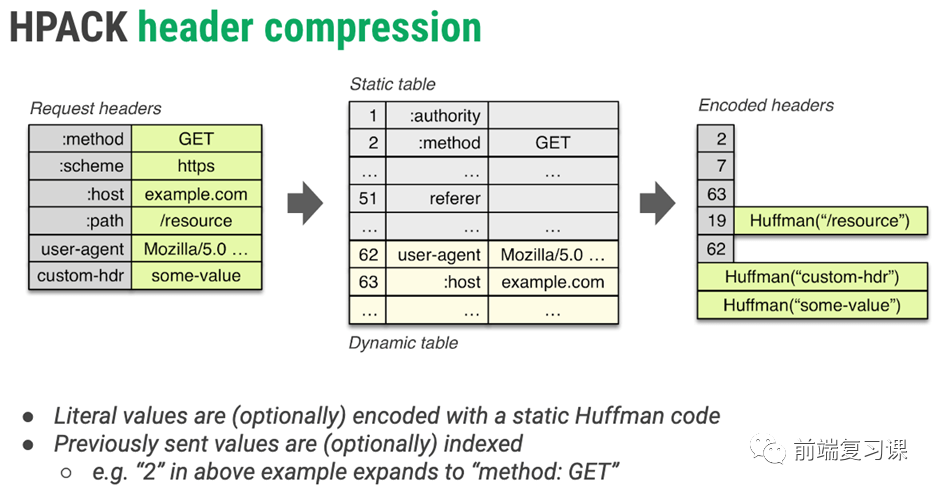


**21. 为什么http1不能实现多路复用？**

http1阶段是基于文本传输的，由于没有流的概念，在使用并行传输（多路复用）传递数据时，接收端在接收到响应后，并不能区分多个响应分别对应的请求，所以无法将多个响应的结果重新进行组装，也就实现不了多路复用。

##### 22. http2首部压缩是什么原理？

因为每个HTTP 报文都要传输臃肿的首部字段导致的网络效率降低，解决思路，通信双方可以都维护一张HTTP 首部字段索引列表，报文中只传输对应字段的索引值，就能大大压缩报文首部的长度，提高网络利用率。HTTP/2 在客户端与服务器端都维护了一张首部字段索引列表， header 字段列表是以key - value 键值对元素构成的有序集合，每个header 字段元素都映射为一个索引值，报文中使用header 字段的索引值进行二进制编码传输，显然比HTTP/1.1 直接使用header 字段ASCII 编码传输，数据量小得多，这种减少header 字段传输开销的技术可以称为首部压缩HPACK。



**23. 讲解一下https的工作原理？**

HTTPS在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。TLS/SSL协议不仅仅是一套加密传输的协议，更是一件经过艺术家精心设计的艺术品，TLS/SSL中使用了非对称加密，对称加密以及HASH算法。握手过程的简单描述如下：

浏览器将自己支持的一套加密规则发送给网站。

网站从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

获得网站证书之后浏览器要做以下工作：

验证证书的合法性（颁发证书的机构是否合法，证书中包含的网站地址是否与正在访问的地址一致等），如果证书受信任，则浏览器栏里面会显示一个小锁头，否则会给出证书不受信的提示。

如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。

使用约定好的HASH计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给网站。

网站接收浏览器发来的数据之后要做以下的操作：

使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。

使用密码加密一段握手消息，发送给浏览器。

浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

**24. http请求跨域问题，你都知道哪些解决跨域的方法？**

跨域处理的方案是在太多太多了，我们公司主要是通过nginx进行转发，前端发起请求的地址是和页面地址一致的所以不存在跨域，nginx将请求转发到正确的服务。页面地址：web.taobao.com，请求接口地址web.taobao.com/api.taobao.com/\*\*\*

##### 25. 描述一下事件循环机制，哪些常见的宏任务微任务？

**1）为什么会有Event Loop**JavaScript的任务分为两种同步和异步，它们的处理方式也各自不同，**同步任务**是直接放在主线程上排队依次执行，**异步任务**会放在任务队列中，若有多个异步任务则需要在任务队列中排队等待，任务队列类似于缓冲区，任务下一步会被移到调用栈然后主线程执行**调用栈**的任务。

调用栈：调用栈是一个栈结构，函数调用会形成一个栈帧，帧中包含了当前执行函数的参数和局部变量等上下文信息，函数执行完后，它的执行上下文会从栈中弹出。

JavaScript是单线程的，单线程是指 js引擎中解析和执行js代码的线程只有一个（主线程），每次只能做一件事情，然而ajax请求中，主线程在等待响应的过程中回去做其他事情，浏览器先在事件表注册ajax的回调函数，响应回来后回调函数被添加到任务队列中等待执行，不会造成线程阻塞，所以说js处理ajax请求的方式是异步的。

综上所述，检查调用栈是否为空以及讲某个任务添加到调用栈中的个过程就是event loop，这就是JavaScript实现异步的核心。

**2）浏览器中的 Event Loop**

**Micro-Task 与 Macro-Task**

浏览器端事件循环中的异步队列有两种：macro（宏任务）队列和 micro（微任务）队列。

常见的 macro-task：setTimeout、setInterval、script（整体代码）、 I/O 操作、UI 渲染等。

常见的 micro-task: new Promise().then(回调)、MutationObserve 等。

**requestAnimationFrame**

requestAnimationFrame也属于异步执行的方法，但该方法既不属于宏任务，也不属于微任务。按照MDN中的定义：

window.requestAnimationFrame() 告诉浏览器——你希望执行一个动画，并且要求浏览器在下次重绘之前调用指定的回调函数更新动画。该方法需要传入一个回调函数作为参数，该回调函数会在浏览器下一次重绘之前执行

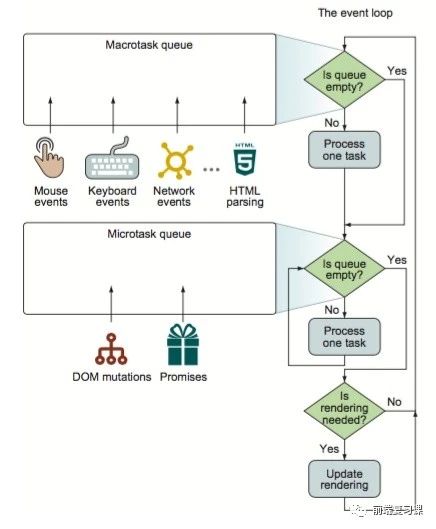
requestAnimationFrame是GUI渲染之前执行，但在Micro-Task之后，不过requestAnimationFrame不一定会在当前帧必须执行，由浏览器根据当前的策略自行决定在哪一帧执行。

**event loop过程**

检查macrotask队列是否为空，非空则到2，为空则到3,执行macrotask中的一个任务,继续检查microtask队列是否为空，若有则到4，否则到5,取出microtask中的任务执行，执行完成返回到步骤3执行视图更新.当某个宏任务执行完后,会查看是否有微

任务队列。如果有，先执行微任务队列中的所有任务，如果没有，会读取宏任务队列中排在最前的任务，执行宏任务的过程中，

遇到微任务，依次加入微任务队列。栈空后，再次读取微任务队列里的任务，依次类推。



**26. 简单聊一聊Promise的原理？**

https://zhuanlan.zhihu.com/p/58428287

**27. js数字精度丢失的问题？**

采用双精度存储造成的问题，这篇文章说的挺清楚https://www.cnblogs.com/snandy/p/4943138.html

(0.1\*10 + 0.2\*10) / 10 == 0.3 *// true*

**28. es6的模块管理与commonjs的对比？**

因为CommonJS的require语法是同步的，所以就导致了CommonJS模块规范只适合用在服务端，而ES6模块无论是在浏览器端还是服务端都是可以使用的，但是在服务端中，还需要遵循一些特殊的规则才能使用

1. CommonJS 模块输出的是一个值的拷贝，而ES6 模块输出的是值的引用
2. CommonJS 模块是运行时加载，而ES6 模块是编译时输出接口，使得对JS的模块进行静态分析成为了可能
3. 因为两个模块加载机制的不同，所以在对待循环加载的时候，它们会有不同的表现。CommonJS遇到循环依赖的时候，只会输出已经执行的部分，后续的输出或者变化，是不会影响已经输出的变量。而ES6模块相反，使用import加载一个变量，变量不会被缓存，真正取值的时候就能取到最终的值
4. 关于模块顶层的this指向问题，在CommonJS顶层，this指向当前模块；而在ES6模块中，this指向undefined
5. 关于两个模块互相引用的问题，在ES6模块当中，是支持加载CommonJS模块的。但是反过来，CommonJS并不能requireES6模块，在NodeJS中，两种模块方案是分开处理的

**29. 为啥会有virtural dom？**

vdom使用js来描述真实的dom，我们不用直接操作dom，操作数据即可，可维护性更高 vdom使得跨平台更加的方便

**30. 组件库设计原理 与一些组件介绍？**

介绍了工作中设计开发的一些组件

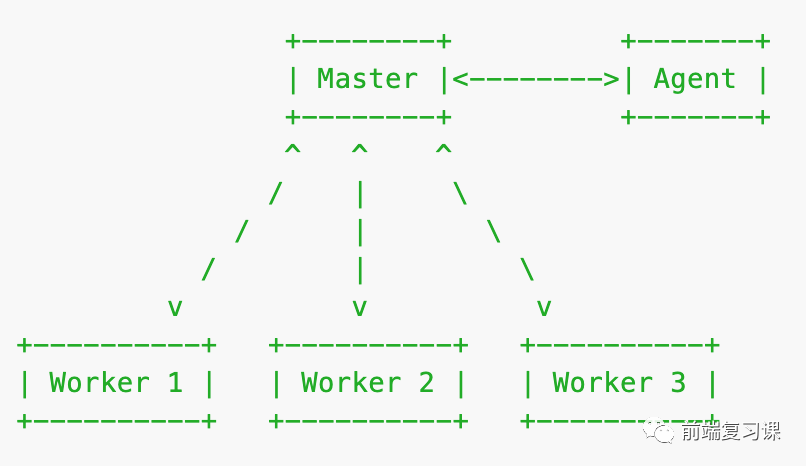
**31. node内存溢出排查？**

分析日志和代码，猜想问题出现的原因，使用工具分析，优化代码分析原因

**32. 说一说node的进程与线程**

这篇文章说的挺详细的：https://segmentfault.com/a/1190000020077274?utm\_source=sf-related

##### 33. egg多进程程原理？



**34. pm2原理？**

pm2 基于 cluster模块 进行了封装，它能自动监控进程状态、重启进程、停止不稳定进程、日志存储等。利用 pm2 时，可以在不修改代码的情况下实现负载均衡集群。

**35. 介绍下xss和csrf吧？**

搜一下一大把，开新的文章来讲

**36. 项目中还遇到哪些前端安全问题？**

可以看我之前写的另外一篇文章

[2020全网最全前端安全综述](http://mp.weixin.qq.com/s?__biz=MzUxODc0ODQ1NA==&mid=2247483881&idx=1&sn=b21c82eb92f38bd3d0d4c13de174003f&chksm=f98561a0cef2e8b6f836235f31dd197d1b4abbf1069edc50c6395b7095ee94d3b96a8986a6d2&scene=21#wechat_redirect)

**37. 你刚说的薅羊毛和爬虫，一般是怎么处理的？**

这边打算开新的文章来讲

**38. babel的作用？**

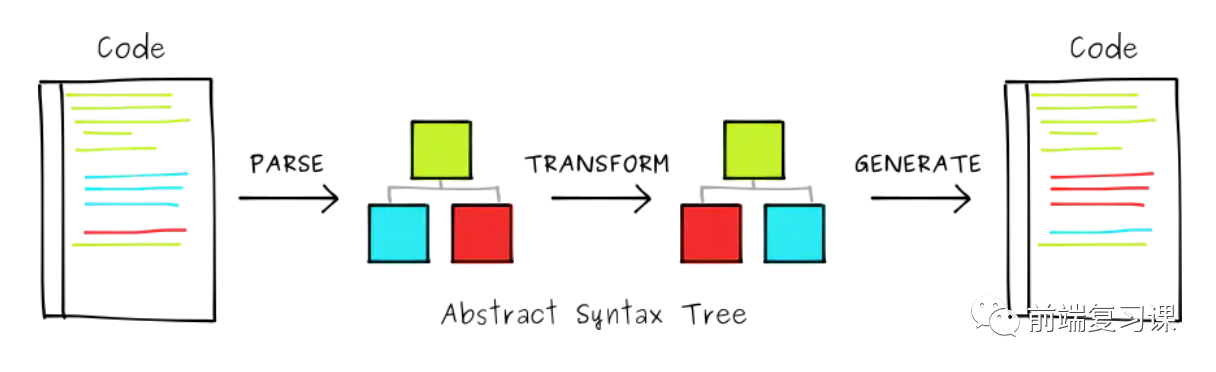
Babel是一个广泛使用的转码器，可以将ES6代码转为ES5代码，需要搭配各种插件使用

**39. 自己写过什么babel、webpack 插件？**

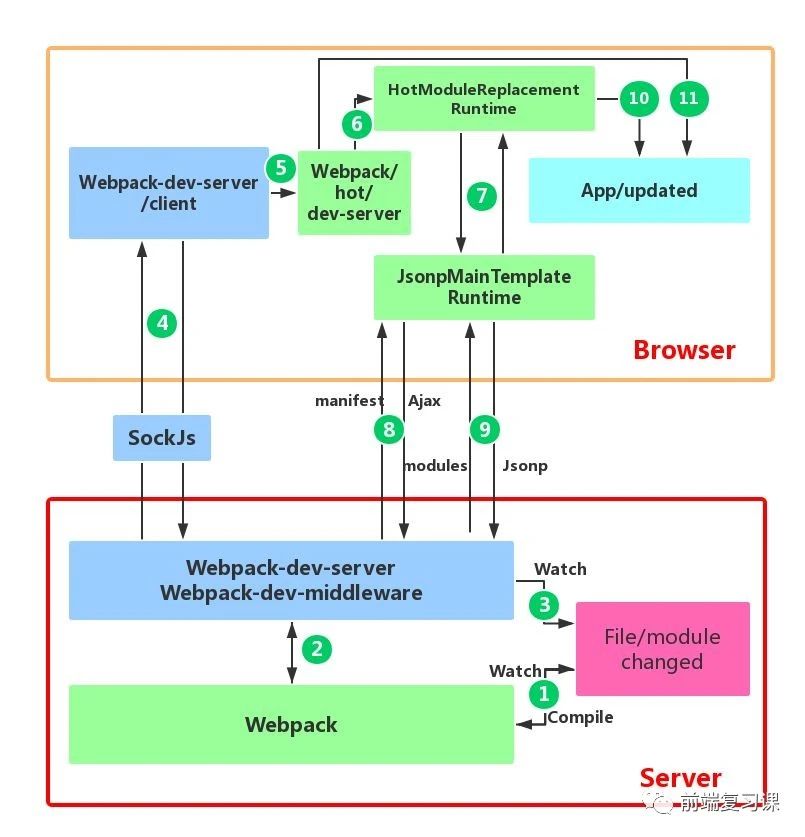
简单介绍了下自己写过的插件

##### 40. ast具体的作用是啥？

AST （Abstract Syntax Tree）是抽象语法树的英文简称，它从源代码到运行的编译程序过程中起到重要的作用。AST 是源代码语法结构的一种抽象表现形式，通常以树的数据结构来表达，树上每一个节点都表现出源代码中的结构。Babel 编译器的编译过程，它将 JavaScript 的高级版本编译成低级版本，整个过程离不开抽象语法树对源代码的抽象过程。



##### 41. 聊一聊Webpack热更新的原理？



1. 第一步，在 webpack 的 watch 模式下，文件系统中某一个文件发生修改，webpack 监听到文件变化，根据配置文件对模块重新编译打包，并将打包后的代码通过简单的 JavaScript 对象保存在内存中。
2. 第二步是 webpack-dev-server 和 webpack 之间的接口交互，而在这一步，主要是 dev-server 的中间件 webpack-dev-middleware 和 webpack 之间的交互，webpack-dev-middleware 调用 webpack 暴露的 API对代码变化进行监控，并且告诉 webpack，将代码打包到内存中。
3. 第三步是 webpack-dev-server 对文件变化的一个监控，这一步不同于第一步，并不是监控代码变化重新打包。当我们在配置文件中配置了devServer.watchContentBase 为 true 的时候，Server 会监听这些配置文件夹中静态文件的变化，变化后会通知浏览器端对应用进行 live reload。注意，这儿是浏览器刷新，和 HMR 是两个概念。
4. 第四步也是 webpack-dev-server 代码的工作，该步骤主要是通过 sockjs（webpack-dev-server 的依赖）在浏览器端和服务端之间建立一个 websocket 长连接，将 webpack 编译打包的各个阶段的状态信息告知浏览器端，同时也包括第三步中 Server 监听静态文件变化的信息。浏览器端根据这些 socket 消息进行不同的操作。当然服务端传递的最主要信息还是新模块的 hash 值，后面的步骤根据这一 hash 值来进行模块热替换。
5. webpack-dev-server/client 端并不能够请求更新的代码，也不会执行热更模块操作，而把这些工作又交回给了 webpack，webpack/hot/dev-server 的工作就是根据 webpack-dev-server/client 传给它的信息以及 dev-server 的配置决定是刷新浏览器呢还是进行模块热更新。当然如果仅仅是刷新浏览器，也就没有后面那些步骤了。
6. HotModuleReplacement.runtime 是客户端 HMR 的中枢，它接收到上一步传递给他的新模块的 hash 值，它通过 JsonpMainTemplate.runtime 向 server 端发送 Ajax 请求，服务端返回一个 json，该 json 包含了所有要更新的模块的 hash 值，获取到更新列表后，该模块再次通过 jsonp 请求，获取到最新的模块代码。这就是上图中 7、8、9 步骤。
7. 而第 10 步是决定 HMR 成功与否的关键步骤，在该步骤中，HotModulePlugin 将会对新旧模块进行对比，决定是否更新模块，在决定更新模块后，检查模块之间的依赖关系，更新模块的同时更新模块间的依赖引用。
8. 最后一步，当 HMR 失败后，回退到 live reload 操作，也就是进行浏览器刷新来获取最新打包代码。

**42. webpack打包太慢了是如何优化的啊？**

使用高版本的 Webpack、多线程/多实例构建、缩小打包作用域、充分利用缓存提升二次构建速度、DLL

**43. 介绍template到render的过程？**

这个最好自己能够看一看源码了

**44. 介绍下React Hooks和vue composition api？**

hooks和composition api是react和vue最新的代码组织方式 他们的出现是为了更好的逻辑复用，使得代码的逻辑更加的清晰 vue2.0中代码组织方式更多的是options，这样很多代码都揉在了一起，并不清晰，复用多用mixin来写，mixin也有很大的问题，多个mixin的时候不知道变量来自哪个mixin，代码跳跃感太强，因为根本没定义就使用了，给人一种难以理解的感觉，同时多个的时候可能会存在变量冲突

**45. for循环中key的作用？**

更快、更准确

**46. 能不能介绍一下vuex的Mutation和Action的区别吗？**

Mutation是同步的，唯一改变store的方法 Action 提交的是 Mutation，而不是直接变更状态。

**47. 为什么Vuex要分为Action和Mutation，Mutation为啥只能是同步的？**

区分 actions 和 mutations 并不是为了解决竞态问题，而是为了能用 devtools 追踪状态变化。vuex 真正限制你的只有 mutation 必须是同步的，同步的意义在于这样每一个 mutation 执行完成后都可以对应到一个新的状态，这样 devtools 就可以打个 snapshot 存下来，然后就可以随便 time-travel 了。

**编程与算法**

**1. 实现sendRequest(promises, max, callback)，同时最多执行max个promises，超过的等待有空闲的开始执行，执行完成后执行callback**

function sendRequest(promises, max, callback) {  
    let current = 0  
    const results = []  
    const originLen = promises.length  
    const next = () => {  
        while (current < max && promises.length) {  
            const index = originLen - promises.length  
            const promise = promises.shift()  
            results[index] = {  
                value: null,  
                reason: null  
            }  
            Promise.resolve(promise).then(value => {  
                results[index].value = value  
            }, reason => {  
                results[index].reason = reason  
            }).finally(() => {  
                current--  
                next()  
            })  
            current++  
        }  
  
        if (current === 0) {  
            callback(results)  
        }  
    }  
    next()  
}

**2. 实现compose函数，类似koa-compose中间件**

function compose(middlewares) {  
    return function() {  
        dispatch(0)  
        function dispatch(i) {  
            const fn = middlewares[i]  
            if (!fn) return  
            return fn(function next() {  
                dispatch(i + 1)  
            })  
        }  
    }  
}

**3. 单项链表是否存在环**

力扣第141题：https://leetcode-cn.com/problems/linked-list-cycle/

function hasCycle(head) {  
    let temp = head  
    const map = new Map()  
    while (temp) {  
        if (map.has(temp)) {  
            return true  
        }  
        map.set(temp, true)  
        temp = temp.next  
    }  
    return false  
};

**4. 螺旋数组**

类似力扣第59题：https://leetcode-cn.com/problems/spiral-matrix-ii/

function generateMatrix (n) {  
    *// 定义一个二维数组进行数据保存*  
    const result = []  
    for (let i = 0; i < n; i++) {  
        result.push(new Array(n))  
    }  
    let left = 0  
    let right = n - 1  
    let top = 0  
    let bottom = n - 1  
    let current = 1  
    while(left <= right && top <= bottom) {  
        *// 左边从上到下*  
        for (let i = top; i <= bottom; i++) {  
            result[i][left] = current++  
        }  
        if (++left > right) break  
        *// 下边从左到右*  
        for (let i = left; i <= right; i++) {  
            result[bottom][i] = current++  
        }  
        if (--bottom < top) break  
        *// 右边从下到上*  
        for (let i = bottom; i >= top; i--) {  
            result[i][right] = current++  
        }  
        if (--right < left) break  
        for (let i = right; i >= left; i--) {  
            result[top][i] = current++  
        }  
        if (++top > bottom) break  
    }  
    return result  
}

**5.实现 findFirstIndex函数，找到有序数组 [1, 2, 3, 4, 7, 7, 7, 9, 12, 23, 34, 45, 55, 67]中第一次出现的位置，比如7第一次出现的位置是4**

function findFirstIndex(arr, target) {  
    let begin = 0  
    let end = arr.length  
    while(begin < end) {  
        const mid = (begin + end) >>> 1  
        if (arr[mid] >= target) {  
            end = mid  
        } else {  
            begin = mid + 1  
        }  
    }  
    if (begin === arr.length) return -1  
    return arr[begin] === target ? begin : -1  
}

**6. 十进制转换成任意进制**

function tenToOther(num, base) {  
    const baseNumber = '0123456789abcdefghijklmnopqrstuvwxyz'  
    const result = []  
    while (num) {  
        const rest = num % base  
        num = Math.floor(num / base)  
        result.unshift(baseNumber[rest])  
    }  
    return result.join('')  
}

**7. 大数相加**

function bigAdd(a, b) {  
    const aArr = a.split('')  
    const bArr = b.split('')  
  
    let flag = 0  
    let result = []  
    while(aArr.length || bArr.length) {  
        const left = aArr.pop() || 0  
        const right = bArr.pop() || 0  
  
        const value = Number(left) + Number(right) + flag  
  
        result.unshift(value % 10)  
  
        flag = parseInt(value / 10)  
    }  
  
    if (flag) {  
        result.unshift(flag)  
    }  
  
    return result.join('')  
}

**8. 深拷贝**

function deepClone(obj, map = new WeakMap()) {  
    if (obj === null || typeof obj !== 'object') {  
        return obj  
    }  
  
    if (map.has(obj)) {  
        return map.get(obj)  
    }  
  
    const copy = Array.isArray(obj) ? [] : {}  
  
    map.set(obj, copy)  
  
    const keys = Reflect.ownKeys(obj)  
  
    keys.forEach(key => {  
        copy[key] = deepClone(obj[key], map)  
    })  
  
    return copy  
}

**9. 青蛙跳台阶**

其实就是爬楼梯问题，剑指 Offer 10- II. 青蛙跳台阶问题: https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/

var climbStairs = function(n) {  
    if (n <= 2) return n  
    let n1 = 1  
    let n2 = 2  
    let nn = 0  
    for (let i = 3; i <= n; i++) {  
        nn = n1 + n2  
        n1 = n2  
        n2 = nn  
    }  
    return nn  
};

**10. 图片懒加载写代码**

function lazyload() {  
    const observe = new IntersectionObserver(enteris => {  
        enteris.forEach(entry => {  
            const lazyImage = entry.target  
            if (entry.isIntersecting && lazyImage.getAttribute('src') == 'loading.gif') {  
                lazyImage.src = lazyImage.dataset.src  
                observe.unobserve(lazyImage)  
            }  
        })  
    })  
  
    for (let i = 0; i < imgs.length; i++) {  
        observe.observer(imgs[i])  
    }  
}

**11. 二叉树的最大深度**

力扣第104题：https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/

var maxDepth = function(root) {  
    if (!root) {  
        return 0  
    }  
    const left = maxDepth(root.left)  
    const right = maxDepth(root.right)  
    return Math.max(left, right) + 1  
};

**12. 包含min函数的栈**

剑指 Offer 30: https://leetcode-cn.com/problems/bao-han-minhan-shu-de-zhan-lcof/

*/\*\*  
 \* initialize your data structure here.  
 \*/*  
var MinStack = function() {  
   this.stack=[];  
   this.stackmin=[];  
};  
  
*/\*\*   
 \* @param {number} x  
 \* @return {void}  
 \*/*  
MinStack.prototype.push = function(x) {  
   this.stack.push(x);   
   if(this.stackmin.length===0 || this.stackmin[this.stackmin.length-1]>=x){  
       this.stackmin.push(x);  
   }  
};  
  
*/\*\*  
 \* @return {void}  
 \*/*  
MinStack.prototype.pop = function() {  
   let x=this.stack.pop();  
   if(x===this.stackmin[this.stackmin.length-1]){  
     this.stackmin.pop();  
   }  
};  
  
*/\*\*  
 \* @return {number}  
 \*/*  
MinStack.prototype.top = function() {  
   return this.stack[this.stack.length-1];  
};  
  
*/\*\*  
 \* @return {number}  
 \*/*  
MinStack.prototype.min = function() {  
  return this.stackmin[this.stackmin.length-1];  
   
};

**13. 连续子数组的最大和**

力扣第104题：https://leetcode-cn.com/problems/maximum-subarray/

var maxSubArray = function(nums) {  
    let current = nums[0]  
    let max = nums[0]  
    for (let i = 1; i < nums.length; i++) {  
        if (current <= 0) {  
            current = nums[i]  
        } else {  
            current += nums[i]  
        }  
  
        if (max < current) {  
            max = current  
        }  
    }  
  
    return max  
};

**14. 最小路径和**

力扣第64题：https://leetcode-cn.com/problems/minimum-path-sum/

var minPathSum = function(grid) {  
 let n = grid.length;  
    let m = grid[0].length;  
    let dp = Array.from(new Array(n), () => new Array(m));  
    for (let i = 0; i < n; i ++) {  
        for (let j = 0; j < m; j ++) {  
            if (i != 0 && j != 0) {  
                dp[i][j] = Math.min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];  
            } else if (i == 0 && j != 0) {  
                dp[i][j] = dp[i][j - 1] + grid[i][j];  
            } else if (i != 0 && j == 0) {  
                dp[i][j] = dp[i - 1][j] + grid[i][j];  
            } else if (i == 0 && j == 0) {  
                dp[i][j] = grid[i][j];  
            }  
        }  
    }  
    return dp[n - 1][m - 1];  
};

**15. 二叉树的左右子树交换代码实现**

力扣第226题：https://leetcode-cn.com/problems/invert-binary-tree/

var invertTree = function(root) {  
    if (root === null) return null;  
    const left = invertTree(root.left);  
    const right = invertTree(root.right);  
    root.left = right;  
    root.right = left;  
    return root;  
};

**16. 二树叉，找出两节点的最近公共祖先**

力扣第235题：https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

const lowestCommonAncestor = (root, p, q) => {  
  while (root) {  
    if (p.val < root.val && q.val < root.val) {  
      root = root.left;  
    } else if (p.val > root.val && q.val > root.val) {  
      root = root.right;  
    } else {  
      break;  
    }  
  }  
  return root;  
};

**开放问答**

**1. 项目开发中有遇到什么挑战没**

可以从技术上说一说，也可以从业务处理上说一说

**2. 工作中能够持续学习么**

谈一谈自己学习的动力，以及怎么学习的

**3. 未来会有什么样的规划**

说了一些自己的想法，未来希望成长到什么程度

**4. 对哪个项目印象比较深刻深刻，遇到最难的项目是啥**

根据自己的项目来回答了，其实项目都上线了，并没有啥难不难的了，问题都解决了，主要是解决问题的过程比较重要，最后总结得出了啥

**5. 项目研发流程中作为前端开发一般扮演的啥角色**

自己开发中扮演的角色

**6. 现在有的项目中觉得哪些项目可以继续优化，为啥没有优化**

说一说现在的项目还存在的哪些问题，为啥没一开始就优化，自己打算以后怎么维护

**7. 最近比较关心的技术**

说一说最近看的比较新的东西