B.前端面试标准

面试流程

面试前准备：提前阅读候选人简历

- 对候选人的资历进行一个大致了解
- 针对性的准备一些技术问题

面试阶段：

1、双方自我介绍，面试官先来自我介绍，介绍整个面试流程，前半段项目和基础知识点，后半段笔试题

2、项目和基础知识点，控制在 30 分钟以内，针对每个知识点由浅入深，当深入到候选人回答不上了就换下一个知识点

3、笔试题，控制在 30 分钟以内

4、提问环节，介绍团队和公司情况

基础知识点汇总

前端面试题库

类型	题目	考察点	答案	评分建议	适用级别	推荐人	好用
JS	【陈述题】异步编程有哪些实现方式，各方案的优缺点是什么呢	1. callback、promise、generator、async/await 的基本了解 2. 各方案的优缺点		初阶： 能讲出2个以上特性 中高阶： 能讲出各方案的基本概念，回调地狱的问题即及格 能说出执行权转移、错误捕获等可加分	all	蔡璐麒	
	【陈述题】 1. Event Loop 的大致流程 2. 哪些是宏任务，哪些属于微任务 3. 执行栈、任务队列的理解			能回答出大致流程以及宏任务、微任务划分无误即可及格 对执行栈、任务队列有涉及并且理解可加分 如果有使用 node，对node 和 浏览器的 Event Loop 区别了解可加分	all	蔡璐麒	
	【陈述题】 1. 代码会输出什么?为什么 2. 如果 var 换成 const，输出会变吗 `var a = 1` `var obj = {` `a: 2,` `getA: () => {` `return this.a` `}` `}` `console.log(obj.getA())`	1. this 指向 2. let、const 的特性 3. 执行环境的理解 活动对象和变量对象的区别 作用域是如何确立的 结合闭包来考察		第一题回答正确并了解函数的 this 指向即及格 第二题回答正确并能提及执行环境即加分	all	蔡璐麒 陈宁	
	ES模块规范和commonjs模块规范的相同点和不同点	模块化 1. ES6 输出的是值的引用，cjs 则是值的拷贝 2. ES6 是编译时加载，cjs 是运行时加载		答出一个即及格，两个加分	all	蔡璐麒	

【笔试题】Object.assign polyfill的实现	1、考察候选人对对象拷贝的了解 2、考察候选人编码能力	参见：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object/assign ``` if (typeof Object.assign != 'function') { // Must be writable: true, enumerable: false, configurable: true Object.defineProperty(Object, "assign", { value: function assign(target, varArgs) { // .length of function is 2 'use strict'; if (target == null) { // TypeError if undefined or null throw new TypeError('Cannot convert undefined or null to object'); } let to = Object(target); for (var index = 1; index < arguments.length; index++) { var nextSource = arguments[index]; if (nextSource != null) { // Skip over if undefined or null for (let nextKey in nextSource) { // Avoid bugs when hasOwnProperty is shadowed if (Object.prototype.hasOwnProperty.call(nextSource, nextKey)) { to[nextKey] = nextSource[nextKey]; } } } } return to; }, writable: true, configurable: true }); } ```	初阶： 知道assign的准确作用，且能实现拷贝的过程及格 有边界判断逻辑，考虑的比较周全加分 中阶： 有边界判断逻辑且主要功能能实现及格	<=中阶	左现金	
【笔试题】写出如下代码的执行结果，并说出为什么 ``` new Promise((resolve,reject)=>{ console.log(1); setTimeout(()=>{ console.log(2); }); resolve(); }).then(()=>{ console.log(3); }).then(()=>{ return new Promise((resolve,reject)=>{ console.log(4); }).then(()=>{ console.log(5); }); }).then(()=>{ console.log(6); }); console.log(7); ```	1、考察候选人对Promise执行过程的理解以及浏览器环境event loop的了解	1 7 3 4 2	中阶： 3个以上连续顺序正确且能讲清楚及格，完全答对且能讲清楚加分 高阶： 全部正确且能讲清楚及格	>= 中高阶	左现金	

	【笔试题】写出如下代码的执行结果 `Object.prototype.type = "x";` `Function.prototype.type = "y";` `function A() {};` `const a = new A();` `console.log(A.type);` `console.log(a.type);` 如果候选人没有思路，可以换一种问法（若上面答对了就可以不问） `console.log(a instanceof Function);` `console.log(a instanceof Object);` `console.log(A instanceof Object);`	深刻理解 JS 原型链	"y"; "x"; ====== false; true; true;	能给出正确答案，得分； 能给出原型链分析过程，加分； 能模拟实现 instanceof，加分；	>= 中阶	张青天	
	数据相关方法的考察（特别是遍历相关）					陈宁	
	用js实现一个继承（考察原型相关）					陈宁	
HTML/CSS	0、盒模型（组成以及标准）	考察候选人 CSS 基础知识	1、每个盒子有四个边界：内容边界 Content edge、内边距边界 Padding Edge、边框边界 Border Edge、外边框边界 Margin Edge 2、CSS 中的 box-sizing 属性定义了 user_agent 应该如何计算一个元素的总宽度和总高度 —— box-sizing: content-box | border-box; 3、box-sizing 的默认值是 content-box，若改为 border-box 则一个元素的 width 设为 100px 时，100px会包含它的 border 和 padding（不含margin）	评分标准： • 至少列出几个组成部分 • 至少知道 content-box 和 border-box 这两个属性值的不同之处	all	未知用户 (panyiqian)	
	1、一个没有任何样式设定的 div元素，使其位于视窗区域水平居中，同时垂直居中的实现方案有哪些?	考察候选人 CSS 应用	1、设定初始样式为 display: inline-block; 2、设定初始样式为 display: flex 3、设定初始样式为 position: absolute; 4、设定初始样式为 display: table-cell	评分标准： • 至少答出 3 种，最好包含 flex 的实现方式； • 满分同学需要描述清父子节点具体条件，如父节点 relative 或 自身 width: 100% 等	all	未知用户 (panyiqian)	
	2、文字下划线的几种实现 下划线文字覆盖问题 比如：global 鱼玉钰昱	考察候选人 CSS 应用	1、box-shadow: 0 1px; 2、border-bottom: 1px solid; 3、background-image: linear-gradient(to top, #f00, #f00 1px, transparent 1px) 4、text-decoration:underline text-decoration-skip: ink 5、svg 滤镜 filter: url('#svg-underline'); 6、https://github.com/wentin/underlineJS	基础 1 2 3 4 扩展 5 6	<=中阶	赵海颖	
	3、元素定位相关 • position 取值有哪几个 • sticky 实现粘性定位 • 几种粘性定位失效的情况	1、考察候选人对浏览器新特性的了解 2、考察候选人对原生 DOM/BOM 操作对了解 3、考察候选人对 position 的了解	• static relative absolute fixed sticky • 解释出 流盒 粘性定位元素的包含块 粘性约束矩形 • 粘性定位元素父元素和自身高度计算值一样的时候、粘性定位元素的某个祖先元素的 overflow 属性值不是 visible	1、可以实现粘性定位（css、js都可以）及格 2、能够讲清楚各种方案的背景、特点加分		赵海颖	
	4、伪元素和伪类的区别，以及分别举例应用场景?	考察候选人 CSS 基础	1. 伪类与伪元素的区别：有没有创建一个文档树之外的元素。 2. 伪类可以设置元素在特殊状态下的样式，如 hover、focus等，也可以获取特定位置的元素，如 nth-child； 3. 伪元素通常可以标示一些文字旁的箭头、对勾等效果，还可以解决一些定位问题	1. 至少可列举出常用伪类，如 hover、active、focus； 2. 至少可列举出一种伪元素应用场景，如::after 加个跟随的对勾	all	未知用户 (panyiqian)	
	5、一个 ul 元素下属 5 个 li 元素，实现仅针对第4个元素设置背景色的方案?	考察候选人 CSS 知识	1、:nth-of-child/nth-of-type 2、:not()	评分标准： 答案需要包含至少一种伪元素实现	all	未知用户 (panyiqian)	

	6、前端需要注意哪些 SEO?	考察候选人 HTML 知识	1. 合理的 title、description、keywords 2. 符合W3C规范（口述基本的语义化标签，如 nav、section） 3. 重要 SEO 相关内容不要用 js 动态输出	评分标准： 第1、2 条为基础；第3条或拓展其他细节加分项	all	未知用户 (panyiqian)	
浏览器	1、浏览器回流与重绘 const ele = document.getElementById('container'); ele.style.width='20px'; ele.style.padding='10px'; ele.style.margin='10px'; 2、文档 link 标签和 script 标签的顺序	1、从题目引出浏览器回流和重绘机制，及如何减少这两种情况的发生，提高页面渲染性能	回流：计算各个 DOM 节点的位置和大小的过程。 重绘：将 DOM 节点计算后的尺寸映射到屏幕像素点的过程。 如何减少以上两种情况发生： 1、尽可能的减少发生次数，题目的答案是：1、ele.style.cssText += 'width: 20px;padding:10px;margin:10px'; 2、添加一个 class。其它可以减少回流重绘次数答案均可以。 2、dom 多次操作时可以使其先脱离文档流后进行一系列操作再填回 DOM 树。 3、避免使用一些情况浏览器布局缓存的元素属性，如 offsetXXX、clientXXX、scrollXXX、getBoundingClientRect，如果一定要用，看看能不能做一些缓存策略，避免频繁直接从元素上读取这些属性。 4、动画节点优先考虑使用绝对定位，使其脱离文档流。 5、开启动画 css3 硬件加速，可以减少一部分回流，但是会占用更多内存，不建议。	答出两个概念及至少两个节省性能的方式及格。	ALL	宗庆然	
Vue 技术栈	Vue2.x数据响应式原理？3.x？是如何处理数组的?	1、MVVM 2、响应式数据		1、至少能说出Dep、Watch和Observer的功能 3分 （ALL level） 2、能结合patch/render过程穿起来讲就加分 （中高阶） 3、render的过程可以发散一下Vue单文本组件的编译 （中高阶）	all	陈宁	
	问题：请详细描述一下vue组件的生命周期?	生命周期		1、至少要能列举出vue的所有生命周期 3分 2、能结合Vue构造函数实例化的过程来解释生命周期，能结合data和props数据的流转的，加分 3、如果提到lkeep-alive的专用生命周期activited，可以发散问一下keep-alive的效果和原理	all	陈宁	
	vue nextTick的作用和原理				>=中高阶	左现金	
	自定义组件实现v-model指令				ALL	陈宁	
	配置策略模式的考察				高阶	陈宁	
React 技术栈	state 与 props 区别? 附加：选择使用 state 还是 props 的依据?	1、考察React基础考察理解与语言组织能力	+ 状态与属性 + 可变与不可变 + 受控与非受控 + UI与交互			丁向洋	
	几个生命周期为什么会被废弃					未知用户 (lihongliang)	
	hook解决来哪些问题					未知用户 (lihongliang)	

	创建组件的方式 扩展：类组件与函数组件区别，PureComponent 与 Component 等	+ 生命周期（扩展：生命周期） + 有无状态 + props 不可变，this 可变	// username 初始值 1，点击按钮后，切换该值为 2， 此时弹出不同的值 // 原因：函数组件捕获渲染所用到的属性值 function FunctionComponent(props) { const handleClick = () => { setTimeout(() => { // 弹出渲染阶段的属性值 1 alert("函数组件值: " + props.username); }, 5000); }; return <button onClick={handleClick}>函数组件点击</button>; } class ClassComponent extends Component { handleClick = () => { setTimeout(() => { // 弹出切换后的属性值 2 alert("类组件值: " + this.props.username); }, 6000); }; render() { return <button onClick={this.handleClick}>类组件点击</button>; } }	加分项 props 不可变，this 可变		丁向洋	
	setState 的使用	+ 用法？使用差异 + 异步与同步 + 事件处理器中多次调用 setState， re-render 次数？原因？ + 视情况考察原理掌握度		加分项 1. 同步场景 2. 原理？		丁向洋	
	React组件复用方案	+ mixin + HOC + Render Props + Function + HOOK扩展： Hook		标准： + 了解 1-2种方案 + 了解 1-2种方案并有实际应用场景 + 完整了解目前常用方案，并完整回答优劣，适用场景等		丁向洋	
	React 性能优化方案	+ key + PureComponent + shouldComponentUpdate + memoize + 其他...		加分项： 从 计算 与渲染层面回答		丁向洋	
协议	HTTP(s) 连接建立过程	1. 了解协议中的请求内容 2. 了解三次握手 3. HTTPS 可以了解一下 SSL 连接建立过程 4. 为什么 HTTPS 可以保证安全	-	1. 至少说出三次握手和四次挥手的过程 2. 至少说出 HTTPS 相对于 HTTP 的链接过程变化	ALL	未知用户 (lihongliang)	
	keep-alive 是干什么的，有什么优点	保持长连接、降低拥塞等	-		ALL	未知用户 (lihongliang)	
	Cache-Control 的作用	1. 对前端缓存的了解 2. Expires 和 Cache-Control 的优先级	1. - 2. Cache-Control 来定义缓存过期时间，Cache-Control -> Expires	1. 至少说出优先级区别 2. 能讲出前端的缓存策略	ALL	未知用户 (lihongliang)	
	cookie http-only 的作用和用法	安全因素（可追问前端的安全方案）			ALL	未知用户 (lihongliang)	
	解释一下什么是同源策略及限制，如何跨域	1. 同源策略 2. 哪些资源标签可以绕过 3. 跨域的方案 4. 哪些请求是受同源策略限制的	1. 协议+域名+端口 必须一致 2. 标签 a. <img src=XXX> b. <link href=XXX> c. <script src=XXX> 3. 方案 a. CORS b. jsonp c. postMessage	能说出同源策略 1. 说出至少一种标签 2. 说出至少一种跨域实现方案	ALL	未知用户 (lihongliang)	

框架/架构	【笔试题】实现一个 jsonp 函数 `const data = await jsonp(url);`	1. 基本的前后端跨域方式的了解程度； 2. jsonp 的原理； 3. promise 的同步转化； 4. 注册全局函数的技巧（可扩展到其它 jsBrige 的实现），内存的手动回收； 5. url 加 callback 参数的细节； 6. 异常情况的处理； 7. Conten-Type:text/javascript	`function jsonp (url) {` `  const getCallbackName = () => { /** todo */ };` `  const createScript = (url) => { /** todo */ };` `  const getNewUrl = (callback) => { /** todo */ }` `  return new Promise((resolve, reject) => {` `  let callback = getCallbackName();` `  let newUrl = getNewUrl(callback);` `  window[callback] = (data) => {` `  resolve(data);` `  delete window[callback];` `  }` `  createScript(newUrl);` `  });` `}`	基本实现及格，注重 url callback 参数细节、注重全局方法回收加分	All	王明全	
	【笔试题】手写一个redux	1. 是否关注并研究常见的类库源码 2. 发布订阅的理解； 3. 即使没看过源码，现场思考，如何破题入手；	`const createStore = (reducer) => {` `  let state;` `  let listeners = [];` `  const getState = () => state;` `  const dispatch = (action) => {` `  state = reducer(state, action);` `  listeners.forEach(listener => listener());` `  };` `  const subscribe = (listener) => {` `  listeners.push(listener);` `  return () => {` `  listeners = listeners.filter(l => l !== listener);` `  }` `  };` `  dispatch({});` `  return { getState, dispatch, subscribe };` `};`	1、2 能答上来及格，3 观察具体水平而定	>=中阶	王明全	
	【陈述题】一个新的项目如何上手，比如： 1. 跨页面联动音乐网站（当某个页面中已经实例化播放器，可以被其它页面共享调用）； 2. 仿微信读书 web 阅读器（仿生翻页、字体、字体大小、主题、目录跳转、书签等）； 3. 仿 ATM（高精度、实体键盘驱动、布局、安全性）	1. 需求理解能力； 2. 项目流程； 3. 难点攻克； 4. 技术选型； 5. 项目设计；		如果候选人其它基础知识问题都不大，可以通过本题目考察一下候选人的工作方式方法、难点攻关思路，看是否可以快速的承担起一块关键业务	>=中阶	王明全	
	【陈述题】异常监控	1. 对于监控的意义的认识和指标的拆分； 2. 常见的技术选型（流程的监控产品、框架、自研的思路）； 3. 自研的话，如何拆解和架构； 4. 如何做前后端的全链路分析； 5. 如何发挥监控的最大价值；		不用特意考察本题，如果候选人的简历中，有相关经历，再详细考察比较合适。	高阶	王明全	

	【陈述题】to B 性质的产品，如何优雅的解决客户问题	抛出神策当前面临的前端工单问题，看看候选人有什么思路： 1. 页面操作录制； 2. 状态录制； 3. 全链路数据收集； 4. 数据脱敏； 5. 数据上传； 6. 数据分析；			高阶	王明全	
	【陈述题】设计并实现一个webpack插件实现检测chunk内容的变更 1. 对于 webpack 的原理、概念的掌握； 2. 常见的优化手段； 3. dll plugin 的原理及实践；				>=中阶	左现金	
	【陈述题】过往项目中有没有一些模块化应用的最佳实践，具体是如何设计和实现的	1. 工作中如何模块化（比如是否如神策的 app、业务单元、业务组件、ui 组件） 2. 如何优雅的维护不同的模块； 3. 关键模块如何保证质量（是否引入 CR、UT、E2E 等）			>=中阶	左现金	
工程化	Tag: 工作流、开放题 请问在最近的工作中，您所在团队的工作流是怎样的 您认为理想的工作流又是怎样的呢	第一问考察经历过的工程工作流 第二问考察对工程相关知识概念的理解和储备	此题为开放题， 在回答过程中，如果答案较笼统，注意追问每个环节的细节	对工程流程有一定概念，熟悉常见的敏捷开发流程，思路清晰即可及格 对流程优化、工程管理有经验的可以适当加分	All	严田	
	Tag: CI/CD、开放题 1. 什么是 CI/CD，CI/CD有什么区别 2. CI/CD 是为了解决什么问题 3.有配置过项目的 CI/CD 么 （需要询问某一个项目具体的CI/CD 的 pipeline 配置）	考察候选人对于 CI/CD 的理解，及实施能力	CI/CD 是一种通过在应用开发阶段引入自动化来频繁向客户交付应用的方法。CI/CD 的核心概念是持续集成、持续交付和持续部署。作为一个面向开发和运营团队的解决方案，CI/CD 主要针对在集成新代码时所引发的问题（亦称：“集成地狱”） CI 指持续集成，应用代码的新更改会定期构建、测试并合并到共享存储库中。该解决方案可以解决在一次开发中有太多应用分支，从而导致相互冲突的问题。 CD 指的是持续交付和/或持续部署，确保尽可能减少部署新代码时所需的工作量。	对 CI/CD有了解，知道是什么，为了解决什么问题 → 及格 配置过 CI → 优秀	>=中阶	严田	
	Tag: 脚手架、开放题 1.什么是前端脚手架 2.脚手架的出现是为了解决什么问题 3.你使用过 / 编写过前端脚手架么，它的（其中一个）原理是什么	考察候选人对于使用前端脚手架实现工程自动化的理解	脚手架定义比较宽泛 基本上是一套工具，可以帮助开发人员快速简便的通过配置生成模板项目，管理项目 总体上以提升前端开发便捷性和舒适性为主的工具，同时还可以帮助统一团队不同项目的相关配置（如 git \）	对脚手架有了解 → 及格 对某个脚手架原理属性/自己编写过 → 优秀	All	严田	
	Tag: 测试、开放题 1.什么是单元测试、什么是集成测试、什么是回归测试 2.有做过单元测试么，要怎么做单元测试，常用框架有哪些 3.（某一个）单元测试框架原理是什么	考察候选人对测试相关知识的掌握	**单元测试**（英語：Unit Testing）又称为模块**测试**，是针对程序模块（软件设计的最小单位）来进行正确性检验的**测试**工作。 程序**单元**是应用的最小可**测试**部件。在过程化编程中，一个**单元**就是单个程序、函数、过程等；对于面向对象编程，最小**单元**就是方法，包括基类（超类）、抽象类、或者派生类（子类）中的方法 **集成测试**，也叫组装**测试**或联合**测试**。 在单元**测试**的基础上，将所有模块按照设计要求（如根据结构图）组装成为子系统或系统，进行**集成测试。确保合在一起的各个模块能按照预期工作。** **回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。注意回归测试要包含全测试用例。** 关于怎么做单测，有很多框架可以做到。第三问可以结合这一问回答的单测框架进行询问	明白第一问对三个测试指什么，了解怎么做单测 → 及格 明白单测框架原理/能说清楚怎么实现一个单测框架 → 优秀	>=中阶	严田	
性能	【陈述题】如何判断当前页面有没有内存泄漏？	1、考察候选人对浏览器环境页面内存泄漏排查处理的经验	1、是否有提到chrome或其他浏览器的devTools 2、是否能进一步讲解通过performance、memory判断内存使用情况	中阶： 至少提到1个，全提到+分 高阶： 能详细的描述出这两种方案中详细的排查细节	>=中阶	左现金	
	【陈述题】请列举哪些情况/哪种写法会导致内存泄漏	1、考察候选人对浏览器环境内存管理的理解 2、考察候选人在大型单页应用中是否有内存泄漏的意识及规避方法	1、在单页应用中向window挂在的内容过多，过大 2、闭包引起的内存泄漏 3、定时器引起的内存泄漏 4、事件绑定引起的 5、缓存引起的 6、DOM节点相关的	中阶： >=1 及格；>=3 优秀 高阶： >=3及格	>=中阶	左现金	

【笔试题】请写出至少一种引起内存泄漏的编码方式	1、考察候选人对浏览器环境内存泄漏的实际编码经验	1、在单页应用中向window挂在的内容过多，过大 `function createGlobalVariables() {` `  leaking1 = 'I leak into the global scope'; // assigning value to the undeclared variable` `  this.leaking2 = 'I also leak into the global scope'; // 'this' points to the global object` `};` `createGlobalVariables();` `window.leaking1; // 'I leak into the global scope'` `window.leaking2; // 'I also leak into the global scope'` 2、闭包引起的内存泄漏 `function outer() {` `  const potentiallyHugeArray = [];` `  return function inner() {` `  potentiallyHugeArray.push('Hello'); // function inner is closed over the potentiallyHugeArray variable` `  console.log('Hello');` `  };` `};` `const sayHello = outer(); // contains definition of the function inner` `function repeat(fn, num) {` `  for (let i = 0; i < num; i++){` `  fn();` `  }` `}` `repeat(sayHello, 10); // each sayHello call pushes another 'Hello' to the potentiallyHugeArray` `// now imagine repeat(sayHello, 100000)` 3、定时器引起的内存泄漏 `function setCallback() {` `  const data = {` `  counter: 0,` `  hugeString: new Array(100000).join('x')` `  };` `  return function cb() {` `  data.counter++; // data object is now part of the callback's scope` `  console.log(data.counter);` `  }` `}` `setInterval(setCallback(), 1000); // how do we stop it?` 4、事件绑定引起的 `const hugeString = new Array(100000).join('x');` `document.addEventListener('keyup', function() {` `// anonymous inline function - can't remove it` `  doSomething(hugeString); // hugeString is now forever kept in the callback's scope` `});` 5、缓存引起的	中阶： >=1 及格；>=3 优秀 高阶： >=3及格	>=中阶	左现金

			let user_1 = { name: "Peter", id: 12345 }; let user_2 = { name: "Mark", id: 54321 }; const mapCache = new Map(); function cache(obj){ if (!mapCache.has(obj)){ const value = `${obj.name} has an id of ${obj.id}`; mapCache.set(obj, value); return [value, 'computed']; } return [mapCache.get(obj), 'cached']; } cache(user_1); // ['Peter has an id of 12345', 'computed'] cache(user_1); // ['Peter has an id of 12345', 'cached'] cache(user_2); // ['Mark has an id of 54321', 'computed'] console.log(mapCache); // ((...) => "Peter has an id of 12345", (...) => "Mark has an id of 54321") user_1 = null; // removing the inactive user //Garbage Collector console.log(mapCache); // ((...) => "Peter has an id of 12345", (...) => "Mark has an id of 54321") // first entry is still in cache 6、DOM节点相关的 function createElement() { const div = document.createElement('div'); div.id = 'detached'; return div; } // this will keep referencing the DOM element even after deleteElement() is called const detachedDiv = createElement(); document.body.appendChild(detachedDiv); function deleteElement() { document.body.removeChild(document.getElementById('detached')); } deleteElement(); // Heap snapshot will show detached div#detached				
	【陈述题】影响前端渲染性能的因素有哪些?	1、考察候选人对浏览器渲染过程的了解 2、考察候选人对渲染性能处理的经验和知识	1、是否有提到关键渲染路径（加分项） 2、是否会提到渲染线程，进而提到单线程模型 3、是否会提到重排、重绘，以及引起重排、重绘的因素； 4、是否会提到requestAnimationFrame 5、实现动画时是否会提到利用css的transform、transition实现，是有有提到硬件加速（及如何启用硬件加速） 6、是否会提到浏览器渲染到多层，最后执行层合并的（加分项） 7、是否会提到transition实现位移比position效率更高及具体原因（加分项）	中阶： 提到 3 及格 高阶： 提到 2、3、5及格	>=中阶	左现金	

	【陈述题】客户反馈某一个页面打开慢、打开后操作反应会变慢、有时候会有卡死的现象，这种情况下应该从哪些方面入手排查？如何解决?	1、考察候选人分析问题的能力（有没有关注到几个重点：网络加载、FPS低、渲染进程被占用时间过长） 2、考察候选人解决不同类型性能问题的经验和知识	排查思路： 1、是否会先提到了解问题“什么时候开始出现的”、“出现的时机”、“频次”等前置问题（加分项） 2、是否会提到网络问题或资源加载或接口访问到问题排查，具体如何排查 3、是否会提到有可能由于数据量大导致解析、内存占用高的问题，具体如何排查 4、是否会提到渲染线程被占用过多，具体如何排查		>=中阶	左现金	
	【陈述题】前端性能优化的思路	话题较大，属于开放问题 1、考察候选人的思路是否清晰（可优化点太多，是否有归纳、总结意识） 2、考察候选人综合性能优化处理经验和知识	1、网络连接/加载方向 2、资源/内容大小方向 3、CSS方向 4、JS执行方向 5、按需、延迟的产品策略方向	中阶： 能描述出3个以上方向及格，细节描述到位、思路清晰+分 高阶段： 思路清晰、能提到个3方向，且细节描述清晰及格	>=中阶	左现金	
	【陈述题】请描述对「时间复杂度」/「空间复杂度」的理解	1、考察候选人对这两个指标的理解	参见：https://zhuanlan.zhihu.com/p/50479555		ALL	左现金	
Server端	1.1 请介绍一下Node事件循环的流程	考察候选人对「事件循环」流程的理解	1. 在进程启动时，Node便会创建一个类似于while(true)的循环，每执行一次循环体的过程我们成为Tick。 2. 每个Tick的过程就是查看是否有事件待处理。如果有就取出事件及其相关的回调函数。然后进入下一个循环，如果不再有事件处理，就退出进程。	答出及格	ALL	闪朔	
	1.2 在每个tick的过程中，如何判断是否有事件需要处理呢?		1. 每个事件循环中有一个或者多个观察者，而判断是否有事件需要处理的过程就是向这些观察者询问是否有要处理的事件。 2. 在Node中，事件主要来源于网络请求、文件的I/O等，这些事件对应的观察者有文件I/O观察者，网络I/O的观察者。 3. 事件循环是一个典型的生产者/消费者模型。异步I/O，网络请求等则是事件的生产者，源源不断为Node提供不同类型的事件，这些事件被传递到对应的观察者那里，事件循环则从观察者那里取出事件并处理。 4. 在windows下，这个循环基于IOCP创建，在linux下则基于多线程创建	答出1及格，2，3，4加分	ALL	闪朔	
	2.1 请简述一下node的多进程架构	考察候选人对node多进程的理解和应用	面对node单线程对多核CPU使用不足的情况，Node提供了child_process模块，来实现进程的复制，node的多进程架构是主从模式	及格		闪朔	
	2.2 创建子进程的方法有哪些，简单说一下它们的区别		1. spawn()： 启动一个子进程来执行命令 2. exec(): 启动一个子进程来执行命令，与spawn()不同的是其接口不同，它有一个回调函数获知子进程的状况 3. execFlie(): 启动一个子进程来执行可执行文件 4. fork(): 与spawn()类似，不同电在于它创建Node子进程需要执行js文件 5. spawn()与exec()、execFile()不同的是，后两者创建时可以指定timeout属性设置超时时间，一旦创建的进程超过设定的时间就会被杀死 6. exec()与execFile()不同的是，exec()适合执行已有命令，execFile()适合执行文件。	答出3个及格，越多越加分		闪朔	
	2.3 请问实现一个node子进程被杀死，然后自动重启代码的思路		在创建子进程的时候就让子进程监听exit事件，如果被杀死就重新fork一下 `var createWorker = function(){` `  var worker = fork(__dirname + 'worker.js')` `  worker.on('exit', function(){` `  console.log('Worker' + worker.pid + 'exited');` `  // 如果退出就创建新的worker` `  createWorker()` `  })` `}`	加分		闪朔	
	2.4 在2.3基础上，实现限量重启，比如我最多让其在1分钟内重启5次，超过了就报警给运维		1. 思路大概是在创建worker的时候，就判断创建的这个worker是否在1分钟内重启次数超过5次 2. 所以每一次创建worker的时候都要记录这个worker 创建时间，放入一个数组队列里面，每次创建worker都去取队列里前5条记录 3. 如果这5条记录的时间间隔小于1分钟，就说明到了报警的时候了	加分		闪朔	
	3.1 如何查看v8的内存使用情况	V8的垃圾回收机制	使用process.memoryUsage(),返回如下 `{` `  rss: 4935680,` `  heapTotal: 1826816,` `  heapUsed: 650472,` `  external: 49879` `}` **heapTotal** 和 **heapUsed** 代表V8的内存使用情况。**external** 代表V8管理的，绑定到Javascript的C++对象的内存使用情况。**rss**, 驻留集大小, 是给这个进程分配了多少物理内存(占总分配内存的一部分) 这些物理内存中包含堆，栈，和代码段。	操作方法及格，具体含义加分		闪朔	

	3.2 V8的内存限制是多少，为什么V8这样设计		64位系统下是1.4GB， 32位系统下是0.7GB。因为1.5GB的垃圾回收堆内存，V8需要花费50毫秒以上，做一次非增量式的垃圾回收甚至要1秒以上。这是垃圾回收中引起Javascript线程暂停执行的事件，在这样的花销下，应用的性能和影响力都会直线下降。	限制数字及格，原因加分		闪朔	
	3.3 v8的内存分代和回收算法		在V8中，主要将内存分为新生代和老生代两代。新生代中的对象存活时间较短的对象，老生代中的对象存活时间较长，或常驻内存的对象。 3.3.1 新生代中的对象主要通过Scavenge算法进行垃圾回收。这是一种采用复制的方式实现的垃圾回收算法。它将堆内存一份为二，每一部分空间成为semispace。在这两个semispace空间中，只有一个处于使用中，另一个处于闲置状态。处于使用状态的semispace空间称为From空间，处于闲置状态的空间称为To空间。 1. 当开始垃圾回收的时候，会检查From空间中的存活对象，这些存活对象将被复制到To空间中，而非存活对象占用的空间将会被释放。完成复制后，From空间和To空间发生角色对换。 2. 应为新生代中对象的生命周期比较短，就比较适合这个算法。 3. 当一个对象经过多次复制依然存活，它将会被认为是生命周期较长的对象。这种新生代中生命周期较长的对象随后会被移到老生代中。 3.3.2 老生代主要采取的是标记清除的垃圾回收算法。与Scavenge复制活着的对象不同，标记清除算法在标记阶段遍历堆中的所有对象，并标记活着的对象，只清理死亡对象。活对象在新生代中只占叫小部分，死对象在老生代中只占较小部分，这是为什么采用「标记清除」算法的原因。	能答出新生代与旧生代及各自算法及格 谈的深入，加分			
	3.4 标记清除算法的问题		主要问题是每一次进行标记清除回收后，内存空间会出现不连续的状态 1. 这种内存碎片会对后续内存分配造成问题，很可能出现需要分配一个大对象的情况，这时所有的碎片空间都无法完成此次分配，就会提前触发垃圾回收，而这次回收是不必要的。 2. 为了解决碎片问题，标记整理被提出来。就是在对象被标记死亡后，在整理的过程中，将活着的对象往一端移动，移动完成后，直接清理掉边界外的内存。	答出问题及格，答出解决方案加分			
	4 Nginx 有哪些负载均衡策略?		1、轮询（默认） 每个请求按时间顺序逐一分配到不同的后端服务器，如果后端某个服务器宕机，能自动剔除故障系统。 2、权重 weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。其次是为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。 3、ip_hash 每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题 4、fair（第三方插件） 对比 weight、ip_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，响应时间短的优先分配。必须安装upstream_fair模块。 5、按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，可以进一步提高后端缓存服务器的效率。必须安装Nginx的hash软件包	答出内容及格 答出5种方案彼此间的不同，及使用场景加分			
	5、为什么要做动静分离?		网站优化的重要点在于静态化网站，网站静态化的关键点则是是动静分离，动静分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们则根据静态资源的特点将其做缓存操作。 让静态的资源只走静态资源服务器，动态的走动态的服务器 Nginx的静态处理能力很强，但是动态处理能力不足，因此，在企业中常用动静分离技术。 1. 对于静态资源比如图片，js，css等文件，我们则在反向代理服务器nginx中进行缓存。这样浏览器在请求一个静态资源时，代理服务器nginx就可以直接处理，无需将请求转发给后端服务器tomcat。 2. 若用户请求的动态文件，比如servlet,jsp则转发给Tomcat服务器处理，从而实现动静分离。这也是反向代理服务器的一个重要的作用。	答出原因，及格 答出具体实现，加分			

笔试题

笔试题要求候选人能写出可执行出正确结果的代码，可适当引导候选人进行一些思路上的提示

数组转换成树形结构对象

输入：

```
[
  {
    id: 1,
    pid: 0,
    name: 'body'
  },
```

```
  {
    id: 2,
    pid: 1,
    name: 'title'
  },
  {
    id: 3,
    pid: 1,
    name: 'div'
  },
  {
    id: 4,
    pid: 3,
    name: 'span'
  },
  {
    id: 5,
    pid: 3,
    name: 'icon'
  },
  {
    id: 6,
    pid: 4,
    name: 'subspan'
  }
]

输出:
[
  {
    "id": 1,
    "pid": 0,
    "name": "body",
    "children": [
      {
        "id": 2,
        "pid": 1,
        "name": "title"
      },
      {
        "id": 3,
        "pid": 1,
        "name": "div",
        "children": [
          {
            "id": 4,
            "pid": 3,
            "name": "span",
            "children": [
              {
                "id": 6,
                "pid": 4,
                "name": "subspan"
              }
            ]
          },
          {
            "id": 5,
            "pid": 3,
            "name": "icon"
          }
        ]
      }
    ]
  }
]
```

```
/**
 * 两数相加
 * 给定一个包含 n 个整数的数组 nums，判断 nums 中是否存在两个元素 a，b ，使得 a + b = target ？ 找出所有满足条件且不重复的二元组。
 * 注意：答案中不可以包含重复的二元组。
 * 例如, 给定数组 nums = [-1, 0, 1, 2, -2, -4]，target = 0
 * 满足要求的二元组集合为：[[-1, 1], [2, -2]]
 * 要求时间复杂度为O(n)
 */
```

```
/**
 * 三数相加
 * 给定一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c =
 * target ？ 找出所有满足条件且不重复的三元组。
 * 注意：答案中不可以包含重复的三元组。
 * 例如, 给定数组 nums = [-1, 0, 1, 2, 4, -4]，target = 0
 * 满足要求的三元组集合为：
 * [
 *   [-1, 0, 1],
 *   [0, 4, -4]
 * ]
 */
```

```
/**
 * 实现一个累加器
 * add(1,2)(3).sumOf()
 * add(1)(2,3).sumOf()
 * add(1)(2)(3).sumOf()
 * add(1,2,3).sumOf()
 */
```

```
/* 实现函数compose，compose接受多个函数作为参数，并返回一个新的函数，新的函数会从右向左依次执行原函数， 并且上一次结果的返回值将会作为下一个函数的参数。*/

function a(msg) {
  return msg + "a";
}
function b(msg) {
  return msg + "b";
}
function c(msg) {
  return msg + "c";
}

const f = compose(
  a,
  b,
  c
);
console.log(f("hello")); // 打印hellocba
```

实现函数curry，该函数接受一个多元（多个参数）的函数作为参数，然后一个新的函数，这个函数 可以一次执行，也可以分多次执行。

eg:

```
// test
function test(a, b, c) {
  console.log(a, b, c);
}

const f1 = curry(test)(1);
const f2 = f1(2);
f2(3);
```

```
/**
 * 写一个函数实现以下数据转换：
 * {a: {b: {c:1}}, d:[1,2]}
 * 转换成：
 * {'a.b.c': 1,  'd[0]':1, 'd[1]':2}
 */
```

如下表格，点击 .date 后使表格按日期排序，当前是正序则改为倒序，当前是倒序则改为正序，要求以原生 JavaScript 实现。

```
<table id="c">
    <thead>
        <tr>
            <th class="date">日期</th>
            <th class="total">总次数</th>
        </tr>
    </thead>
    <tbody id="tbody">
        <tr>
            <td>2017年10月23日</td>
            <td>68,112</td>
        </tr>
        <tr>
            <td>2017年8月6日</td>
            <td>68,020</td>
        </tr>
        <tr>
            <td>2017年11月11日</td>
            <td>69,433</td>
        </tr>
        <tr>
            <td>2016年5月12日</td>
            <td>69,699</td>
        </tr>
        <tr>
            <td>2017年1月18日</td>
            <td>42,565</td>
        </tr>
    </tbody>
</table>
<script>

</script>
```

实现一个 calc 方法，可以将输入的数拆解为尽可能多的乘数，所有数相乘等于输入数。

```
/**
 * @param {number} n 乘积
 * @return {Array} 拆解后的乘数
 */
function calc(n) {
}

console.log(calc(2));
// [2]

console.log(calc(8));
// [2, 2, 2]

console.log(calc(24));
// [2, 2, 2, 3]

console.log(calc(30));
// [2, 3, 5]
```

```
/**
 * 有几个图片资源，存在数组urls中，然后需要实现一个函数load，要求尽快地将urls里的图片加载，且同时加载的图片数量不超过3个
 */
var urls = [
  'https://sensorsdata.cn/assets/img/logo/W1_73d7a9b.png',
  'https://sensorsdata.cn/assets/img/logo/wpf_346a7fd.jpg',
  'https://sensorsdata.cn/assets/img/logo/W3_5a8ea2c.png',
  'https://sensorsdata.cn/assets/img/logo/W5_e365587.png',
  'https://sensorsdata.cn/assets/img/logo/W7_c308f43.png',
  'https://sensorsdata.cn/assets/img/logo/W6_7ca4634.png',
  'https://sensorsdata.cn/assets/img/logo/W8_4c59d35.png',
  'https://sensorsdata.cn/assets/img/logo/W9_4fe8fa4.png',
]
```

1、实现一个request方法，具有以下能力

- 支持设定强缓存（重发请求）
- 支持设定协商缓存（重发请求）
- 支持根据key或者一个函数的返回来支持内存缓存（不重发请求，内存里面拿）

题目要求:

- 不考虑兼容性
- 只考虑get、post方法

2、实现一个request方法，具有以下能力:

- 支持全局配置能力，全局配置能配置请求分类以及请求分类的优先级、配置baseUrl、headers选项
- 同时在发起的请求只能有6个
- 能控制请求的优先级，优先级高的请求先发起

题目要求:

- 不考虑兼容性
- 只实现get、post方法

3、实现Promise.allSettled方法，要求写两遍，一遍不使用async、await，一遍使用async、await

要求:

- 不考虑兼容性

```
Promise.prototype.allSettled = (funcArr) => {
  return new Promise((resolve) => {
    let sttled = 0
    let result = []
    for(let index = 0;index<funcArr.length;index++){
      const element = funcArr[index]
      element
      .then(res => {
        result[index] = {
          status: 'fulfilled',
          value: res
        }
      })
      .catch(err => {
        result[index] = {
          status: 'rejected',
          reason: err
        }
      })
      .finally(() => { ++sttled === funcArr.length && resolve(result) })
    }
  })
}

const allSettled = async funcArr => {
  let res = await Promise.all(
    funcArr.map(function(promise) {
      return promise
        .then(function(value) {
          return { state: "fulfilled", value: value };
        })
        .catch(function(reason) {
          return { state: "rejected", reason: reason };
        });
    })
  );
  return res;
};
var promise1 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(1);
  }, 2000);
});
var promise2 = new Promise((resolve, reject) => {
  setTimeout(() => {
    reject(2);
  }, 1000);
});
let res = await allSettled([promise2, promise1]);
console.log(res);
```

4、实现一个简易模块化机制：要求如下：

- 模块中使用module.export进行导出
- 使用require()进行加载

```
//操作文件的模块
let fs = require('fs');
// 处理路径的模块
let path = require('path');
// 虚拟机模块，沙箱运行，防止变量污染
let vm = require('vm');

// 创建module构造函数
```

```
function Module(id) {
  this.id = id;
  this.export = {};
}

// 根据绝对路径进行缓存的模块对象
Module._cacheModule = {};

// 存放闭包字符串
Module.wrapper = [
  '(function (exports, require, module, __filename, __dirname) {',
  '})',
];

// 将我们读到js的内容传入,组合成闭包字符串
Module.wrap = function (script) {
  return Module.wrapper[0] + script + Module.wrapper[1];
};

// 处理对应后缀名模块
Module._extensions = {
  '.js': function (module) {
    // 对于js文件，读取内容
    let content = fs.readFileSync(module.id, 'utf8');
    // 给内容添加闭包, 后面实现
    let funcStr = Module.wrap(content);
    // vm沙箱运行, node内置模块，前面我们已经引入， 将我们js函数执行，将this指向 module.export
    vm.runInThisContext(funcStr).call(
      module.export,
      module.export,
      req,
      module
    );
  },
  '.json': function (module) {
    // 对于json文件的处理就相对简单了，将读取出来的字符串转换未JSON对象就可以了
    module.export = JSON.parse(fs.readFileSync(module.id, 'utf8'));
  },
};

// 将引入文件处理为绝对路径
Module._resolveFilename = function (p) {
  // 以js或者json结尾的
  if (/\.js$|\.json$/.test(p)) {
    // __dirname当前文件所在的文件夹的绝对路径
    // path.resolve方法就是帮我们解析出一个绝对路径出来
    return path.resolve(__dirname, p);
  } else {
    // 没有后后缀  自动拼后缀
    // Module._extensions 处理不同后缀的模块
    let exts = Object.keys(Module._extensions);
    let realPath; // 存放真实存在文件的绝对路径
    for (let i = 0; i < exts.length; i++) {
      // 依次匹配对应扩展名的绝对路径
      let temp = path.resolve(__dirname, p + exts[i]);
      try {
        // 通过fs的accessSync方法对路径进行查找，找不到对应文件直接报错
        fs.accessSync(temp);
        realPath = temp;
        break;
      } catch (e) {}
    }
    if (!realPath) {
      throw new Error('module not exists');
    }
    // 将存在绝对路径返回
    return realPath;
  }
};

// 根据传入的模块，尝试加载模块方法
```

```
function tryModuleLoad(module) {
  // 前面我们已经提到 module.id 为模块的识别符，通常是带有绝对路径的模块文件名
  // path.extname 获取文件的扩展名
  /* let ext = path.extname(module.id);
  // 如果扩展名是js 调用js处理器 如果是json 调用json处理器
  Module._extensions[ext](module); // export 上就有了数组 */
  let ext = path.extname(module.id); //扩展名
  // 如果扩展名是js 调用js处理器 如果是json 调用json处理器
  Module._extensions[ext](module); // export 上就有了数组
}

// 模块加载
Module._load = function (f) {
  // 相对路径,可能这个文件没有后缀，尝试加后缀
  let fileName = Module._resolveFilename(f); // 获取到绝对路径
  // 判断缓存中是否有该模块
  if (Module._cacheModule[fileName]) {
    return Module._cacheModule[fileName].export;
  }
  let module = new Module(fileName); // 没有就创建模块
  Module._cacheModule[fileName] = module; // 并将创建的模块添加到缓存

  // 加载模块
  tryModuleLoad(module);
  return module.export;
};

// 测试代码
function req(p) {
  return Module._load(p); // 加载模块
}

let strjs = req('./test.js');
let strjson = req('./test.json');
console.log(strjs);
console.log(strjson);
```

5、实现一个tooltip的React、Vue组件，要求如下：

- 支持设定弹出方向
- 弹框带有箭头
- 弹框是半透明黑色

要求：

- 不考虑兼容性

参考：https://stackblitz.com/edit/react-9nikkw-avjvxj?file=index.js

6、实现一个点线图组件，React、Vue。要求如下：

- 传入组件的数据是nodes和links一级自己想定义的任何配置，nodes包含基本的id、name字段，links包含source和target字段
- 对布局的唯一要求是点不重合
- node渲染的时候支持使用html元素渲染
- 可以使用d3

要求：

- 不考虑兼容性