2023年react高频面试题讲解

1. 国内前端岗位形势详解

1.1. 2022发展总结&2023发展趋势

1.1.1. 前端领域层级划分

如果从前端技术的角度出发，2023年应该更专注于前端的特点与擅长。如果以用户为中心点来对整个IT领域工种进行划分的话，前端无疑是最贴近用户的。那么，个人认为前端的关注点应该落脚到“以用户为中心，坚持前端核心技术为基石，关注服务与形态的结合”这样一个基本原则下。

这里，如果将整个前端领域的一些涉猎按照层次化的方案进行梳理，大致可以分出：交付层、基建层、容器层、系统层四个层次：

1. 交付层：主要涉及应用代码、业务逻辑、用户体验、领域模型等相关的开发，这也是前端工程师入门的基础的基础，作为前端开发的基本要求，当然这也相对属于前端领域的浅水区；
2. 基建层：这主要包括框架/库、工程效能、安全兜底、性能稳定等相关的建设，这里也是绝大多数有一定开发经验的前端开发最喜欢探索和涉猎的层级，其仍然以前端常见的开发为主，但又比业务交付层更加抽象一些，通常来说，大部分前端工程师都会这一层次去游走，既不脱离业务，又未涉及底层；
3. 容器层：这通常包括：浏览器内核、运行时环境、标准、协议等相关的深入研究，这也是最不为大多数前端所能触及的部分，但这确实是前端领域需要攻克的难关，虽然下了功夫见效也不大，但是对于前端天花板的突破，其实是一个比较重要的层次；
4. 系统层：这里目前并不属于前端的范畴了，一般来说不会用类js语言去书写操作系统。因而对于前端工程师来说，突破领域主要在于前三个层次，即：交付层、基建层、容器层，保证交付能力，拓展基建能力，涉猎容器能力，这样在纵深层次上也会能得到一定的突破；

1.1.2. 前端细分发展趋势分析

1.1.2.1. 工程化

工程化方向是去年整体更新最多的一个领域，除了前端本身发展过程的历史背景外，工程化作为IT领域的一个重要方向，同时也作为与整个技术管理等相互关联的重要领域，所以前端绝对不会放过这么一个热门方向的。下面是比较重要的更新简要回顾及探讨下工程化的过去与23年的展望：

1. 构建：作为工程化领域最为重要的一个节点，肯定是群雄逐鹿的主战场。对前端构建而言，目前市场占有量仍然以Webpack为主，但是随着对“bundle”和“bundless”方案的探讨；以Vite等为主的构建方案，对构建方式也提出了自己的一套理论方案，但bundless的方案需要有一定的前提支撑。目前，Vite等构建方案也在构建方案中占据了一席之地，同时诸多跨语言构建方案也一直冲击着工程化的市场。因而，个人认为23年工程化构建领域的混战仍会持续，但真要替代Webpack作为生产环境的主流方案，目前来看明年还不太容易达到，毕竟整体的生态而言，Webpack还是有很强的体系支撑的；
2. 模块化：自从诞生了ESM，前端模块化方案逐步从IIFE、UMD、AMD、CommonJS、ESM等诸多模块化方案，逐步收敛到了ESM和CJS之争，并且这个争斗也延伸到了打包器的站队问题中。模块化，作为前端界的阿基里斯之踵，一直以来困扰着诸多前端开发者，但官方原生支持了之后，前端工程链路也都同步进行了转向。个人认为，短期内还是会存在多模块化共存的现象，但长期看好ESM一统江湖的可能，这也给以天生支持ESM的打包器，诸如：Rollup、Vite等一些底气和支撑；
3. Monorepo：作为工程管理方案的一种技术理念，monorepo并不是一种技术方案，而其实是一种技术理念。在前端工程化中的monorepo，则以lerna、pnpm、nx等解决方案为主，同时配合着turbopack推出的turborepo也是monorepo整个仓库管理方案的大闭环解决思路。相较于multirepo及monolith的方案，虽然monorepo有着共享组合的方便，但是对于权限体系管理却有着一定的弊端。个人认为，前端领域的monorepo还需要去寻找一个能够平衡共享与隔离问题的解决方案，最终才能真正的解决工程链路中的体验问题；
4. 包管理：前端的包管理，不同于后端的包体系，除了之前明显的npm地域问题外，还涉及到对应的更新变化问题。虽然前端繁荣发展离不开开源package的层出不穷，但对于整个包体系的管理确实可以借鉴下其他语言的设计思路，既能灵活使用，又能规范可依；

综上，整个工程化领域在22年发展还是很迅猛的，但还是希望能够各位巨佬针对核心问题进行突破创新。可以预见，23年仍会是工程化的一年，毕竟拉来其他领域的开发者共建共享，前端才能更有话语权。

1.1.2.2. Node.js

Node.js方向相对来说还是主要落脚于前端自己的后端化之路，也是其最适合的地方，这里最主要的方向仍然是Serverless的前端贡献。

1. NoSlate：NoSlate是阿里开源的一款Serverless的解决方案，从调度、存储、运行时等多个方面进行优化自研，提出了一套更加简洁轻量高效的方案；
2. Winter：全称是Web-interoperable Runtime，其本质也是一个V8 Worker，经过标准化后，其基本可以作为FaaS函数运行时标；

综上，整个Node.js领域的发展相对来说还是在Serverless细分领域的建树，这其实也是Node.js一个比较好的应用方向，毕竟真正使用Node.js来作为传统后端开发还是有一定的局限性的，相信23年仍然会有和云原生领域更深入的结合。

1.1.2.3. 跨端

整个跨端方向大体来说，去年几乎都不约而同的选择了基于容器的方案，借鉴云原生领域相关的理念，也是通解跨端的一种不错的思路。

1. Lath：Lath是阿里的一款纯前端容器，提供从事件处理等多跨平台的方案，对于多场景进行容器优化。
2. Tauri：作为Electron的竞品，其本身是基于Rust和Webview2进行相关的构建的，虽然不算是一种容器方案，但是对于内存+渲染的组合，其还是对Electron的痛点进行了一定程度的改进。

综上，整个跨端领域相对来说还是专注于“Write Once, Run EveryWhere”的理念，但确实很难真正的做到。个人认为，跨端方向可以专注于某几种场景的通用，真正的全部通用是没有意义的，因为兼容越多，意味着polyfill也就越多，有时候却是得不偿失的。

1.1.2.4. 智能化

智能化方向最突出的进展莫过于低代码领域的相关发展，本身将低代码划入到智能化领域确实稍微有些牵强，但低代码的自动化实现确实是可以借助于AI的相关能力的，因而也将其划入到智能化方向；除了本身的D2C发展外，最近新出的ChatGPT在前端领域也有一定的发展前景。

1. D2C：作为前端领域传统的智能化细分领域，利用机器视觉自动生成代码，关键在于对不同模型的优化。
2. 低代码：低代码领域的规范与约定形成更为重要，阿里开源的LowCodeEngine可以作为低代码构建的一个参考，另外与智能化方向的自动生成其实可能更有研究价值。
3. ChatGPT：作为22年年底最火的AI场景，对于回答等的薅羊毛行为，想必会很快出台禁止方案。但是在前端领域，对于代码方案等进行相关指导还是有一定的借鉴意义的。

综上，整个智能化领域除了传统的D2C方案外，可以考虑NLP相关发展在前端领域的落地与创新。

1.1.2.5. 互动方向

互动方向应该是目前前端最为神秘的一个方向，充满了很多未知，个人认为，最主要的在于对新交互方式的探索与场景运用。

1. 元宇宙：前端作为一种靠近用户的工种，其本身技术也会提供诸如：XR的形态，在虚拟人物生成等方面，确实还是会有一些研究可能存在；
2. Web3：前端作为Web领域的重要一环，在第三世代中，肯定也会有重要的运用场景。个人认为，对于区块链相关的上层应用，如DApp等，可能还是有一些发展方向的；

综上，互动方向应该是23年变化最快的一个领域，应该也是最有可能产生新突破的方向，目前还充满很多机会与挑战，有想法的同学可以提前布局。

1.1.2.6. 中后台

随着单页的瓶颈出现，中后台方向又出现了“分久必合，合久必分”的态势，不论是微前端还是Islands架构，其都有一种新瓶装旧酒的感觉。

1. 微前端：目前通用的微前端方案大都以“类SPA”形式进行创建，其本身在接入其他SPA时有着天然的弊端。因而，真正意义上的“微前端”还尚未出现，私以为或许考虑借鉴下微服务的容器化思维，来真正的实现“微”的效果。
2. Islands：Islands架构的本质其实是多种渲染方式的选择，从MPA到SPA，又从SPA回归到MPA。其实，可以基于场景的不同进行不同的选择，资源的合理使用来提供最佳的用户体验。

综上，中后台方向基本还是基于模板的选择构建，技术的选择并不一味的向前突进，有时走下复古风格，也是另一种思考和体验。

1.1.2.7. 可视化方向

对于可视化方向，可以对于不同的特定场景进行底层构建，从而产生更适合特定领域的图形可视库。

1. 领域图形库：对于可视化领域而言，最重要的其实是图形库的选择和开发，但是对于不同层次的可视化方案，也是构建出基于特定领域的专有图形库，解决一类问题，从而有一定的突破和创新。
2. 3D图形库：大部分的图形库往往注重与二维图形库的创建，对于三维可视/编辑领域，也是有着十分广阔的场景的。

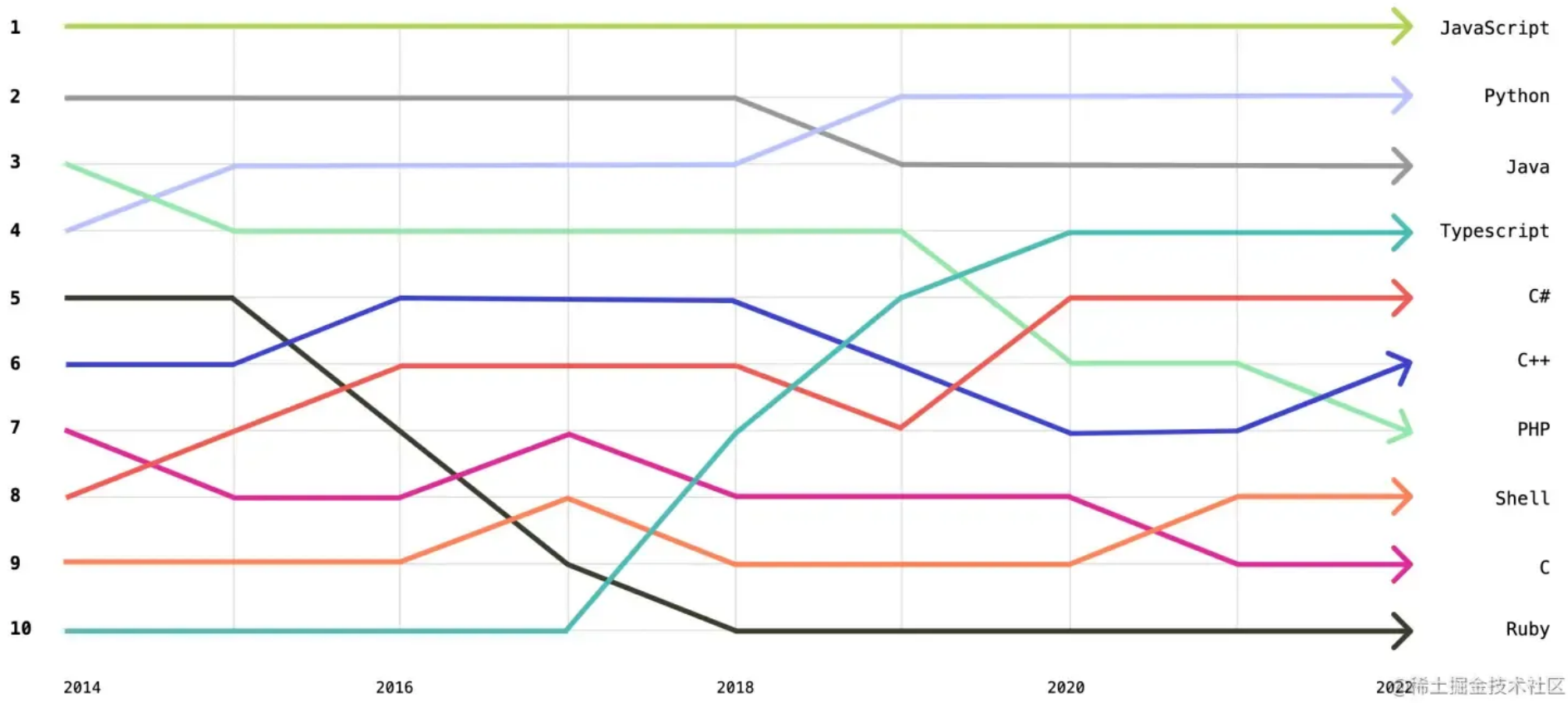
综上，可视化方向可以关注特定领域的底层建设，以及对应于三维图形库的突破与开发。

1.2. 2023年就业情况分析

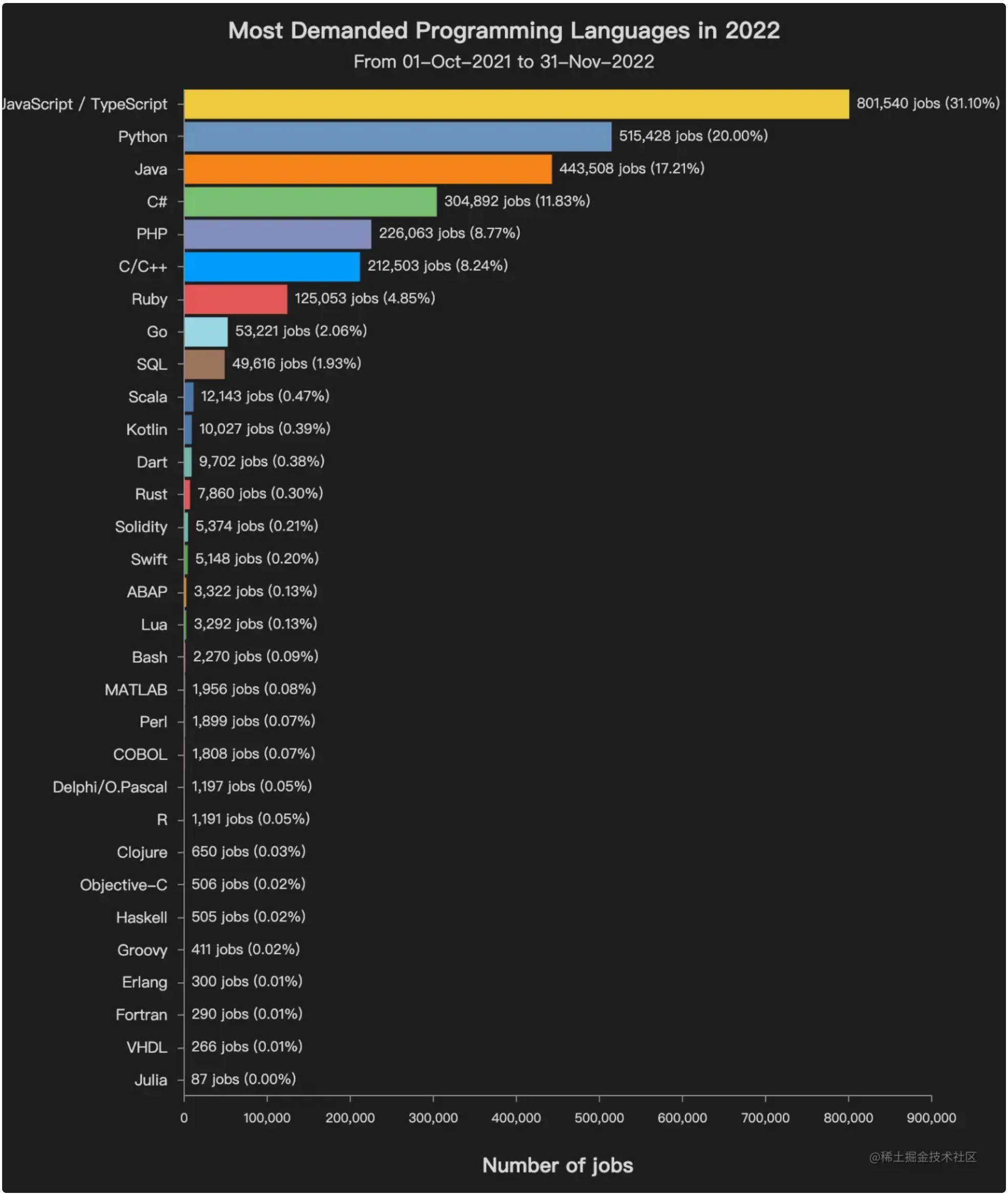
整体上，前端的状态是趋于成熟，已经过了第一波爆发式增长期，但它还在垂类细化领域不断发展中，所以我还是非常看好的。

就2022年所谓的就业形式而言，整个互联网局势都很差，从人才济济到“人才挤挤”，主要原因就是经济环境和行业发展遇到瓶颈。加上前端是互联网中的一个工种，这就意味着前端在互联网局势很差的情况下也会很差。

这里看一下截止至2022年各种语言的使用情况：



根据 `DevJobsScanner` 网站统计，时间是从 `Oct-2021` 到 `Nov-2022` 之间，从1200万个工作机会中进行筛选的记过。数据如下：



从图中，我们可以得知，`JavaScript / TypeScript`相关工作高居榜首，这意味着前端相关技能适用面非常广泛。

辩证的看，前端技能和前端职位数成正比，这是正常的。这一点国内和国外的是没有区别的，多端和跨端，`Node.js`等混合成为大前端，守好体验和效率的门卫，当然是必不可少。

1.2.1. 外包化严重

没有增长的公司大致是会维持或降本。对于技术来说，除了服务器等软硬件采购，就是人力成本。出于降本的考虑，一些能够由成本不高的人能够完成的活，就不会让高成本的人来做。所以说，外包化是整个互联网行业都在做的事儿，不单单只是前端。

对于前端来说，门槛不高，技术成熟度高的工作是最容易外包化的。比如`ToB`端相关工作外包化就严重。原因很简单用户是内部人员，页面不追求极致体验，甚至是能用就行。另外，技术上没有新框架，React这种框架使用上还是很好用的，所以ToB端CRUD能外包的就外包。

熟练外包确实是好的，但外包和正式比例短期内还是正式更多。大家也不必过度担心，java至今也没有外包比正式多，只是会让大家去追求专业度更高的事儿，这其实也是好事。

外包是一种常态化选择，不只是在裁员潮下面，外包和技术提效一直都有，只是在局势不好的时候，它会被放大，甚至引起很多人的恐慌。业务萎缩，挤掉些水分是正常的，但这不是行业坍塌，需求量依然还在的，对我们而言，更多的调整好心态，强大自己，技术过硬，与时俱进就好。

就未来讲，真正惨的事儿不只是外包化，而是正式员工干外包的活，外包被辞退，这也是非常有可能的。比如类似国企性质的公司，不方便辞退的公司，大概率只能这样选择。成本优化，很多时候是先挤掉水分，然后清理外包，正式员工加量不加价（降薪不好操作），正式流失，扛不住的时候再补外包。

1.2.2. 高级岗位变少

前端领域里所谓的架构师类(或者说专家类）比例还是非常多的。前端为什么会高级岗位很多，其实是2013到2019年的前端爆发期有关的。以前Java同学还能写jQuery，但到了React时代，搞定Webpack，sass，TypeScript等等，他们就显得力不从心了，专业前端都很痛苦，何况他们。所以那个时候非常细化招高级岗位，除了解决问题外，还有很多基建的建设。

随着基建的完善，比如create-react-app(cra)、Umi、Next.js这样的基建成熟，如果只是单纯做业务，我们还需要那么多高级岗位吗？明细是不会的。从成本角度看，高级岗位的成本，大家也是心里清楚的。所以我的判断是高级岗位会变少，很有可能会慢慢变成架构师角色，比例和Java等差不多。

高级岗位溢出的人，创业、转管理、转型也都是好的，学学Winter、Hax、Phodal、安晓辉等其实也挺好。像TL这种比例不大动，变化不大。优胜略汰，正常比例，只是流动量少，竞争会非常激烈。

1.2.3. 专业前端细分化

专业前端，依然是大多数，比如c端这种重体验的必须专业前端，还有垂类，比如互动游戏，3d，webrtc这种有专业难度的小众分类也必须专业前端，比如可视化编辑器，AFFiNE，QUill，X6这种都是需要专业前端的。

就目前的行情来看，发展中的企业依然是按照专业分工工作，成熟的公司更愿意搞全栈，降薪不好操作，就只能加量不加价。目前看，前端垂类，其实是最吃香的部分。

1.2.4. 岗位融合新机会

岗位融合，对于Retool带来革命性的交互方式，会颠覆很多角色的，包括前端、服务端、数据分析等。未来可能会出现低码工程师，或者类似全栈工程师这样的胶水类岗位，也是非常有可能的。大家把心态放宽，没工作是很难的，最怕的自己放弃自己，坚持每日精进，又怎么会被时代抛弃呢。

体力活和技术含量低的活儿慢慢被技术升级所替代，这在任何时代都是必然的事儿。

2. 面试优质前端岗需要如何准备

在具体讲解前，我们先来看看目前大厂职级的详细介绍。

2.1. 大厂职级详解

要求\级别	主要目标	核心能力	要求
P5	从学生转变为“打工人”	在别人知道下完成任务	技术：岗位基本技术&团队常用技术 业务：熟悉业务功能的处理逻辑 管理：熟悉项目流程
P6	成为独立自主的“项目能手”	独立负责端到端的项目任务	技术：熟练掌握端到端的工作流技术 业务：熟悉某业务的所有功能 管理：项目子任务推进
P7	成为让人信服的“团队专家”	指挥单个团队达到目标	技术：精通团队相关技术 业务：关注业务的整体情况 管理：指挥10人以内的小团队
P8	成为“跨团队指挥”	指挥多个团队达成目标	技术：精通领域相关技术 业务：熟悉多个业务或精通端到端业务 管理：核心是抓重点

2.2. 前端Job Model

类型	体系职能
技术–前端–开发	负责人机交互层的界面开发，实现交互功能
技术–前端–架构	熟悉业务领域与前端技术发展，负责前端类库框架、研发流程等基础体系的设计并推动实现，负责业务领域的前端技术选型并推动落地
技术–前端–数据可视化	负责常规统计图表、业务洞察分析、地理空间数据分析等相关引擎、服务、产品及生态建设
技术–前端–Node	基于Node进行web服务开发或工具开发
技术–前端–图像互动	负责互动/游戏化业务，2D&3D图像渲染等研发工作
前端体验	1. 体验度量：通过设计体验模型来度量产品体验功能 2. 体验优化：体验分析，如核心链路分析、体验验收&用户反馈等
前端工程化	1. 基础前端工程服务平台、前端上层链路研发平台、开发支撑平台 2. 前端研发工具，如WebIDE etc
跨端技术	1. 跨软硬件设备，IoT、跨操作系统、跨App、跨渲染容器（最常见，如webView，weex，rn etc） 2. 跨端生态
中后台技术	1. 基础UI组件，研发工具 2. low Code、no Code逻辑编排 etc
前端智能化	1. D2C（Design to Code 设计稿转代码） 2. 分析用户特征，推荐UI排版，千人千面
Serverless	1. 含义：Serverless = Server + less = 少/无服务器 2. FaaS：Function as a service 代码+相关依赖+配置
数据可视化	1. 基础前端工程服务平台、前端上层链路研发平台、开发支撑平台 2. 前端研发工具，如WebIDE etc
多媒体技术	1. 音视频基础、流媒体播放、视频剪辑等 2. 多端（web、Hybrid、小程序）直播间

2.3. 面试相关汇总

2.3.1. 面试内容

1. 中小厂 / 国企：八股文 + 项目
    a. 直接是八股文的原题，最多变式（手写防抖、节流；手写call、apply；手写Promise.all、Promise.allSettled）；
    b. 对项目经验要求度不高，对具体实现细节考察比较少，主要考察对基础的理解程度；
2. 大厂：技术深度广度挖掘：八股文 + 项目 + 算法
    a. 以八股文作为切入点，扩展至类似技术栈的具体实现，可能会涉及到框架的具体实现，看候选人对技术基础的理解程度：P6居多；
    b. 以项目具体经验作为切入点，根据项目经验延伸至前端项目的具体落地，结合市场上比较优秀的框架对比，主要考察技术面的广度和深度，以及是否能够解决行业内某一具体细分的问题，如工程化，脚手架，CI/CD等：P7居多；

2.3.2. 面试流程

- 中小厂：一般为3轮：技术面2轮+HR面1轮；
- 国企：一般为4轮：线上笔试1轮+技术面2轮+HR面1轮；
- 大厂：一般为4~5轮：技术面3轮+（交叉面1轮）+HR面1轮；

2.3.3. 技术深度广度拓展

- 前端框架：React、Vue、Svelte、Angular；
- Node框架：Next、Nest、Nuxt；
- 构建工具：Vite、esbuild、webpack、rollup；
- CSS in JavaScript：Styled component、twind、CSS module；
- 测试框架：Storybook、Jest；
- 移动端开发：React Native；
- 状态管理：Zustand、Redux、Vuex、MobX；

3. 面试常见问题&面试注意事项

3.1. 从用户输入网址到客户端展现，中间发生了什么过程?

> 目的在于是否有对性能优化的实践

1. 输入网址 --- 告诉浏览器你要去哪里
2. 浏览器查找DNS --- 网络世界是IP地址的世界，DNS就是ip地址的别名。从本地DNS到最顶级DNS一步一步的网上爬，直到命中需要访问的IP地址
    - DNS预解析 --- 使用CDN缓存，加快解析CDN寻找到目标地址（dns-prefetch）；
3. 客户端和服务器建立连接 --- 建立TCP的安全通道，3次握手
    - CDN加速 --- 使用内容分发网络，让用户更快的获取到所要内容；
    - 启用压缩 --- 在http协议中，使用类似Gzip压缩的方案（对服务器资源不足的时候进行权衡）；
    - 使用HTTP/2协议 --- http2.0针对1.0优化了很多东西，包括异步连接复用，头压缩等等，使传输更快；
4. 浏览器发送http请求 --- 默认长连接（复用一个tcp通道，短连接：每次连接完就销毁）
    - 减少http请求 --- 每个请求从创建到销毁都会消耗很多资源和时间，减少请求就可以相对来说更快展示内容；
        - 压缩合并js文件以及css文件
        - 针对图片，可将图片进行合并然后下载，通过css Sprites切割展示（控制大小，太大的话反而适得其反）
    - 使用http缓存 --- 缓存原则：越多越好，越久越好。让客户端发送更少请求，直接从本地获取，加快性能；
    - 减少cookie请求 --- 针对非必要数据（静态资源）请求，进行跨域隔离，减少传输内容大小；
    - 预加载请求 --- 针对一些业务中场景可预加载的内容，提前加载，在之后的用户操作中更少的请求，更快的响应；
    - 选择get和post --- 在http定义的时候，get本质上就是获取数据，post是发送数据的。get可以在一个TCP报文完成请求，但是post先发header，再发送数据。so，考虑好请求选型；
    - 缓存方案选型 --- 递进式缓存更新（防止一次性丢失大量缓存，导致负载骤多）；
5. 服务器响应请求 --- tomcat、IIS等服务器通过本地映射文件关系找到地址或者通过数据库查找到数据，处理完成返回给浏览器
    - 后端框架选型 --- 更快的响应，前端更快的操作；
    - 数据库选型和优化 --- 更快的响应，前端更快的操作；
6. 浏览器接受响应 --- 浏览器根据报文头里面的数据进行不同的响应处理
    - 解耦第三方依赖 --- 越多的第三方的不确定因素，会导致web的不稳定性和不确定性；
    - 避免404资源 --- 请求资源不到浪费了从请求到接受的所有资源；
7. 浏览器渲染顺序
    a. HTML解析开始构建dom树；
    b. 外部脚本和样式表加载完毕
        - 尽快加载css，首先将CSSOM对象渲染出来，然后进行页面渲染，否则导致页面闪屏，用户体验差
        - css选择器是从右往左解析的，so类似 `#test a {color: #444}` ,css解析器会查找所有a标签的祖先节点，所以效率不是那么高；
        - 在css的媒介查询中，最好不要直接和任何css规则直接相关。最好写到link标签中，告诉浏览器，只有在这个媒介下，加载指定这个css；
    c. 脚本在文档内解析并执行

- 按需加载脚本，例如现在的webpack就可以打包和按需加载js脚本；
- 将脚本标记为异步，不阻塞页面渲染，获得最佳启动，保证无关主要的脚本不会阻塞页；
- 慎重选型框架和类库，避免只是用类库和框架的一个功能或者函数，而引用整个文件；

d. HTML DOM完全构造起来

- DOM 的多个读操作（或多个写操作），应该放在一起。原则：统一读、统一写；

e. 图片和外部内容加载

- 对多媒体内容进行适当优化，包括恰当使用文件格式，文件处理、渐进式渲染等；
- 避免空的src，空的src仍然会发送请求到服务器；
- 避免在html内容中缩放图片，如果你需要使用小图，则直接使用小图；

f. 网页完成加载

- 服务端渲染，特别针对首屏加载很重要的网站，可以考虑这个方案。后端渲染结束，前端接管展示；

3.2. 前端工程化有哪些实践?

1. 技术选型：主要指基于什么原因，选择哪种前端框架：React、Vue、Angular（对微应用的理解）；
2. 规范统一：
    a. git hooks、git commit配置；
    b. eslint配置；
    c. 项目结构规范：CLI；
    d. UI规范：组件库的选择、开发与使用；
3. 测试：Jest的使用，与其他框架的对比；
4. 构建工具：webpack、rollup、vite的选择；
5. 部署：
    a. 使用Jenkins构建前端项目并部署到服务器；
    b. 如何使用github action 或者gitLab action关联项目；
6. 性能监控：
    a. 前端监控的理解与实践，performance的使用；
    b. 数据上报的方式；
    c. 如何上传错误的sourcemap；
    d. 无埋点；
7. 性能优化:
    a. 加载时性能优化：lighthouse、HTTP、CDN缓存、SSR；
    b. 运行时性能优化：重绘重排、长列表渲染；
8. 重构：为什么要重构，如何重构，重构的思想；
9. 微前端：针对巨石项目如何支持；
10. serverless：什么时候使用serverless；

3.3. 前端前沿知识有哪些了解?

1. 微前端；
2. low code、no code；
3. 前端工程化搭建；
4. 自动化部署；
5. 前端性能检测；
6. 跨端；
7. 前端组件化；
8. 前端稳定性建设；
9. 在线文档；

3.4. 面试问题

接下来我们通过问题看优质岗位的前端需要做什么：

1. 基础：
    a. 从用户输入网址到客户端展现，中间发生了什么过程?
2. 框架：
    a. React、Vue的区别，最新的版本对比；Vue2，React的源码；
    b. Vue3中composition API的使用与原理；
    c. Vue2和Vue3中diff的区别；
3. 项目问题：
    a. 组件库有自己业务定制化么？如何实现组件库的主题切换?
    b. 项目的监控告警如何配置?
    c. 项目的性能优化如何处理?
    d. node服务鉴权有哪些了解?
4. 常见问题：
    a. 做过技术复杂度最高的项目是什么；
    b. 前端工程化的实践；
    c. 前端前沿知识有哪些了解；

4. 课间休息&课程介绍

2022-爪哇前端大厂工程师训练营.pdf（4.1 MB）