2023年react高频面试题讲解

1. 国内前端岗位形势详解

1.1. 2022年前端主流趋势

2022年的前端还是有很多新兴的技术出现的，比如Vite和Turbo的性能之争、恶劣的Faker.js事件、Justjavac在Deno & Rust之外又参与了Astro源码贡献、Antfu开源的Vitest非常棒，通过种种的观察，前端专业成熟度基本上见顶了，基建基本稳定，2022年做的事都是深耕基建和前端垂类细化，整体看是围绕研发体验优化做的事儿为主，各垂类全面专业化。下面围绕性能、运行时、体积、Rust等四个方面进行讲解。

从2005年之后，相继出现Prototype、jQuery等为了抹平浏览器差异的库，极大的降低了全栈开发门开。随着电商高速发展，促成了前后端分离，并最终催生了前端工程师这一专业工种。

在2009年，Node.js横空出世，由于Node.js使用V8这个最高效的JavaScript引擎，所以在语法上，对前端更友好，或间接或直接的促进了前端构建领域的日渐成熟，比如对html不满足，就有了ejs等模版，比如css不满足，就有了sass，less等CSS预处理器，比如js不满足，就开始有了Coffee等新的衍生语言。除此之外，在模块规范层面也是有了很多演进，AMD到CommonJS到UMD，完成了很多优秀的探索，比如require.js，sea.js这样的优秀实践。另外JavaScript语言本身也取得非常大的发展，从ES5一路升级，在TC39组织下每年一个版本，为了兼容各个版本特性，诞生了Babel这样的怪物，同时，从Callback hell到Promise到Generator到Async function，虽然很乱，到总归是回到了正路上。除了这些基础外，在UI框架层面也做了大量时间，从早年各种 `hack` 面向对象基础，到extjs类的面向对象的开发方式，甚至是从Flex等富客户端，但都未能一直称雄。

随后2013年之后，开始出现Backbone，Angular类似插件，将MVVM、指令等引入到前端，继而促使React、Vue在DSL，VDom层面展开激烈角逐，并促成了React、Vue、Angular三家争霸的局面。同时，也使得分散的技术栈开始变得不满足构建需求，于是从Grunt、Gulp等传统构建器，演进为以Webpack代表的打包器。

这还只是前端爆发的开始，2015年之后，Node.js开始稳定在1.0版本，并持续发力，使得BFF（backend for frontend）遍地开花。同时围绕React、Vue、Angular三大框架的移动端开发也如火如荼，比如react-native、weex等都变得流行起来，几乎是强运营场景下必备的神器。

从2013到2019年间，是大前端的爆发式增长期。从前端基建，跨端，BFF（backend for frontend）等领域都取得了极大的进步，这也促使前端人才需求极大，尤其是对专业型人才需求极大，薪资也是可以用“狂飙”来形容。可以说这6年是前端发展最好的时期，移动端更是在2015年下半年之后开始式微，在这种情况下，只有前端几乎是唯一大的增长点。

甚至那几年有前端同学调侃说：“学不动了”，就是因为三大框架生态和大前端的快速发展导致的。但，2020年之后，基本上就没有人这样说了，前端的新轮子也不再那么高频率出现了，就连几个核心框架，更新的内容也都开始挤牙膏了，甚至是互相借鉴。这就意味着前端成熟度已经趋于平稳，剩下的事儿就是如何趋于成熟。我们看一下这2年前端所发生的变化也确认如此，你能看到的变化，已经不是轮子上的广度上追求变化，而是更多的聚焦核心场景围绕深度上进行探索，这就在稳定性和开发者体验上做更多努力，这是极好的变化。

1.1.1. 性能

说到性能，最大的问题有这几个：

1、启动时间慢，大家都受够了Webpack阵营：于是就有了swc、esbuild这些编译器，以及Turbo、Vite这样的基于新编译器的构建方式；

2. 配置太多：约定大于配置或0配置的开箱即用；
3. 包太大，动不动1个G的NPM包大小：Yarn，pnpm；
4. 模块拆分：Lerna | Yarn/ pnpm workspace；

其实都是在解决大规模编程问题，如果再直接点说，那就是解决的是开发者自己的体验问题。整体看前端生态真的做的已经很好了。

最开始的时候，很多前端使用Grunt，基于模版和插件机制的构建工具，比如jQuery等都是基于Grunt构建的。但Grunt有2个问题，就是性能不太好，另外配置复杂。于是Gulp横空出世，很快就替代了Grunt，它是基于Node.js stream机制做，只要机器有资源就可以完全利用上，在效率上比基于文件模版的方式高出一大截。即使在今天看，Gulp也是极为优秀的构建方案。

但前端发展非常快，各种CSS预处理器，模版，语言，还有Babel等各种兼容，于是就有了Webpack这样的Bundler，通过插件和loader机制，把前端这点乱七八糟的快速发展产物整理的顺畅很多，在很长很长一段时间，Webpack都随着三大框架一起成长，结果臃肿，笨重，一个稍大一点的项目几秒到几十秒都是常事。

到2019年之后，开始出现Snowpack，主要是针对ESM进行Bundless打包方案，随后Vite也借鉴了Snowpack的思路，提升开发者体验，并以Rollup和esbuild生态辅助，以高明的设计，最小的投入，获得社区快速的认同，有点像当年Gulp替代Grunt一样。按照正常剧本的话，Vite会和Gulp一样大火。偏偏半路杀出一个程咬金，Vercel开源了Turbo，以Rust做基建，拿Rust优势为底，重写底层工具链，做了一个硬核的事儿。

1.1.2. 运行时

2013到2019年，TC39组织发展的非常快。曾经阿里，360都以能加入TC39为荣，能够从语言这样的底层规范上参与，能够很大程度提升中国前端的影响力。比如2021年阿里巴巴提案Error Cause 进入Stage3，也是当年很火的新闻。但实际上经过这么多年的发展，能提的都提了，能借鉴的也都借鉴了，就好比现在Node.js源码想去提个PR是不容易的一样。

其实，Node.js从2009诞生之后，除了当年iojs分家又合并的事外，就是ry另立门户做的Deno，其实它也是一个JavaScript运行时，无非是对浏览器和最新标准支持的更好，加上对TypeScript直接支持，开箱即用是非常好的设计。

1.1.3. 体积

通常产品开发都是先实现功能，然后再优化，然后再做周边。其实这种思路在大前端生态里也是这样的。举个例子，SSR就是非常典型的例子。以淘宝的活动页为例，早年一直是没有SSR支持的，后来拉新需求以及技术突破，自然而然就会做SSR了。当你想提升性能的时候，除了万金油缓存外，减小体积就是最简单的方式，SSR就是最典型的实现，当下SSR、SSG等已经成了前端标配。

1.1.4. Rust正在变成前端新基建

说 Rust 是 WebAssembly 未来，目前看还不是，多语言里大家几率不会差太多。云原生是 Go 的地盘，难下手，机器学习 Python 才是王者，大数据和后端 Java 天下，可玩的有限。综上，JavaScript 依然是应用软件最好的选择。但Rust做基建，提供更好的开发体验，倒是大有可为的。

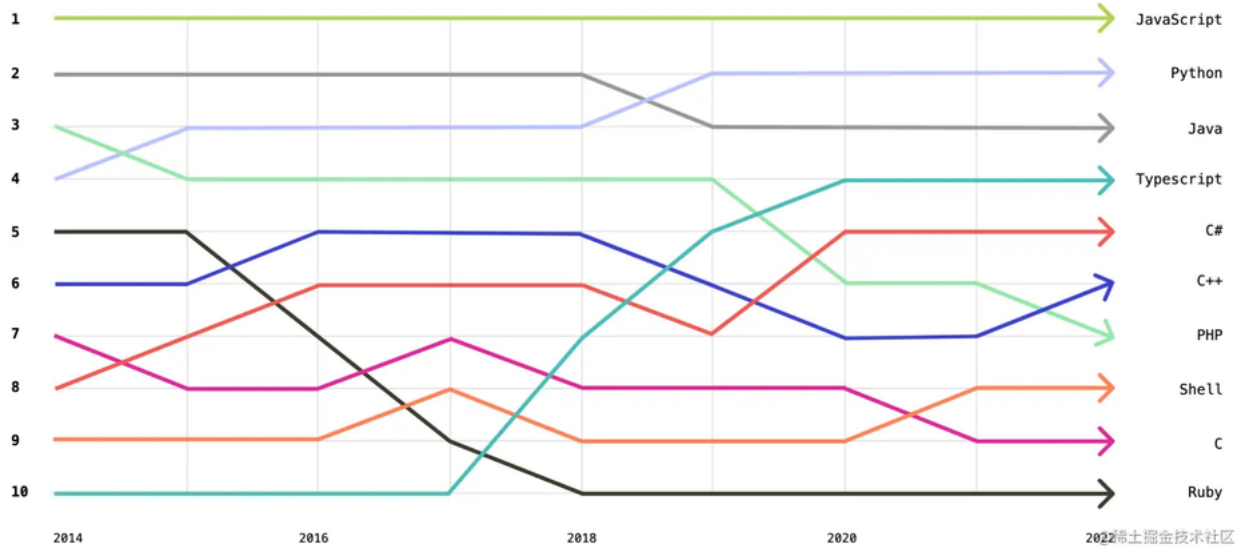
Rust 语言在前端工具链的影响越来越大，目前可以看到 Next.js对 Rust 重仓，招揽大量人才，swc 作者，Rollup 作者等等，未来可能是一个很好的解决前端体验的方向。

1.2. 2023年就业情况分析

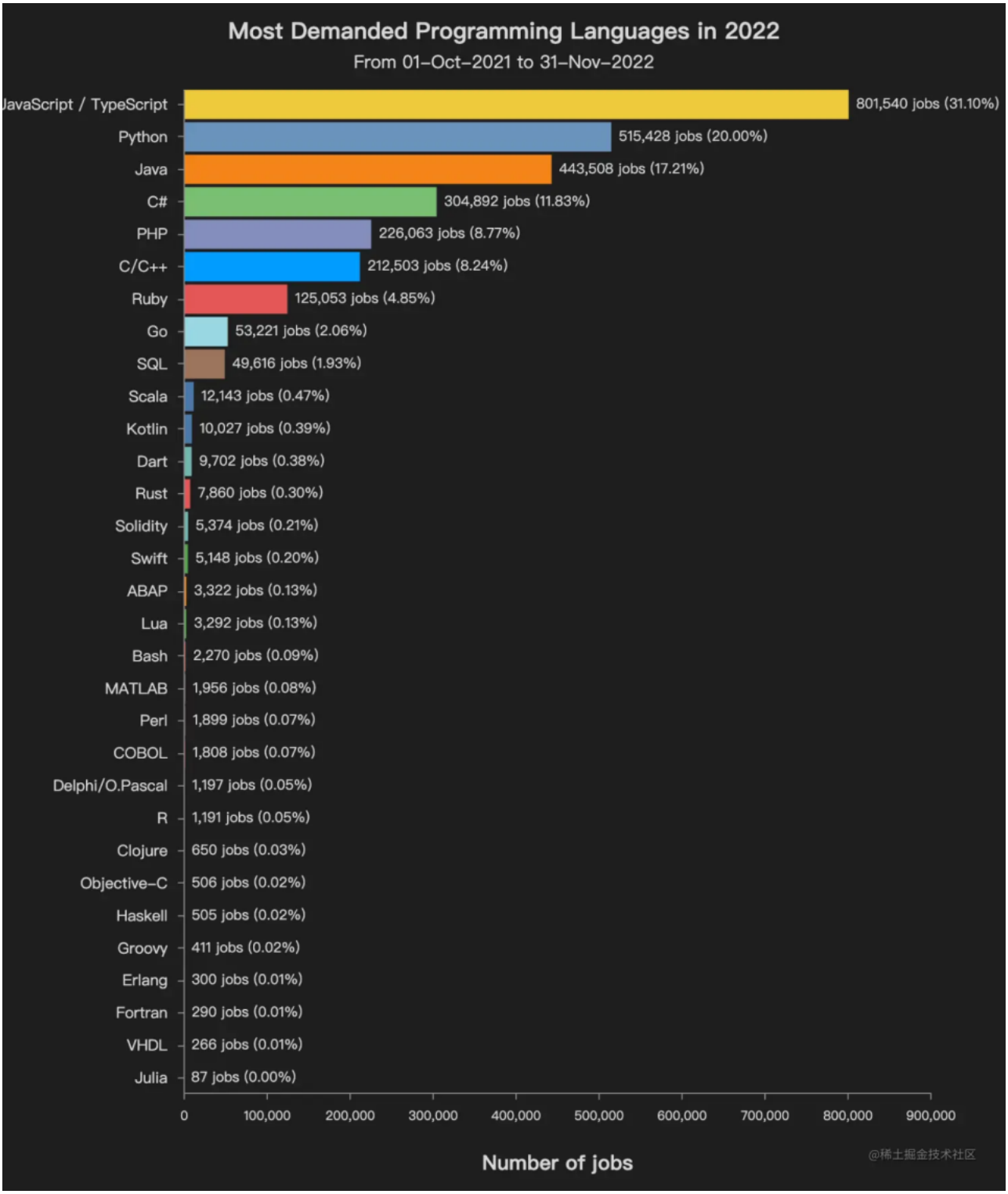
整体上，前端的状态是趋于成熟，已经过了第一波爆发式增长期，但它还在垂类细化领域不断发展中，所以我还是非常看好的。

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