前22年的Loser，后4年和自己赛跑的人 | 最惨前端面经

跳槽原因

前东家部门是做旅游的，在这次疫情打击下，基本玩完。

于是我半休半远程三个月后，在4月底领了裁员便当。至今，差不多找了两个月的工作。

本篇不是标准的面经，想从中获取大厂跳槽经验的可以歇一歇。

1. `Offer` 情况

个人比较懒，一周可能就面2～3家，只约下午。部分星期没有面试邀约。

囿于学历+公司，两招聘软件都被我用成“ `Boss` 直拒”和“拉钩上吊”

粗略算了下，面了约12家大中小型公司，仅4家 `Offer` ，情况分别为：

- 某游戏公司
- 某小公司
- 风变编程
- 金山系某司

作为一个社交孤儿，在本次跳槽历程中也是发现自己不少的问题，且听我慢慢道来。

本篇虽然有点丧，但你们可以从中找到对应的问题（我几乎犯了所有面试的低级错误）

部分的公司有： `360` 奇舞团，某上市游戏公司，风变编程，金山系某司，阿里。

2. 高频面试题汇总

面过的公司有点多，一并说了吧。

1. 从“在浏览器输入域名”到“页面静态资源完全加载”的整个流程

> **见于：某游戏公司、小鹅通、阿里一面、另外三家小公司**

这问题的答案，我结合了 yck 《前端面试之道》和 浏览器原理专栏：

整个过程可以分为几步：

1. 用户输入

   当用户输入关键字并键入回车之后，这意味着当前页面即将要被替换成新的页面，不过在这个流程继续之前，浏览器还给了当前页面一次执行 `beforeunload` 事件的机会，`beforeunload` 事件允许页面在退出之前执行一些数据清理操作，还可以询问用户是否要离开当前页面。

2. `URL` 请求过程

   首先，网络进程会查找本地缓存是否缓存了该资源。

   如果有缓存资源，那么直接返回资源给浏览器进程；如果在缓存中没有查找到资源，那么直接进入网络请求流程。这请求前的第一步是要进行 `DNS` 解析，以获取请求域名的服务器 `IP` 地址。如果请求协议是 `HTTPS`，那么还需要建立 `TLS` 连接。

   - 其中， `DNS` 也有几步缓存：浏览器缓存， `hosts` 文件，
   - 如果本地域名解析服务器也没有该域名的记录，则开始递归+迭代解析
   - `TCP` 三次握手， `HTTP` 。 `TLS` 握手， `HTTPS` 。

   接下来就是利用 `IP` 地址和服务器建立 `TCP` 连接。连接建立之后，浏览器端会构建请求行、请求头等信息，并把和该域名相关的 `Cookie` 等数据附加到请求头中，然后向服务器发送构建的请求信息。

   数据在进入服务端之前，可能还会先经过负责负载均衡的服务器，它的作用就是将请求合理的分发到多台服务器上，这时假设服务端会响应一个 `HTML` 文件。

   首先浏览器会判断状态码是什么，如果是 `200` 那就继续解析，如果 `400` 或 `500` 的话就会报错，如果 `300` 的话会进行重定向，这里会有个重定向计数器，避免过多次的重定向，超过次数也会报错。

   浏览器开始解析文件，如果是 `gzip` 格式的话会先解压一下，然后通过文件的编码格式知道该如何去解码文件。

3. 准备渲染进程

   默认情况下， Chrome 会为每个页面分配一个渲染进程，也就是说，每打开一个新页面就会配套创建一个新的渲染进程。

4. 渲染阶段

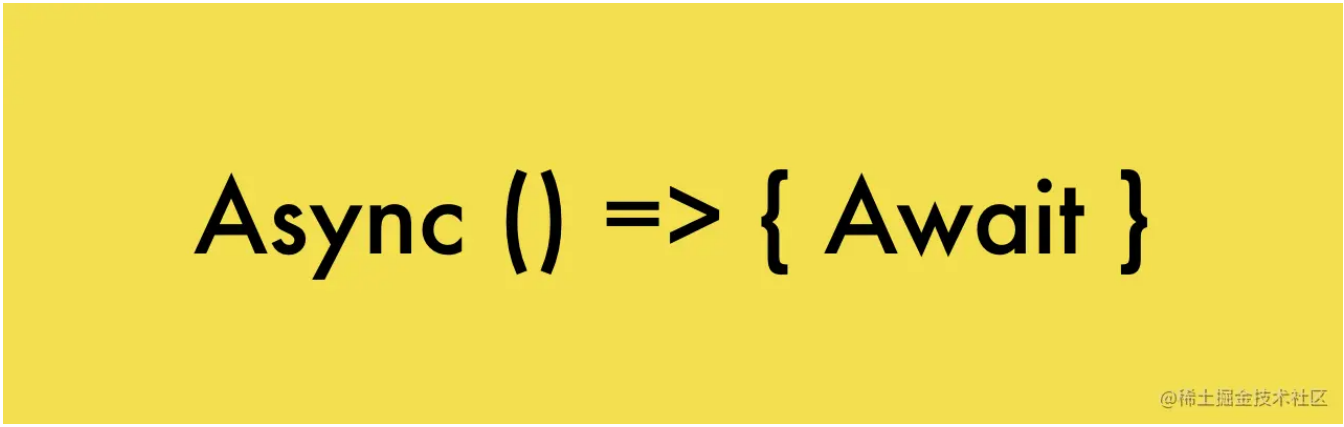
   文件解码成功后会正式开始渲染流程，先会根据 HTML 构建 DOM 树，有 CSS 的话会去构建 CSSOM 树。如果遇到 script 标签的话，会判断是否存在 async 或者 defer ，前者会并行进行下载并执行 JS，后者会先下载文件，然后等待 HTML 解析完成后顺序执行。

   如果以上都没有，就会阻塞住渲染流程直到 JS 执行完毕。

CSSOM 树和 DOM 树构建完成后会开始生成 Render 树，这一步就是确定页面元素的布局、样式等等诸多方面的东西

在生成 Render 树的过程中，浏览器就开始调用 GPU 绘制，合成图层，将内容显示在屏幕上了。

2. eventloop 机制， promise 的实现和静态方法、 async 实现。



这题聊起来可就大了，进程，线程，协程。部分还会配以那道最经典的 eventloop 题目。

见于：阿里一面、小鹅通、头条一面、360一面、风变编程、以及其它四家公司，必考。

Event Loop 是什么?

JavaScript 的事件分两种，宏任务(macro-task)和微任务(micro-task)

- **宏任务**：包括整体代码 script，setTimeout，setInterval
- **微任务**： Promise.then(非new Promise) ， process.nextTick(node中)

- 事件的执行顺序，是先执行宏任务，然后执行微任务，这个是基础，任务可以有同步任务和异步任务，同步的进入主线程，异步的进入 `Event Table` 并注册函数，异步事件完成后，会将回调函数放入 `Event Queue` 中(宏任务和微任务是不同的 `Event Queue`)，同步任务执行完成后，会从 `Event Queue` 中读取事件放入主线程执行，回调函数中可能还会包含不同的任务，因此会循环执行上述操作。

`Promise` 的含义

`Promise` 是一个异步编程的解决方案，简单来讲， `Promise` 类似一个盒子，里面保存着在未来某个时间点才会结束的事件。

三种状态：

- `pending` ： 进行中
- `fulfilled` ： 已经成功
- `rejected` ： 已经失败 状态改变，只能从 `pending` 变成 `fulfilled` 或者 `rejected` ，状态不可逆。

`async` 实现和常用方法

`async` 函数的实现原理，就是将 `Generator` 函数和自动执行器，包装在一个函数里。

`Generator` 函数是协程在 ES6 的实现，最大特点就是可以交出函数的执行权（即暂停执行）。

协程是一种用户态的轻量级线程， 协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

3. `Vue` 和 `React` 之间的区别

> 摘自yck《前端面试之道》

1. Vue 的表单可以使用 `v-model` 支持双向绑定，相比于 React 来说开发上更加方便，当然了 `v-model` 其实就是个语法糖，本质上和 React 写表单的方式没什么区别。

2. 改变数据方式不同，Vue 修改状态相比来说要简单许多，React 需要使用 `setState` 来改变状态，并且使用这个 API 也有一些坑点。并且 Vue 的底层使用了依赖追踪，页面更新渲

染已经是最优的了，但是 React 还是需要用户手动去优化这方面的问题。

3. React 16以后，有些钩子函数会执行多次，这是因为引入 Fiber 的原因，这在后续的章节中会讲到。

4. React 需要使用 JSX，有一定的上手成本，并且需要一整套的工具链支持，但是完全可以通过 JS 来控制页面，更加的灵活。Vue 使用了模板语法，相比于 JSX 来说没有那么灵活，但是完全可以脱离工具链，通过直接编写 `render` 函数就能在浏览器中运行。

5. 在生态上来说，两者其实没多大的差距，当然 React 的用户是远远高于 Vue 的。

6. 在上手成本上来说，Vue 一开始的定位就是尽可能的降低前端开发的门槛，然而 React 更多的是去改变用户去接受它的概念和思想，相较于 Vue 来说上手成本略高。

4. `Vue 3.0` 面试题

> **见于：360一面、风变编程，预计下半年必考。**

> 1. `Vue3.0` 都有哪些重要新特性?

- 建议往Composition API和Tree-shaking方面答，对应比较 `React Hooks` 和 `webpack` 的 `Tree-shaking`

> 2. `Vue3.0` 对比 `Vue2.0` 的优势在哪?

> 3. `Vue3.0` 和 `React 16.X` 都有哪些区别和相似处?

- 可以更新下你的横比答案了，重点突出两者开始相互借鉴，互有优点。记得夸夸 `Vue3.0` 抄过来，却做得更好的部分。

> 4. `Vue3.0` 是如何实现代码逻辑复用的?

- 可以先对比 `Composition API` 和 `mixin` 的差异，并凸显出 `Vue2.0` 那种代码上下反复横跳的缺点。

以上答案基本可以在下面两篇博客里找到：

> 《抄笔记：尤雨溪在Vue3.0 Beta直播里聊到了这些...》

> 《Vue3 究竟好在哪里？（和 React Hook 的详细对比）》

5. React 高频面试题

> 1. React 16.X 的 Fiber 原理

> 2. setState 原理，什么时候是同步的？

> 3. React Hooks 相对高阶组件和 Class 组件有什么优势/缺点？

> 4. React 16.X 的生命周期，以及为何要替换掉以前的？

> 5. React 跨平台的实现原理。

> 6. 说一说 redux ，以及比 flux 先进的原因。

平心而论，如果面试前未要求技术栈，建议往 Vue 方向引。 React 的面试题要高一两个难度....

6. HTTP 高频面试题

见于：阿里一面、头条一面、360一面、风变编程、以及其它四家公司，必考。

> 1. 讲一讲强缓存和协议缓存？

2. HTTP/2.0 都有哪些特性？头部压缩的原理？

3. TCP 三次握手和四次挥手？以其存在意义。

4. 状态码。 302.304.301.401.403 的区别？

5. 状态码。 204 和 304 分别有什么作用？

6. HTTP 和 HTTPS 握手差异？

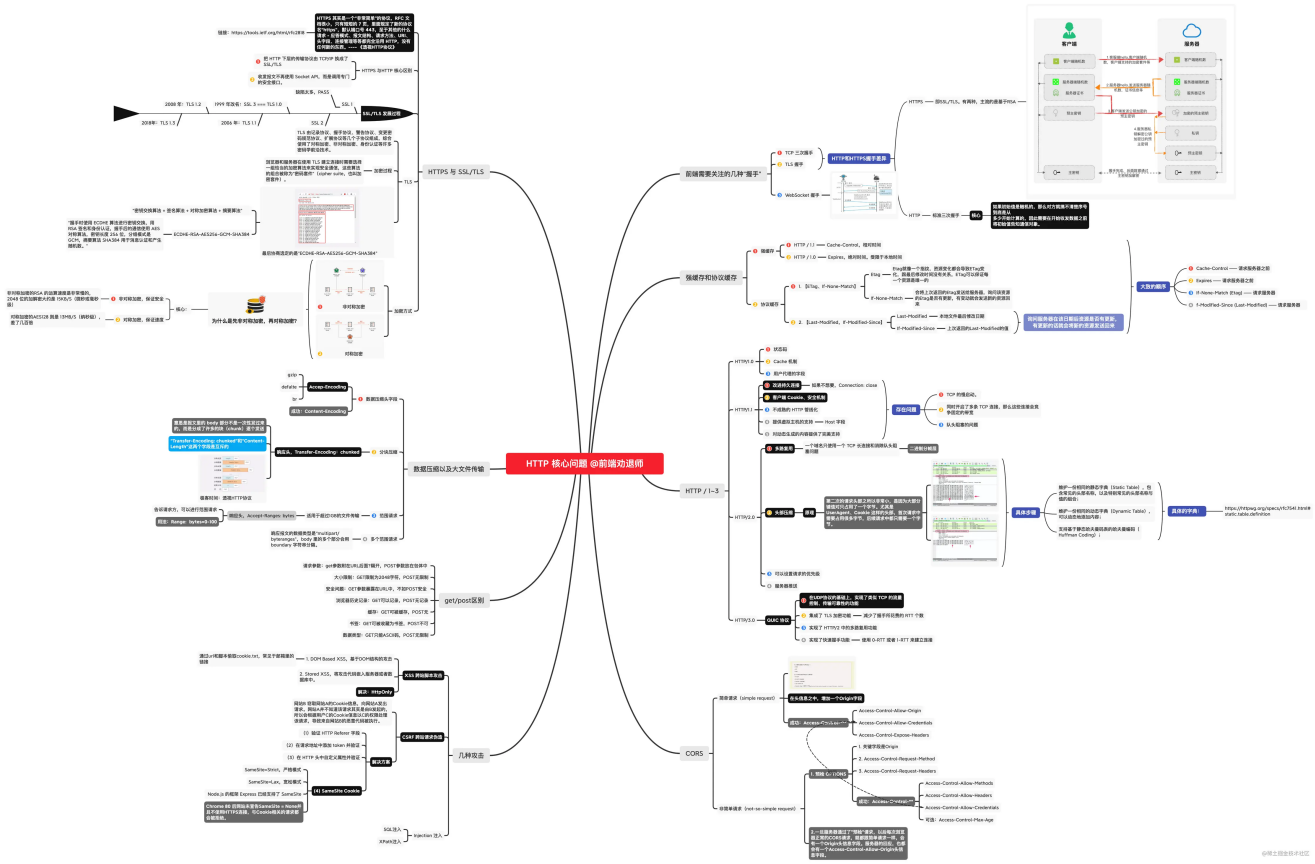
7. CSRF 跨站请求伪造和 XSS 跨站脚本攻击是什么？

8. 你是如何解决跨域的？都有几种？

9. nginx 了解吗？你都用来做什么？

10. 有了【 Last-Modified，If-Modified-Since 】为何还要有【 ETag、If-None-Match 】

我总结过一张 xmind 图，欢迎到我公众号里自取。



7. JS/CSS 高频基础问题

见于：阿里一面、头条一面、风变编程、以及其它多家公司，非常高频。

> 1. 弹性盒子中 `flex: 0 1 auto` 表示什么意思？
>
> 2. 箭头函数可以用 `new` 实例化吗？聊聊 `this` 的指向问题。
>
> 3. 聊一聊原型链。
>
> 4. 垃圾回收中的堆和栈的区别。
>
> 5. `0.1 + 0.2 != 0.3` 背后的原理？

> 6. TypeScript 用过吗？聊聊你对 TypeScript 的理解？

> 7. 图片懒加载的原理。

> 8. call、apply 和 bind 方法的用法以及区别

> 9. Webpack 原理，以及常用插件

8. 项目及优化相关

> 1. 项目中遇到的难点，以及解决思路。

> 2. 你是如何做 Web 性能优化的？首屏渲染如何处理？

这个问题很大，我有个简略版，回答思路引自专栏《 浏览器工作原理与实践》：

主要围绕着两个阶段：加载阶段 和 交互阶段

加载阶段：

1. 减少关键资源的个数和大小（ Webpack 拆/合包，懒加载等）
2. 减少关键资源 RTT 的时间（ Gzip 压缩，边缘节点 CDN ）

交互阶段：

1. JS 代码不可占用主线程太久，与首屏无关的脚本加上延后处理（ aysnc/defer ）属性，与 DOM 无关的交给 Web Worker 。
2. CSS 用对选择器（尽可能绑定 Class 或 Id ）,否则会遍历多次。
3. 首屏渲染中如果有动画，加上 will-change 属性，浏览器会开辟新的层处理（触发合成机制）
4. 避免强制同步布局，避免布局抖动。
5. 图片懒加载（有四种方式）

> 3. 埋点数据上报的方案做过吗?

> 4. 前端架构思考，你是如何考量部门的技术栈的?

> 5. 前端重构思考，老项目在新业务紧急与重构技术债务间如何衡量轻重?

9. 全链路以及 DevOps 认知

由于我第二家公司部门是做 DevOps 平台，有些与前端无关的面试题。

> 1. 单元测试做过吗？你是怎么做的?

> 2. docker 准备流程?

> 3. DevOps 平台关键功能点?

> 4. 自动化测试， CI/CD 的关键核心都有哪些?

> 5. 如何保障 DevOps 推动?

> 6. 接口如何做优化? Mock 平台搭建方案?

3. 面试感受

今年，太难太难了。 @木易杨 大佬的这段话很对:

> 先来说下大环境，感觉非常不好，就一二线互联网来说招人的没几家公司，裁员的、内部调整的、锁 HC 的确是一大堆，所以大家在换工作的时候一定不要裸辞，风险太大。

> 今年面试和往年感受有些不同，对于项目的重难点、亮点、个人在团队中做的贡献、对项目的 Owner 意识等比较关注，还有就是编程能力的考察会更多一些。

简单讲，就是学历是第一竞争力，你靠 APP 投简历几乎没反馈，**内推最靠谱。**

其次，如果你仅有大专，能力非高得不可替代（或有大佬愿做信任背书），不建议在今年频繁投大厂（白白浪费宝贵的机会），我就经历过各种各样简历面挂和HR面挂，部分原因：

1. 跳槽频繁（我四年三家，其中两家是部门解散/倒闭）
2. 博客写太多，专注力不够，英语不够好。
3. 过往项目没亮点，直接否定。

平心而论，过往跳槽都不是很顺利，但今年是真感受到了天花板：

1. 大公司投简历投不进。
2. 小公司薪资满足不了。

充分体会到一句话：

你往后的日子里，都在为高中不努力买单。

4. 面试技巧

严格说，本面渣不配给一些面试技巧，但有些是我没做好却很不错的东西。建议你们看一看：

1. 付费找人包装简历

如果你一没学历，二没大厂简历，简历写得稀烂。

相信我，投递之前，**付个几十块找业内大佬帮你改改简历，你一定会有更多的投递反馈。**

2. 优秀的自我介绍模版

面试官您好!

1. 我叫 *** ，很开心今天来应聘 ** 岗位，我有 * 年 *** 岗位工作经验，工作内容包括 **、*** ** 等，曾参与 *** 项目/工作，完成 ** 业绩，这些经验锻炼了我 *** 能力。除了日常业务开发外，我还在 *** 方面.....

2. 面试之前，我了解到咱们公司主要从事 *** 业务、 *** 类产 品，属于行业排名 *** 的企业，我对这个行业非常看好，也想在这行 长期发展！这个岗位要求的 ** *** 能力和经验，与我的工作经历很 匹配，相信我能够胜任这个岗位，谢谢！

3. 项目问题提前备好草稿

在我面试一个月左右，发现经常挂在二/三面后，开始审视自己的问题：

1. 回答项目难点，解决方案时太草率，只描述做了什么，并没体现方案给项目带来的进步。
2. 缺乏断点，一个劲儿说不停，缺乏对语句的掌控。

说实话，挺难述说的，特别是我这种没啥大项目经验，前三年都在做中后台系统居多，只能从一些细节入手，以下是我的草稿（虽然很烂），供你们参考一下。：

Vue后台细颗粒权限控制与防多人操作：

1. 一般简单的权限控制会以角色区别，但开发/运维们想自己设成员，**需求1**
2. 给他们做了一个权限管理的模块，但因为DevOps的功能模块太多，记不住，开发/运维们想要直观的修改权限，**需求2。**
3. 考虑到这个痛点，我设计了一个“授权模式”，高权限的拥有切换功能。

   切换后，该模块下所有的按钮都会先拦截左键点击，并拦截默认右键，多了一个自定义属性。以及授权/冻结 菜单。授权功能，直接拥有选择成员的树控件弹窗。

   使用后会上传 成员数组 + 自定义属性。下一次访问后台便返回 该自定义属性 对象 ，以确定新的权限。

   权限问题解决后，开发/运维们又发现，一个部署/发布 点击模块存在同时操作的人很多。会冲突。**需求4。**
4. 在多人频繁触发模块，加入了 `WebSocket` 管理，实现类似在线文档的显示功能，显示当前操作人，并设屏蔽（同时操作会有显示姓名，并锁住）。
   - 简单版本：当前按钮 点击后，后台设个时间阀值，该段时间内其他人点击了就弹出提醒当前有人操作。
   - 复杂版本：多人 `WebSocket` 维护一个值。

对应场景： `DevOps` 平台

5. 和自己赛跑的人

去年写过一篇年度总结：

> 我叫“笑妄”，16年地理信息系统专业专科毕业，自学的前端。 目前三年半的经验，前后工作过几家中小公司，做过Python爬虫，也曾在运维开发部混过。前两年的工作，都在

> 为生计挣扎，做码农仅因自身一无所长，看这行工资高，就挤进来了。--- 《前端废材的自我劝退之路》

`y1s1`，我底子很差。1年时才搞懂布局，2年不会函数 `return`，3年才熟悉 `React`。

入这行以来，一路虽有伯乐，但于前端领域仍单打独斗居多。

却得益于这行，让我知“井外方觉天地大”。认识优秀的人越多，越想努力，精进。

所以这段时间也没闲着，看了一些还不错的书/专栏：

1. 《网络是怎样连接的》，难，建议笔记。
2. 《`DevOps` 实战笔记》，有趣，但需有一定的基础。
3. 《透视 `HTTP` 协议》，还不错，值二刷。
4. 《图解 `Google V8`》，相对第一本《浏览器原理》，水了不少，部分值二刷。
5. 《`Nginx` 核心知识100讲》，目前正在看。

如何提高自己的技术竞争力（或收入）？

1. **写技术博客。**“从某种意义上说，博客是我最好的学习笔记和个人名片。在IT行业内，技术博客是了解一个开发者最好的方式之一，特别是当你没有一张足够分量的文凭或者一段出彩的工作经历时，你就应该沉下心来好好打磨自己技术，打造自己的博客。往者不可谏，来者犹可追。” ---@浪里行舟 《写技术博客那点事》
2. **立足于前端，放眼全链路。**今年我原以为的弱势 `DevOps`，帮助我搞掂了不少2/3面技术面。
3. **跟对人**，这可说是收入涨幅最大的依靠（也是最可遇不可求）。我曾见过某友，从17年9K，到如今50k+的飞跃。不说了，我真的酸。。。

这一路上，我的能力，学历，背景样样不如人，也曾懒惰曾迷茫，但一直都在和自己赛跑，我不服输：

虽然在你离开学校的时候

所有的人都认为你不会有出息

你却没有因此怨天尤人自暴自弃

....

在那时侯我们身边都有一卡车的难题

不知道成功的意义就在超越自己

我们都是和自己赛跑的人

为了更好的未来拼命努力

争取一种意义非凡的胜利

为了更好的明天拼命努力