中小厂面试真是题目（三年经验）

A公司

整体总结

远程面试，面试官是一个小姐姐，但我觉得应该是一个新手面试官，面试题估计是从哪个博客看来的，只会问问题，我答的好不好，对不对没有任何反馈，也没有根据我回答的情况继续追问，然后面试的过程也没有什么章法，先问我 `Vue` 原理，到后面又问我基础。整个过程也是很灵异的，开头没让自我介绍，结尾没有问我有没有问题，面试完直接就挂了，像是在完成任务而不是招一个同事。

面试题一览

1. `Vue2` 和 `Vue3` 有什么区别
2. `Vue2` 相比 `Vue3` 有什么缺点
3. 虚拟 `dom` 是什么，为什么要使用虚拟 `dom`
4. `diff` 算法中 `key` 是做什么用的
5. 用 `index` 做 `key` 会有什么问题吗，为什么不推荐使用 `index` 做 `key`
6. 用 `index` 拼接一个随机字符串做 `key` 可以解决上述的问题吗
7. 组件之间的通讯方式有哪些
8. 说一下你对原型和原型链的理解
9. `JS` 实现继承有哪些方式
10. `webpack` 的构建过程是什么
11. 解决跨域有哪些方式
12. `get` 和 `post` 请求有什么区别
13. `localStorage，session，cookie` 的区别是什么
14. `promise` 是做什么的
15. `promise` 的实现原理是什么
16. `let,const,var` 有什么区别
17. 数组去重都有哪些方式

`Vue2` 和 `Vue3` 有什么区别

这个只要同时有 `Vue2` 和 `Vue3` 开发经验的，应该都能说上一些，建议熟读 `Vue3` 官方文档。

- 响应式原理
- 生命周期钩子名称
- 自定义指令钩子名称
- 新的内置组件
- `diff` 算法
- `Composition API`
- 对于 `TS` 的支持
- ... ...

`Vue2` 相比 `Vue3` 有什么缺点

这个问题和上一个差不多，主要是从区别上下手

- 性能问题：`Vue2` 采用的是双向数据绑定和脏检查的方式，对于大型应用或复杂组件来说，性能可能会受到影响。
- 大量代码：`Vue2` 需要引入大量的代码来支持其功能，使得包的体积较大。
- 逻辑复用：`Vue2` 对于逻辑复用的支持不是很好，`Vue3` 的 `compositionApi` 很好的解决了这个问题。
- `TypeScript` 支持不佳：`Vue2` 对于 `TypeScript` 的支持较弱，需要借助额外的插件来实现类型检查和补充 `IntelliSense` 等功能。
- 原生支持数组的响应式，无需重写数组原型方法。

虚拟 `dom` 是什么，为什么要使用虚拟 `dom`

`Virtual DOM` 是 `DOM` 节点在 `JavaScript` 中的一种抽象数据结构，之所以需要虚拟 `DOM`，是因为浏览器中操作 `DOM` 的代价比较昂贵，频繁操作 `DOM` 会产生性能问题。

虚拟 `DOM` 的作用是在每一次响应式数据发生变化引起页面重渲染时，`Vue` 对比更新前后的虚拟 `DOM`，匹配找出尽可能少的需要更新的真实 `DOM`，从而达到提升性能的目的。

虚拟 `DOM` 的实现原理主要包括以下 **3** 部分：

- 用 `JavaScript` 对象模拟真实 `DOM` 树，对真实 `DOM` 进行抽象；
- `diff` 算法 — 比较两棵虚拟 `DOM` 树的差异；
- `pach` 算法 — 将两个虚拟 `DOM` 对象的差异应用到真正的 `DOM` 树。

👉👉 什么是虚拟DOM

`diff` 算法中 `key` 是做什么用的

`key` 是为 `Vue` 中 `vnode` 的唯一标记，通过这个 `key`，我们的 `diff` 操作可以更准确、更快速。`Vue` 的 `diff` 过程可以概括为：`oldCh` 和 `newCh` 各有两个头尾的变量 `oldStartIndex`、`oldEndIndex` 和 `newStartIndex`、`newEndIndex`，它们会新节点和旧节点会进行两两对比，即一共有 **4** 种比较方式：`newStartIndex` 和 `oldStartIndex`、`newEndIndex` 和 `oldEndIndex`、`newStartIndex` 和 `oldEndIndex`、`newEndIndex` 和 `oldStartIndex`，如果以上 **4** 种比较都没匹配，如果设置了 `key`，就会用 `key` 再进行比较，在比较的过程中，遍历会往中间靠，一旦 `StartIdx > EndIdx` 表明 `oldCh` 和 `newCh` 至少有一个已经遍历完了，就会结束比较。

在 `diff` 算法中，`key` 是用来标识组件或元素的唯一性的。它的作用是在 `Virtual DOM` 对比更新过程中帮助确定哪些组件或元素需要被更新、删除或重新渲染。当 `key` 被添加到组件或元素上时，`Vue` 会使用 `key` 来追踪它们的标识。在进行 `diff` 算法的过程中，`Vue` 会比较新旧 `Virtual DOM` 树中的组件或元素的 `key`，如果某个组件或元素在新旧树中的 `key` 相同，`Vue` 会认为它是同一个组件或元素，然后根据需要进行更新、移动或删除操作。使用 `key` 可以有效提高 `diff` 算法的性能，避免重复渲染和更新无关的组件或元素。它可以帮助 `Vue` 更精确地判断哪些部分需要更新，提高组件的复用性和渲染效率。需要注意的是，`key` 应该是稳定、唯一且可预测的，最好使用具有唯一标识的属性作为 `key` 值，如 `id` 或具有唯一性的字段。避免使用 `index` 作为 `key`，并且同一级别下的兄弟元素或组件的 `key` 值应该是唯一的，以确保 `diff` 算法的准确性。

所以 `Vue` 中 `key` 的作用是：`key` 是为 `Vue` 中 `vnode` 的唯一标记，通过这个 `key`，我们的 `diff` 操作可以更准确、更快速，因为带 `key` 就不是就地复用了，在 `sameNode` 函数 `a.key === b.key` 对比中可以避免就地复用的情况。利用 `key` 的唯一性生成 `map` 对象来获取对应节点，比遍历方式更快，源码如下：

```
function createKeyToOldIdx (children, beginIdx, endIdx) {
  let i, key
  const map = {}
  for (i = beginIdx; i <= endIdx; ++i) {
    key = children[i].key
    if (isDef(key)) map[key] = i
  }
  return map
}
```

用 `index` 做 `key` 会有什么问题吗，为什么不推荐使用 `index` 做 `key`

使用 `index` 做 `key` 会有以下问题：

- 不稳定性：组件或元素的索引可能会发生变化，当列表中的组件或元素被重新排序、添加或删除时，其对应的 `index` 也会发生变化。这会导致 `key` 不稳定，可能会引发一些错误的更新或重新渲染。

- 渲染性能：当列表项中的组件或元素重新排序时，使用 `index` 作为 `key` 可能会触发不必要的重新渲染。因为 `Vue` 在进行 `diff` 算法时，会认为同一个 `index` 的组件或元素是同一个，不会重新创建和更新。这样就可能导致原本是不同的组件或元素被错误地复用，从而引发意料之外的问题。
- 面临删除与插入时的性能问题：当列表中的组件或元素发生删除、插入操作时，使用 `index` 作为 `key` 无法准确识别被删除或插入的内容，可能会导致不必要的重新渲染。这是因为 `Vue` 会认为被删除的组件或元素与新添加的组件或元素具有相同的 `index`，从而复用之前的组件或元素，而不是根据具体的变化进行创建或删除。

使用 `index` 作为 `key` 和没写基本上没区别，因为不管数组的顺序怎么颠倒，`index` 都是 `0, 1, 2...` 这样排列，导致 `Vue` 会复用错误的旧子节点，做很多额外的工作。

👉👉 为什么 Vue 中不要用 index 作为 key?

用 `index` 拼接一个随机字符串做 `key` 可以解决上述的问题吗

不可以，同样会产生以下问题。

- 不稳定性：随机字符串虽然可以确保 `key` 的唯一性，但在重新渲染组件或元素时，生成的随机字符串会发生变化。当组件或元素重新排序、添加或删除时，`index` 值会被重新分配，从而导致随机生成的字符串也会发生变化，使得 `key` 不是稳定的。
- 性能问题：由于随机生成的字符串是随机的，`diff` 算法无法利用它们的稳定性进行优化。每次更新时，`Vue` 都会认为组件或元素是不同的，导致不必要的重新渲染和更新。

组件之间的通讯方式有哪些

（1）`props / $emit` 适用 父子组件通信

（2）`ref` 适用 父子组件通信

- `ref`：如果在普通的 `DOM` 元素上使用，引用指向的就是 `DOM` 元素；如果用在子组件上，引用就指向组件实例

（3）`$parent` / `$children` / `$root`：访问父 / 子实例 / 根实例

（4）`EventBus （$emit / $on）` 适用于 父子、隔代、兄弟组件通信

这种方法通过一个空的 `Vue` 实例作为中央事件总线（事件中心），用它来触发事件和监听事件，从而实现任何组件间的通信，包括父子、隔代、兄弟组件。

（5）`$attrs` / `$listeners` 适用于 隔代组件通信

- `$attrs`：包含了父作用域中不被 `prop` 所识别 (且获取) 的特性绑定 (`class` 和 `style` 除外)。当一个组件没有声明任何 `prop` 时，这里会包含所有父作用域的绑定 (`class` 和 `style` 除外)，并且可以通过 `v-bind="$attrs"` 传入内部组件。通常配合 `inheritAttrs` 选项一起使用。

- `$listeners`：包含了父作用域中的 (不含 `.native` 修饰器的) `v-on` 事件监听器。它可以通过 `v-on="$listeners"` 传入内部组件

（6）`provide / inject` 适用于 隔代组件通信

祖先组件中通过 `provide` 来提供变量，然后在子孙组件中通过 `inject` 来注入变量。`provide / inject API` 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

（7）`Vuex` 适用于 父子、隔代、兄弟组件通信

`Vuex` 是一个专为 `Vue.js` 应用程序开发的状态管理模式。每一个 `Vuex` 应用的核心就是 `store`（仓库）。`store` 基本上就是一个容器，它包含着你的应用中大部分的状态 (`state`)。

- `Vuex` 的状态存储是响应式的。当 `Vue` 组件从 `store` 中读取状态的时候，若 `store` 中的状态发生变化，那么相应的组件也会相应地得到高效更新。
- 改变 `store` 中的状态的唯一途径就是显式地提交 `(commit) mutation`。这样使得我们可以方便地跟踪每一个状态的变化。

（8）插槽

`Vue3` 可以通过 `usesolt` 获取插槽数据。

（9）`mitt.js` 适用于任意组件通信

`Vue3` 中移除了 `$on`， `$off` 等方法，所以 `EventBus` 不再使用，相应的替换方案就是 `mitt.js`

说一下你对原型和原型链的理解

👉👉 深入JavaScript系列（六）：原型与原型链

`JS` 实现继承有哪些方式

这个知识点平时真不关注，也不用，就答上了原型链继承和 `ES6 Class`

👉👉 js继承的七种方式

`webpack` 的构建过程是什么

`Webpack` 是一个模块打包工具，它将应用程序的代码及其依赖项打包到一个或多个静态资源 (`bundle`)中，以便在浏览器中加载。

`Webpack` 的构建过程主要包括以下几个步骤：

- **Entry（入口）**：指定一个或多个入口文件作为构建的起点， `Webpack` 会从这些入口文件开始递归地解析和构建依赖关系。
- **Module（模块）**：解析入口文件及其依赖的模块。 `Webpack` 会根据配置中的不同模块规则（ `rules` ）来处理不同类型的模块，如 `JS` 文件、 `CSS` 文件、图片等。

- **Chunk（代码块）**：根据模块之间的依赖关系，`Webpack` 将模块分组成不同的块（`Chunk`）。一个块可以是一个文件或多个文件组成的一个逻辑单元。
- **Loaders（加载器）**：`Webpack` 使用加载器来处理非 `JavaScript` 文件。加载器允许在打包过程中对模块进行预处理。例如，`Babel loader` 可以将 `ES6/ES7` 代码转换为浏览器可以理解的 `ES5` 代码。
- **Plugins（插件）**：插件用于执行更广泛的任务和自定义构建过程。它们可以用于优化输出、资源管理、注入环境变量等。
- **Output（输出）**：指定打包后的文件输出的位置和命名规则。`Webpack` 会将打包后的文件输出为一个或多个静态资源(`bundle`)。
- **DevServer（开发服务器）**：`Webpack` 还提供了一个开发服务器(`Webpack Dev Server`)，可以在开发过程中实时重新加载文件，使开发更加高效。

在以上步骤完成后，就可以生成一个或多个静态资源(bundle)，准备用于在浏览器中运行应用程序。

解决跨域有哪些方式

👉👉 九种跨域方式实现原理（完整版）

`get` 和 `post` 请求有什么区别

`GET` 和 `POST` 是两种常见的 `HTTP` 请求方法，它们有以下区别：

语义不同：

- `GET` 请求用于获取服务器上的某个资源，并将其返回给客户端。它是一种幂等的请求，不应该对服务器产生任何副作用。比如，获取网页内容、获取数据、进行搜索等。
- `POST` 请求用于向服务器提交数据，请求服务器执行特定的动作，或者在服务器上创建资源。它可能会对服务器产生副作用，比如添加、更新或删除数据等。

数据位置：

- `GET` 请求的数据通过 `URL` 的查询参数传递，会将数据附加在 `URL` 的末尾，可见于请求的 `URL` 中。
- `POST` 请求的数据通过请求体（`request body`）中进行传递，不会直接暴露在 `URL` 中，对数据的长度和格式没有限制。

数据传输安全性：

- `GET` 请求的数据以明文形式在 `URL` 中传输，可能会被缓存、保存在浏览器历史记录或服务器访问日志中，安全性较低。
- `POST` 请求的数据通过请求体传输，相对于 `GET` 请求更加安全，能够降低数据泄漏的风险。

使用场景：

- `GET` 请求适用于获取资源、获取数据，比如获取网页内容、获取 `API` 数据等。

- `POST` 请求适用于提交数据、执行动作，比如表单提交、创建资源、更新数据等。

总结来说，`GET` 用于获取数据，`POST` 用于提交数据或触发服务器上的动作。它们在语义、数据位置、数据安全性和使用场景等方面有所区别。

`localStorage，session，cookie` 的区别是什么

👉👉 理解cookie、session、localStorage、sessionStorage之不同

`promise` 是做什么的

`Promise` 是 `JavaScript` 的一种异步编程解决方案，用于处理异步操作和回调函数的复杂性。它可以用于更优雅地处理异步代码，并且提供了更好的可读性和可维护性。

`promise` 的实现原理是什么

👉👉 【面试题解】详解 Promise A Plus ，从规范角度看 Promise

`let,const,var` 有什么区别

（1）块级作用域： 块作用域由 `{ }` 包括，`let` 和 `const` 具有块级作用域，`var` 不存在块级作用域。块级作用域解决了 `ES5` 中的两个问题：

- 内层变量可能覆盖外层变量
- 用来计数的循环变量泄露为全局变量

（2）变量提升： `var` 存在变量提升，`let` 和 `const` 不存在变量提升，即在变量只能在声明之后使用，否在会报错。

（3）给全局添加属性： 浏览器的全局对象是 `window`，`Node` 的全局对象是 `global`。`var` 声明的变量为全局变量，并且会将该变量添加为全局对象的属性，但是 `let` 和 `const` 不会。

（4）重复声明： `var` 声明变量时，可以重复声明变量，后声明的同名变量会覆盖之前声明的遍历。`const` 和 `let` 不允许重复声明变量。

（5）暂时性死区： 在使用 `let`、`const` 命令声明变量之前，该变量都是不可用的。这在语法上，称为**暂时性死区**。使用 `var` 声明的变量不存在暂时性死区。

（6）初始值设置： 在变量声明时，`var` 和 `let` 可以不用设置初始值。而 `const` 声明变量必须设置初始值。

（7）指针指向： `let` 和 `const` 都是 `ES6` 新增的用于创建变量的语法。`let` 创建的变量是可以更改指针指向（可以重新赋值）。但 `const` 声明的变量是不允许改变指针的指向。

区别	var	let	const
是否有块级作用域	×	✔	✔
是否存在变量提升	✔	×	×
是否添加全局属性	✔	×	×
能否重复声明变量	✔	×	×
是否存在暂时性死区	×	✔	✔
是否必须设置初始值	×	×	✔
能否改变指针指向	✔	✔	×

数组去重都有哪些方式

这个实现的方式就太多了

- `ES6 Set`
- 双重循环（`for`，`forEach`，`filter`，`reduce`，`includes` 等等任意两种遍历方式的结合）

B公司

整体总结

也是远程面试，面试官是一个小姐姐，也是只会问问题的那种，问完技术问题之后，跟我说他们公司加班挺多的，然后也没有任何加班福利，加班费啊调休啊什么的都没有，最后问我能不能接受加班，我说接受加班，不接受无偿加班，她就直接挂了，还大言不惭的说我们就是义务加班，然后也没有后续了。

说点题外话，我对加班这件事情的看法就是只要你给我钱并且不怕我死在公司的话，我可以接受零零七。但是你什么都不给，让我用爱发电，那我一分钟也不想多干，特殊情况偶尔需要加班那我就当献爱心了，形成一个固定的义务加班模式那就不太能接受了。

面试题一览

- `Vue2` 和 `Vue3` 有什么区别
- 如何用 `CSS` 实现一个三角形

- 有哪些常用的布局方式
- 如何解决浏览器的兼容问题
- `JS` 有哪些数据类型
- 如何实现一个深拷贝
- `JS` 是如何实现继承的
- 说一下原型和原型链
- `new` 的过程发生了什么
- `http` 有哪些请求方式
- `http` 状态码都有哪些，`504`，`302` 是什么
- 项目当中有哪些印象深刻的地方
- `vue` 如何实现一个自定义指令
- `vue` 当中使用了哪些设计模式
- `data` 当中数据层级很深的时候，修改数据视图没有更新怎么解决

`Vue2` 和 `Vue3` 有什么区别

见 `A` 公司相同面试题

如何用 `CSS` 实现一个三角形

👉👉 【面试技巧】老生常谈之 n 种使用 CSS 实现三角形的技巧

有哪些常用的布局方式

- **流动布局（Flow Layout）**：元素按照文档流从上到下、从左到右进行排列，自动换行。常见的网页布局默认即为流动布局。
- **弹性盒子布局（Flexbox Layout）**：使用 `display: flex` 将元素的父容器设置为弹性容器，通过使用弹性盒子属性来实现灵活排列和对齐。弹性盒子布局适用于一维排列的情况，可以轻松实现常见的水平居中、垂直居中以及等分布局等。
- **网格布局（Grid Layout）**：使用 `display: grid` 将元素的父容器设置为网格容器，通过使用网格的行（`row`）和列（`column`）的定义来实现多维排列和对齐。网格布局适用于复杂的二维布局需求，可以实现灵活的行列组合和元素位置控制。
- **定位布局（Positioning Layout）**：使用 `position` 属性和相关的定位值（如 `top`、`left`、`right`、`bottom`）来实现元素的绝对或相对定位。通过设置元素的定位属性和数值，可以将元素放置在指定的位置，并可通过 `z-index` 调整元素的层叠顺序。

- **响应式布局（Responsive Layout）**：通过使用媒体查询和百分比/弹性单位等技术，根据设备屏幕的尺寸和方向调整页面布局，以适应不同的设备和屏幕大小。

这些布局方式可以单独使用，也可以组合使用来实现更复杂和灵活的页面布局效果。

如何解决浏览器的兼容问题

👉👉 浏览器的兼容问题及解决方案整理

`JS` 有哪些数据类型

👉👉 【面试题解】JavaScript数据类型相关的六个面试题

如何实现一个深拷贝

👉👉 【面试题解】JavaScript的深浅拷贝，如何手写深拷贝?

`JS` 是如何实现继承的

👉👉 js继承的七种方式

说一下原型和原型链

👉👉 深入JavaScript系列（六）：原型与原型链

`new` 的过程发生了什么

使用 `new` 运算符创建对象时，会经历以下步骤：

- 创建一个空对象：创建一个新的空对象，该对象将成为实例化后的对象。
- 将原型连接到对象：将新创建的对象的原型指向构造函数的原型对象，通过原型链实现继承。
- 执行构造函数：将构造函数作为普通函数调用，传入新创建的对象作为执行上下文（`this`）。在构造函数内部，可以使用 `this` 关键字引用新创建的对象，并且可以添加实例属性和方法。
- 返回对象：如果构造函数没有显式返回一个对象，则返回新创建的对象；否则，如果构造函数返回的是一个对象，则返回该对象。

`http` 有哪些请求方式

在 `HTTP` 中，常见的请求方式有以下几种：

1. **GET**：用于获取资源，通过 `URL` 向服务器请求数据。
2. **POST**：用于提交数据，将数据发送到服务器进行处理。
3. **PUT**：用于更新服务器上的资源。
4. **DELETE**：用于删除服务器上的资源，将指定的资源从服务器上删除。

5. **PATCH**：用于对资源进行局部更新，只更新部分字段。
6. **OPTIONS**：用于获取服务器支持的请求方法，客户端可以使用 `OPTIONS` 请求来了解服务器支持哪些方法。
7. **HEAD**：与 `GET` 类似，但不返回实际内容，只返回响应头信息。通常用于检查资源的元数据，如是否修改过或是否存在等。

`http` 状态码都有哪些，`504`，`302` 是什么

状态码第一位数字决定了不同的响应状态，如下：

- 1 表示消息
- 2 表示成功
- 3 表示重定向
- 4 表示请求错误
- 5 表示服务器错误

1xx

代表请求已被接受，需要继续处理，这类响应是临时响应，只包含状态行和某些可选的响应信息，并一空行结束

常见的有：

- `100` （客户继续发送请求，这是临时响应） 这个临时响应是用来通知客户端它的部分请求已经被服务器接收，且仍未被拒绝。客户端印当据需发送请求的剩余部分，或者如果请求已经完成，忽略这个响应，服务器必须在请求完成后向客户端发送一个最终响应
- `101` 服务器根据客户端的请求切换协议，主要用于 `websocket` 或 `HTTP2` 升级

2xx

代表请求已成功被服务器接收，处理，并接受

- `200` （成功） 请求已成功，请求所希望的响应头或数据体将随此响应返回
- `201` （已创建）请求成功并且服务器创建了新的资源
- `202` （已创建）服务器已经接受请求，但尚未处理
- `203` （非授权信息）服务器已成功处理请求，但返回的信息可能来自另一来源
- `204` （无内容）服务器成功处理请求，但没有返回任何内容
- `205` （重置内容）服务器成功处理请求，但没有返回任何内容
- `206` （部分内容）服务器成功处理了部分请求

3xx

表示要完成请求，需要进一步操作，通常这些状态代码用来重定向

- `300` （多种选择）针对请求，服务器可执行多种操作。
- `301` （永久移动）请求的网页已永久移动到新位置。
- `302` （临时移动）服务器目前从不同位置的网页响应请求，但请求者应该继续使用原有位置来进行以后的请求
- `303` （查看其它位置）请求者应当对不同位置使用单独的 `GET` 请求来检索响应时，服务器返回此代码
- `305` （使用代理）请求者只能使用代理访问请求的网页。
- `307` （临时重定向）服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求

4xx

代表了客户端看起来可能发生了错误，妨碍了服务器的处理

- `400` （错误请求）服务器不理解请求的语法
- `401` （未授权）请求要求身份验证。
- `403` （禁止）服务器拒绝请求
- `404` （未找到）服务器找不到请求的网页
- `405` （方法禁用）禁用请求中指定的方法
- `406` （不接受）无法使用请求的内容特性响应请求的网页
- `407` （需要代理授权）此状态代码与 `401` （未授权）类似，但指定请求者应当授权使用代理
- `408` （请求超时）服务器等候请求时发生超时

5xx

表示服务器无法完成明显有效的请求。这类状态代码代表了服务器在处理请求的过程中有错误或异常状态发生

- `500` （服务器内部错误）服务器遇到错误，无法完成请求
- `501` （尚未实施）服务器服务器不具备完成请求的功能
- `502` （错误网关）服务器作为网关或代理，从上游服务器收到无效响应
- `503` （服务不可用）服务器目前无法使用，（由于超载或停机维护）
- `504` （网关超时）服务器作为网关或代理，但是没有及时从上游服务器收到请求
- `505` （ `HTTP` 版本不受支持）服务器不支持请求中所用的 `HTTP` 协议版本

项目当中有哪些印象深刻的地方

没有啥印象深刻的地方🤣🤣

`vue` 如何实现一个自定义指令

自定义指令是 `vue` 对 `HTML` 元素的扩展，给 `HTML` 元素增加自定义功能。`vue` 编译 `DOM` 时，会找到指令对象，执行指令的相关方法。

自定义指令有五个生命周期

- **bind**：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。
- **inserted**：被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)。
- **update**：被绑定于元素所在的模板更新时调用，而无论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新。
- **componentUpdated**：被绑定元素所在模板完成一次更新周期时调用。
- **unbind**：只调用一次，指令与元素解绑时调用。

`vue` 当中使用了哪些设计模式

- **观察者模式（Observer Pattern）**：`Vue` 使用观察者模式来实现数据绑定和响应式更新。`Vue` 中的数据和视图是通过观察者模式进行绑定，当数据发生变化时，会通知视图进行更新。
- **发布订阅模式（Publish-Subscribe Pattern）**：`Vue` 也使用了发布订阅模式来实现组件间的通信。`Vue` 实例通过 `$emit` 方法发布事件，其他组件通过 `$on` 方法订阅事件，从而实现了解耦和灵活的组件通信。
- **工厂模式（Factory Pattern）**：在 `Vue` 中，组件的创建使用了工厂模式。`Vue` 组件是通过 `Vue.extend` 方法创建的构造函数，然后使用 `new` 关键字实例化组件对象。
- **装饰器模式（Decorator Pattern）**：`Vue` 中的指令、计算属性、过滤器等功能都使用了装饰器模式来扩展组件的功能。通过在组件上添加不同的装饰器，可以实现不同的功能。
- **单向数据流模式（One-Way Data Flow Pattern）**：`Vue` 推崇单向数据流，即数据从父组件向子组件传递，子组件通过 `props` 接收父组件的数据，子组件不能直接修改父组件数据，通过触发事件的方式向父组件传递数据变化。

`data` 当中数据层级很深的时候，修改数据视图没有更新怎么解决

`Vue` 默认只会对已经存在的属性进行响应式处理。所以当添加一个新的属性时，`Vue` 无法检测到这个变化

使用 `Vue.set()` 或 `vm.$set()` 方法进行属性添加

C公司

整体总结

这家约面试的时候，还没来北京，所以是在线上面的。公司是研究脑机接口的，听起来很高级。面试我的应该是一个领导吧，好像没有专业的前端，整个过程五十分钟，有四十分钟在听他给我吹牛逼，剩下的十分钟就是问我以前做的一些项目，是什么方向的，面向人群是什么，跟技术相关的几乎没有，也没有后续，我猜他们是想找一个经验丰富的大佬独当一面，觉得我不太能担任起来吧。

D公司

整体总结

面试官是结合项目进行提问的，比如看到我项目用了 `nuxt`，就提问一些 `nuxt` 的问题，然后看我做过移动端，就问一下移动端的问题，面试的过程更倾向于去了解你做过什么项目，在项目中运用了哪些技术和能力，个人更偏向于这种面试方式，比八股文强，大多都是项目相关的问题，所以面试题不多。

面试题一览

- `Nuxt` 和 `Vue` 有什么区别
- 哪些生命周期在服务端执行，哪些在客户端执行
- 如何进行移动端适配
- 移动端 `1px` 问题如何解决
- 什么是 `BFC`
- `flex` 布局有哪些属性
- `es6` 有哪些常用的特性
- 箭头函数和普通函数有什么区别
- `import` 和 `require` 有什么区别
- `Vue3` 和 `Vue2` 的区别
- `express`，`koa`，和 `egg` 有什么异同
- 防抖和节流
- 什么是闭包
- 闭包会导致什么问题
- 浏览器缓存机制
- 最近正在学习的有哪些

`Nuxt` 和 `Vue` 有什么区别

`Nuxt.js` 是基于 `Vue.js` 的一个扩展框架，它在 `Vue` 的基础上提供了更多的功能和约定，用于构建更复杂的服务端渲染（`SSR`）应用

哪些生命周期在服务端执行，哪些在客户端执行

👉👉 【Nuxt】这样理解Nuxt的生命周期真是太棒了

如何进行移动端适配

👉👉 移动端适配的5种方案

👉👉 我要怎么才能实现：移动端的适配操作?

移动端 `1px` 问题如何解决

👉👉 为什么会存在1px问题？怎么解决?

什么是 `BFC`

👉👉 【面试题解】CSS布局，定位布局，浮动布局，BFC,IFC,FFC,GFC

`flex` 布局有哪些属性

👉👉 「一劳永逸」48张小图带你领略flex布局之美

`es6` 有哪些常用的特性

👉👉 阮一峰ES6 入门教程

箭头函数和普通函数有什么区别

- 箭头函数使用简洁的语法来定义函数，省略了 `function` 关键字和大括号。如果只有一个参数，还可以省略括号
- 箭头函数没有自己的 `this` 绑定，它会捕获定义时的上下文的 `this` 值，因此在箭头函数中使用的 `this` 是指向当前所处的词法作用域的 `this`，而不是像普通函数那样根据函数的调用方式而确定 `this`
- `arguments` 对象：普通函数内部可以使用 `arguments` 对象来访问所有传递给函数的参数，但是箭头函数没有自己的 `arguments` 对象。如果需要访问参数，可以使用 `rest` 参数语法

`import` 和 `require` 有什么区别

`import` 是 `ESM` 中的模块导入语法，使用关键字 `import` 加上模块路径来导入模块。例如：

```
1 import { something } from 'module';
```

而 `require` 是 `CommonJS` 中的模块导入语法，使用 `require` 函数来导入模块。例如：

```
const something = require('module');
```

`import` 和 `require` 实现了相同的功能，即将一个模块引入到当前模块中。但是，`import` 语句在导入模块时会进行静态解析，通过静态分析代码来确定导入的模块，只能在顶层使用，不能在条件语句或循环中使用。而 `require` 函数在运行时根据传入的模块路径动态加载模块，可以在代码的任何位置使用。

`Vue3` 和 `Vue2` 的区别

见 `A` 公司相同面试题

`express`，`koa`，和 `egg` 有什么异同

- `express` 是最早发布的 `Node.js Web` 框架之一，它提供了一个简单而灵活的方式来构建 `Web` 应用程序。它具有中间件架构，开发者可以根据需要自由组合中间件，实现各种功能。
- `koa` 是由 `express` 的原作者设计的新一代框架，它借鉴了许多 `express` 的优点，并在开发体验和性能方面进行了改进。相对于 `express`，`koa` 更注重异步操作的处理，通过 `async/await` 语法来简化异步流程控制。
- `egg` 是一个企业级的 `Node.js` 框架，它以 `koa` 为基础，提供了更多的约定和功能，适用于构建复杂的大型应用程序。`egg` 包含了很多可插拔的插件，如安全、认证、缓存等，开发者可以根据需要选择安装使用。

防抖和节流

👉👉 手把手教你轻松手写防抖和节流

什么是闭包

👉👉 【面试题解】初识 JavaScript 闭包

闭包会导致什么问题

👉👉 万恶的前端内存泄漏及万善的解决方案

浏览器缓存机制

👉👉 一文读懂浏览器缓存

最近正在学习的有哪些

学什么就说什么呗，没学就要说自己已经会的

E公司

整体总结

这家公司其实准备了机考题，但我当时是用手机进行的面试，所以不太方便敲代码，然后就改成了面试官口述，我嘴答，但面试完之后感觉就算是上机我也答不出来几个。基本上都是一些场景题，还是挺难的，有很多平时都是用第三方库，也没研究过怎么实现的，经过这次面试，也认识到自己的不足，以后有时间还是要看一看这些工具库的源码。

面试题一览

- 使用 `CSS` 实现一个可滚动的列表选中视图中的前三个赋予单独的样式
- 使用 `CSS` 实现一个弹窗，弹窗的内容文案少的时候是居中的，文案多的时候是换行左对齐的
- 把一个 `div` 做成宽高比 `4:3` 的矩形，距离屏幕两边有 `50px` 的距离
- 实现一个 `sleep` 方法
- 实现一个 `confirm` 组件，通过 `res = await this.$refs.confirm.show()` 方法的返回值 `res` 获取到用户点击的是确认按钮还是取消按钮
- 然后是一个手写函数柯里化的问题
- 问了几个 `http` 状态码的含义
- `http2` 相比于 `http1` 做了哪些升级
- `http` 缓存策略，缓存策略两个字段之间的关系是什么
- 链表和数组有什么区别

使用 `CSS` 给一个列表的前三个赋予单独的样式

使用 `nth-child(-n+3)`

使用 `CSS` 实现一个弹窗，弹窗的内容文案少的时候是居中的，文案多的时候是换行左对齐的

```
/*当文字为一行是，则P的宽度小于div的宽度，p标签居中显示在盒子内，文字也就居中了 ;当大于一行时，P的宽度和div的宽度是一致的 ,文字就居左对齐了*/
.content {
 width: 200px;
 border: 1px solid #ee2415;
 text-align: center  ;
 padding: 2px 5px
}
/*display: inline-block使P的宽度根据文字的宽度伸缩 */
.content p {
  text-align: left;
  display: inline-block
```

```
}
```

把一个 `div` 做成宽高比 `4:3` 的矩形，距离屏幕两边有 `50px` 的距离

`padding` 的尺寸可以根据元素的宽度进行计算，所以只需要保证 `padding-top` 和 `width` 的比例是 `4:3` 就行了。

```
width: 100%;
margin: 0 50px;
padding-top: 75%
```

实现一个 `sleep` 方法

```
const imitateDelay = (timeout) =>
  new Promise((resolve) => {
    const timeoutHandle = setTimeout(() => {
      clearTimeout(timeoutHandle);
      resolve();
    }, timeout);
  });

async function test() {
  console.log('The first log');
  await imitateDelay(1000);
  console.log('The second log with 1000 ms delay'); // => 1000 毫秒后输出 The second log with 1000 ms delay
}
```

实现一个 `confirm` 组件，通过 `res = await this.$refs.confirm.show()` 方法的返回值 `res` 获取到用户点击的是确认按钮还是取消按钮

这个问题当时是不会的，回来以后查资料才实现了，就是把 `promise` 的 `resolve` 和 `reject` 另存起来，在需要时调用

```
<div class="confirm" v-show="showConfirm">
  <div class="confirm-content">
    <p>确认要执行此操作吗? </p>
    <div class="confirm-buttons">
```

```
          <button class="confirm-btn" @click="handleConfirm">确认</button>
          <button class="cancel-btn" @click="handleCancel">取消</button>
      </div>
    </div>
</div>

 data() {
    return {
      showConfirm: false,
      resolve: null,
      reject: null
    };
  },
  methods: {
    show() {
      this.showConfirm = true;
      return new Promise((resolve, reject) => {
        this.resolve = resolve;
        this.reject = reject;
      });
    },
    handleConfirm() {
      this.resolve(true); // 用户点击了确认按钮
      this.showConfirm = false;
    },
    handleCancel() {
      this.resolve(false); // 用户点击了取消按钮
      this.showConfirm = false;
    }
  },

```

然后是一个手写函数柯里化的问题

👉👉 「前端进阶」彻底弄懂函数柯里化

问了几个 `http` 状态码的含义

见 `B` 公司相同面试题

`http2` 相比于 `http1` 做了哪些升级

👉👉 解读 HTTP1/HTTP2/HTTP3

`http` 缓存策略，缓存策略两个字段之间的关系是什么

👉👉 一文读懂浏览器缓存

链表和数组有什么区别

不知道，算法和数据结构都没了解过，所以说我进不了大厂。

F公司

整体总结

两个面试官，一个问项目上的，一个问技术，问的都挺简单的，没有什么特别之处。

面试题一览

- `Vue3` 和 `Vue2` 有什么区别
- `es6` 有哪些新的特性
- 给我展示了一段代码，让我说输出顺序，其实考的就是事件循环
- `Vuex` 由哪几部分组成
- `a` 标签打开一个新页面如何把 `sessionstorage` 给带过去
- 说一下 `Vue` 的虚拟 `dom`
- 说一下项目当中有哪些优化方式
- 有没有封装过 `axios`
- 如何解决跨域问题

`Vue3` 和 `Vue2` 有什么区别

见 `A` 公司相同面试题

`es6` 有哪些新的特性

👉👉 阮一峰ES6 入门教程

给我展示了一段代码，让我说输出顺序，其实考的就是事件循环

👉👉 做一些动图，学习一下EventLoop

`Vuex` 由哪几部分组成

有五种，分别

- **State**：定义了应用状态的数据结构，可以在这里设置默认的初始状态。

- **Getter**：允许组件从 `Store` 中获取数据， `mapGetters` 辅助函数仅仅是将 `store` 中的 `getter` 映射到局部计算属性。
- **Mutation**：是唯一更改 `store` 中状态的方法，且必须是同步函数。
- **Action**：用于提交 `mutation` ，而不是直接变更状态，可以包含任意异步操作。
- **Module**：允许将单一的 `Store` 拆分为多个 `store` 且同时保存在单一的状态树中。

👉👉 【初学者笔记】一文学会使用Vuex

`a` 标签打开一个新页面如何把 `sessionstorage` 给带过去

使用参数

```
<a href="`newpage.html?sessionData=${sessionStorage.getItem('sessionData')}`">打开新页面</a>
```

说一下 `Vue` 的虚拟 `dom`

👉👉 深入剖析：Vue核心之虚拟DOM

说一下项目当中有哪些优化方式

👉👉 前端性能优化 24 条建议

有没有封装过 `axios`

这个封装方式就太多了，但我只封装过 `JS` 的， `TS` 搞不明白

👉👉 不要再被误导了，封装 Axios 只看这一篇文章就行了

👉👉 在项目中用ts封装axios，一次封装整个团队受益

如何解决跨域问题

👉👉 九种跨域方式实现原理（完整版）

G公司

整体总结

远程，面试官说看了我的掘金，内容挺多的，就简单面试点吧，然后面试了半小时😊，不过确实八股文没问多少，主要都在跟我聊项目。

面试题一览

- `es6` 常用的内容有哪些
- 箭头函数和普通函数有什么区别
- `Vue` 的双向绑定原理
- 说一下 `MVVM` 模式
- 父子组件生命周期的渲染顺序
- 项目中哪些生命周期比较常用，有哪些场景
- `vue` 的 `nextTick` 原理是什么
- 说一下事件循环机制
- `Vue` 组件之间通讯的方式有哪些

`es6` 常用的内容有哪些

👉👉 阮一峰ES6 入门教程

箭头函数和普通函数有什么区别

见 `D` 公司相同面试题

`Vue` 的双向绑定原理

👉👉 Vue双向数据绑定原理

说一下 `MVVM` 模式

👉👉 看完这篇关于MVVM的文章，面试通过率提升了80%

父子组件生命周期的渲染顺序

`Vue` 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 **4** 部分：

- 加载渲染过程 `父beforeCreate` -> `父created` -> `父beforeMount` -> `子beforeCreate` -> `子created` -> `子beforeMount` -> `子mounted` -> `父mounted`
- 子组件更新过程 `父beforeUpdate` -> `子beforeUpdate` -> `子updated` -> `父updated`
- 父组件更新过程 `父beforeUpdate` -> `父updated`
- 销毁过程 `父beforeDestroy` -> `子beforeDestroy` -> `子destroyed` -> `父destroyed`

项目中哪些生命周期比较常用，有哪些场景

- `created` 获取数据
- `mounted` 操作 `dom`

- `beforedestroy` 销毁一些实例，定时器

`vue` 的 `nextTick` 原理是什么

`Vue` 的 `nextTick` 方法用于在下次 `DOM` 更新循环结束之后执行延迟回调。它的原理如下：

- 当你调用 `Vue` 实例的 `nextTick` 方法时， `Vue` 会将传入的回调函数放入一个待执行的回调队列中。
- 在同一个 `tick` 内，如果多次调用了 `nextTick` ， `Vue` 会将这些回调函数都放入同一个队列中。
- 当 `tick` 循环开始时， `Vue` 会先执行同步任务，然后进行异步任务的处理，其中就包括执行 `nextTick` 队列中的回调函数。
- `Vue` 会根据浏览器支持情况使用不同的异步延迟策略，如微任务（`Promise` 、`MutationObserver` ）或宏任务（`setImmediate` 、 `setTimeout` ）来处理异步任务。
- 在下次 `tick` 循环结束之后， `DOM` 更新完成， `Vue` 会执行 `nextTick` 队列中的所有回调函数。

这样，你可以通过 `nextTick` 方法来确保在 `Vue` 完成 `DOM` 更新之后再执行一些操作，例如访问更新后的 `DOM` 、执行某些操作依赖于更新后的数据等。

说一下事件循环机制

👉👉 做一些动图，学习一下EventLoop

`Vue` 组件之间通讯的方式有哪些

见 `A` 公司相同面试题

H公司

整体总结

这个公司在 `boss` 上聊天的时候，就先给你发来一堆问题，然后才约面试。看了发过来的问题，我以为面试会更有难度，没想到面试反而简单的一批。

面试题一览

`boss` 上发给我的问题

- 请问您曾使用过 `Vue.js` 和 `React` 框架开发过哪些项目？请简要描述项目背景和您的主要职责。
- 您是否熟悉 `Vue2` 和 `Vue3` ？请简要介绍这两个版本的主要区别。
- 您是否有使用 `CDN` 加速静态资源的经验？请简要描述在项目中如何实现 `CDN` 加载。

- 请问您如何使用 `Webpack` 进行项目打包、优化和部署？请举例说明在项目中如何实现压缩和优化。
- 请描述您在进行前端性能优化时，采用的一些策略和技巧。
- 如何解决不同浏览器的兼容性问题？请提供一些实际例子。
- 请介绍一下您在项目中封装过的通用组件，并描述它是如何实现的。
- 假设您需要优化一个电商首页的性能，您会采取哪些策略？请举例说明。
- 请详细介绍您在项目中实现图片懒加载的方法。
- 在您的项目中，如何实现首页内容的分模块异步加载？请举例说明。

实际面试的问题

- 在 `vue` 中切换页面如何保存状态
- `Vue3` 和 `Vue2` 有什么区别（没错，又问了一遍）
- 扩展运算符是什么
- `a` 是一个空对象，打印 `a.b` 是什么， `a.b.c` 是什么
- 如何不报错
- 如何交换两个变量的值
- 如何获取两个数组之间不同的元素
- 三个盒子如何实现左边两个右边一个
- `html` 里的 `meta` 字段都有哪些内容
- 浏览器缓存策略
- 什么是闭包，会导致什么问题
- 如果排查内存泄漏

CDN 加速静态资源

👉👉 一文明白CDN加速是个啥

`Webpack` 打包、优化和部署

👉👉 当面试官问Webpack的时候他想知道什么

👉👉 玩转 webpack，使你的打包速度提升 90%

前端性能优化

👉👉 前端性能优化 24 条建议（2020）

浏览器的兼容性问题

👉👉 浏览器的兼容问题及解决方案整理

图片懒加载

- 最简单的实现方式是给 `img` 标签加上 `loading="lazy"`
- 把图片的地址放入到 `data-src` 属性里，然后监听图片是否进入可视区域内，把 `data-src` 赋值给 `src`

如何实现首页内容的分模块异步加载

使用异步组件

在 `vue` 中切换页面如何保存状态

使用 `keep-alive`

`keep-alive` 是 `Vue` 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染，其有以下特性：

- 一般结合路由和动态组件一起使用，用于缓存组件；
- 提供 `include` 和 `exclude` 属性，两者都支持字符串或正则表达式， `include` 表示只有名称匹配的组件会被缓存， `exclude` 表示任何名称匹配的组件都不会被缓存，其中 `exclude` 的优先级比 `include` 高；
- 对应两个钩子函数 `activated` 和 `deactivated` ，当组件被激活时，触发钩子函数 `activated` ，当组件被移除时，触发钩子函数 `deactivated` 。

`Vue3` 和 `Vue2` 有什么区别（没错，又问了一遍）

见 `A` 公司相同面试题

扩展运算符是什么

👉👉 ES6中解构、扩展运算符和rest运算符

`a` 是一个空对象，打印 `a.b` 是什么，`a.b.c` 是什么

- `a.b`： `undefined`
- `a.b.c`： 报错

如何不报错

使用可选链操作符 `?.`， `a?.b?.c`

如何交换两个变量的值

👉👉 10种JavaScript交换值的方法

如何获取两个数组之间不同的元素

```
const getDifferenceFrom = (arr1, arr2) => {
  const values = new Set(arr2);
  return arr1.filter((element) => !values.has(element));
};
```

三个盒子如何实现左边两个右边一个

`flex` 布局给左边两个盒子再套一层容器，使用 `justify-content: space-between;`

`html` 里的 `meta` 字段都有哪些内容

👉👉 HTML 标签

浏览器缓存策略

👉👉 一文读懂浏览器缓存

什么是闭包，会导致什么问题

👉👉 【面试题解】初识 JavaScript 闭包

如果排查内存泄漏

👉👉 万恶的前端内存泄漏及万善的解决方案

I公司

整体总结

问了项目中什么技术栈，用过 `Vue3` 么，搭建项目的工具掌握么，有没有自己封装过插件和组件，能不能独立完成项目。面试官说话得喝的，给我一种他很骄傲的感觉，可能是性格如此。

面试题一览

- 组件间传值的方式有哪些
- 讲一下 `Vue` 指令，如何添加自定义指令
- 如何处理用户权限
- `Vue3` 和 `Vue2` 有什么区别

- `promise resolve` 之后的代码还会执行吗
- 解构赋值是深拷贝还是浅拷贝
- 如何实现单行/多行文本溢出的省略样式

组件间传值的方式有哪些

见 `A` 公司相同面试题

讲一下 `Vue` 指令，如何添加自定义指令

见 `B` 公司相同面试题

如何处理用户权限

可以使用 `VueRouter` 的路由守卫来做权限管理

`Vue3` 和 `Vue2` 有什么区别

见 `A` 公司相同面试题

`promise resolve` 之后的代码还会执行吗

会

解构赋值是深拷贝还是浅拷贝

浅拷贝

如何实现单行/多行文本溢出的省略样式

👉👉 如何实现单行／多行文本溢出的省略样式?

J公司

整体总结

`J` 公司，知名外包公司 **XX国际**，上一次在大连的时候也参加过他家的面试，问的问题就是那种找能干活的人，问的都是简单的八股文，还有我之前项目是怎么搭建的，面试过程挺顺利的，后面让我写安哥拉，我就拒绝了。

面试题一览

- `css` 选择器的权重
- `css` 中有哪些隐藏元素的方法

- `es6` 有哪些处理数组的方法
- 如何实现数组去重
- 什么是深拷贝浅拷贝，如何实现深拷贝
- 项目中如何做优化的
- 数据的类型一共有几种
- 实现一个首字母转大写的函数
- `for of` 和 `for in` 的区别

`css` 选择器的权重

👉👉 【面试题解】前端人必须掌握的13种CSS选择器

`css` 中有哪些隐藏元素的方法

👉👉 CSS中，有哪些方式可以隐藏页面元素？有什么区别

`es6` 有哪些处理数组的方法

👉👉 【面试题解】你了解JavaScript常用的的十个高阶函数么？

如何实现数组去重

见 `A` 公司相同面试题

什么是深拷贝浅拷贝，如何实现深拷贝

👉👉 【面试题解】JavaScript的深浅拷贝，如何手写深拷贝？

项目中如何做优化的

👉👉 前端性能优化 24 条建议（2020）

数据的类型一共有几种

👉👉 【面试题解】JavaScript数据类型相关的六个面试题

实现一个首字母转大写的函数

```
1 const capitalizeWord = (string) => string.replace(/\b[a-z]/g, (letter) =>
  letter.toUpperCase());
2
```

`for of` 和 `for in` 的区别

`for of` 和 `for in` 是两种不同的循环语句，用于遍历可迭代对象和枚举对象的属性，它们的区别如下：

- 遍历的对象类型不同：
  - `for of`：用于遍历可迭代对象，如数组、字符串、`Set`、`Map` 等。
  - `for in`：用于遍历对象的可枚举属性，包括继承而来的属性。
- 遍历的内容不同：
  - `for of`：用于遍历可迭代对象中的元素值。
  - `for in`：用于遍历对象的键名，即属性名。
- 遍历顺序不同：
  - `for of`：按照迭代器对象定义的顺序，从前往后依次遍历。
  - `for in`：遍历对象的属性名时，顺序不确定，可能是随机的。
- 可以使用的对象类型不同：
  - `for of`：适用于可迭代对象，如数组、字符串、`Set`、`Map` 等。
  - `for in`：适用于所有对象，包括普通对象、数组、字符串等。

K公司

整体总结

先让我讲了讲我 `GitHub` 上的几个开源项目，然后又看了看我的掘金。说我平时挺爱学习和研究东西的。问我最近在看什么，我说微前端，然后聊了聊微前端。后来又问了一些技术问题。

面试题一览

- 常见的水平垂直居中实现方案
- `CSS3` 中的伪类和伪元素的区别
- 解释一下浏览器中的同源策略
- 什么是事件冒泡和事件捕获，如何阻止事件冒泡
- 什么是事件委托
- 对比 `call` 和 `apply` 的作用和区别
- `Vue3` 和 `Vue2` 有什么区别（我都答烦了）
- `teleport` 组件是干什么用的
- `vue3` 实现一个用于表格分页的 `hook`

常见的水平垂直居中实现方案

👉👉 【面试题解】宽高固定的12种和宽高不固定的29种CSS居中方案

`CSS3` 中的伪类和伪元素的区别

在 `CSS` 中，伪类和伪元素都是用来选择 `DOM` 元素的特殊方式，但它们有一些区别。

伪类是用于选择文档树中的某些特定状态的元素，例如 `hover`、`active`、`visited` 等。它们是以冒号（:）表示的，并通过在选择器中添加伪类来使用。伪类的语法是在选择器后面使用冒号加上伪类的名称，比如 `a:hover`、`input:checked` 等。

伪元素用于在文档中的某些特定位置插入内容，例如在元素的前面、后面、内部的某个位置。它们是以双冒号（::）表示的，并通过在选择器中添加伪元素来使用。伪元素的语法是在选择器后面使用双冒号加上伪元素的名称，比如 `::before`、`::after` 等。

总的来说，伪类用于选择元素的某种状态，而伪元素用于插入额外的内容或样式。使用伪类可以改变元素的样式，使用伪元素可以在元素的特定位置插入内容。

解释一下浏览器中的同源策略

同源策略是浏览器中一种安全机制，用于限制不同源（域名、协议、端口）之间的交互。同源策略的目的是防止恶意网站通过脚本访问其他网站的敏感数据或执行恶意操作。

同源策略实行以下限制：

1. `Cookie`、`LocalStorage` 和 `IndexDB` 等存储在浏览器中的数据只能在同源网站中共享。
2. `JavaScript` 脚本只能访问同源网站的 `DOM`，不能读取或修改不同源网站的 `DOM`。
3. `XMLHttpRequest` 和 `Fetch` 等网络请求只能发起同源请求，不能访问不同源的接口。
4. `iframe` 内嵌的网页只能与父页面同源的网页进行交互。

但是，如果目标网站明确设置了允许跨域访问的响应头（`CORS`），那么浏览器将允许跨源请求。

同源策略的作用在于保护用户的隐私和安全，防止恶意网站获取另一个网站的敏感数据，并防止 `CSRF`（跨站请求伪造）等网络攻击。但同时也使得不同源的网站之间受限，为了实现跨域数据的交互，需要使用其他的机制，`如CORS`、`JSONP`、代理等。

什么是事件冒泡和事件捕获，如何阻止事件冒泡

事件冒泡和事件捕获是两种不同的事件传播机制。

事件冒泡是指当一个元素触发(或接受到)某个事件时，该事件会从最内层的元素向父元素一级一级地传播，直到传播到最外层的元素。在事件冒泡过程中，事件会经过每个祖先元素，父元素会先接收到该事件，然后是祖父元素，依次向上。

事件捕获是指当一个元素触发(或接受到)某个事件时，该事件会从最外层的元素开始，逐级向内层元素传播，直到传播到最内层的元素。在事件捕获过程中，事件会经过每个祖先元素，最内层的元素会先接收到该事件。

阻止事件冒泡，可以使用事件对象的 `stopPropagation()` 方法。这个方法可以阻止事件继续向上层元素传播，即停止事件在 `DOM` 树上的冒泡过程。例如：

```
element.addEventListener('click', function(event) {
  event.stopPropagation();
});
```

需要注意的是，阻止事件冒泡只会影响到父级元素的事件监听器，不会影响当前元素本身上的其他事件监听器。

什么是事件委托

事件委托（`Event delegation`）是一种将事件处理程序绑定到其父元素上，从而利用事件冒泡机制来处理其子元素上发生的事件的技术。

常见的事件绑定方式是将事件处理程序直接绑定到特定的元素上，例如：

```
var button = document.getElementById('myButton');
button.addEventListener('click', function() {
  console.log('Button clicked!');
});
```

然而，当有大量的子元素需要触发相同事件时，为每个子元素都绑定事件处理程序可能会导致性能问题。这时候可以利用事件委托，将事件处理程序绑定到共同的父元素上，通过事件冒泡机制捕获并处理子元素上的事件，例如：

```
var parent = document.getElementById('parentElement');
parent.addEventListener('click', function(event) {
  var target = event.target;
  if (target.tagName === 'BUTTON') {
    console.log('Button clicked!');
  }
});
```

在这个例子中，父元素 `parentElement` 上绑定了 `click` 事件处理程序，当用户点击子元素 `button` 时，事件会冒泡到父元素，并在父元素上触发 `click` 事件处理程序。通过判断 `event.target` 来确定用户点击的是哪个子元素。

事件委托的优点是减少了事件处理程序的数量，提高了性能，并且能够动态地处理新增的子元素。

对比 `call` 和 `apply` 的作用和区别

👉👉 【面试题解】你了解call，apply，bind吗？那你可以手写一个吗？

`Vue3` 和 `Vue2` 有什么区别（我都答烦了）

见 `A` 公司相同面试题

`teleport` 组件是干什么用的

把 `teleport` 组件包裹的内容传送到指定的节点中

`vue3` 实现一个用于表格分页的 `hook`

```ts
import { reactive } from 'vue';

type IGetTableData = () => void;

/**
 * 用于表格数据的分页，只处理获取数据 ，如有其它需求，自行实现
 * 需要传入一个调用获取数据API的方法,这个方法实现在外部组件中，在这个方法中需要修改 total
 * 支持传入 pageSize ，默认是20
 */
const useTablePagination = (getTableDataFn: IGetTableData, pageSize = 20) => {
  // 分页器相关数据
  const pagination = reactive({
    // 当前页码
    currentPage: 1,
    // 每页数据条数
    pageSize,
    // 总数据条数
    total: 0,
  });

  // 切换每页数据数量的回调
  const handleSizeChange = () => {
    getTableDataFn();
  };

  // 切换当前页的回调
  const handleCurrentChange = () => {
    getTableDataFn();
  };
  return {
```

```
    pagination,
    handleSizeChange,
    handleCurrentChange,
  };
};

export default useTablePagination;

```

L公司

整体总结

这就是我现在入职的公司，一面面试官主要是问了一些场景题，二面面试官主要问了一些开放式的问题，比如说如何看待微前端，如何看待 `TS`，聊了聊彼此的见解，这也是我比较喜欢的面试方式，考验人解决问题的思路还有看待事物的角度，而不是考验人背八股文的能力。

面试题一览

- `Vue2` 中 `$set` 是干什么用的，引出 `Vue` 的响应式原理
- 实现一个组件，用户可以通过 `v-model` 去使用这个组件，引出 `v-model` 的原理
- 二次封装 `el-table` 组件，用户可以传入这个 `el-table` 的 `props`，也可以传入自己组件的 `props`，引出 `$attrs` 和 `$listens`
- 增强一个组件，如何把插槽透传到组件里，引出插槽的原理
- 设计一个组件，`props` 有 `userId`，通过 `userId` 向后台获取数据
- 继续增强这个组件，`props` 发生了变化，重新获取数据
- `props` 变化了两次，但网络不确定哪个请求先回结果，如何保证页面渲染的是后一次请求的数据（数据竞争）
- 实现一个 `message` 组件，通过命令方式调用
- `git` 和 `github` 是什么关系

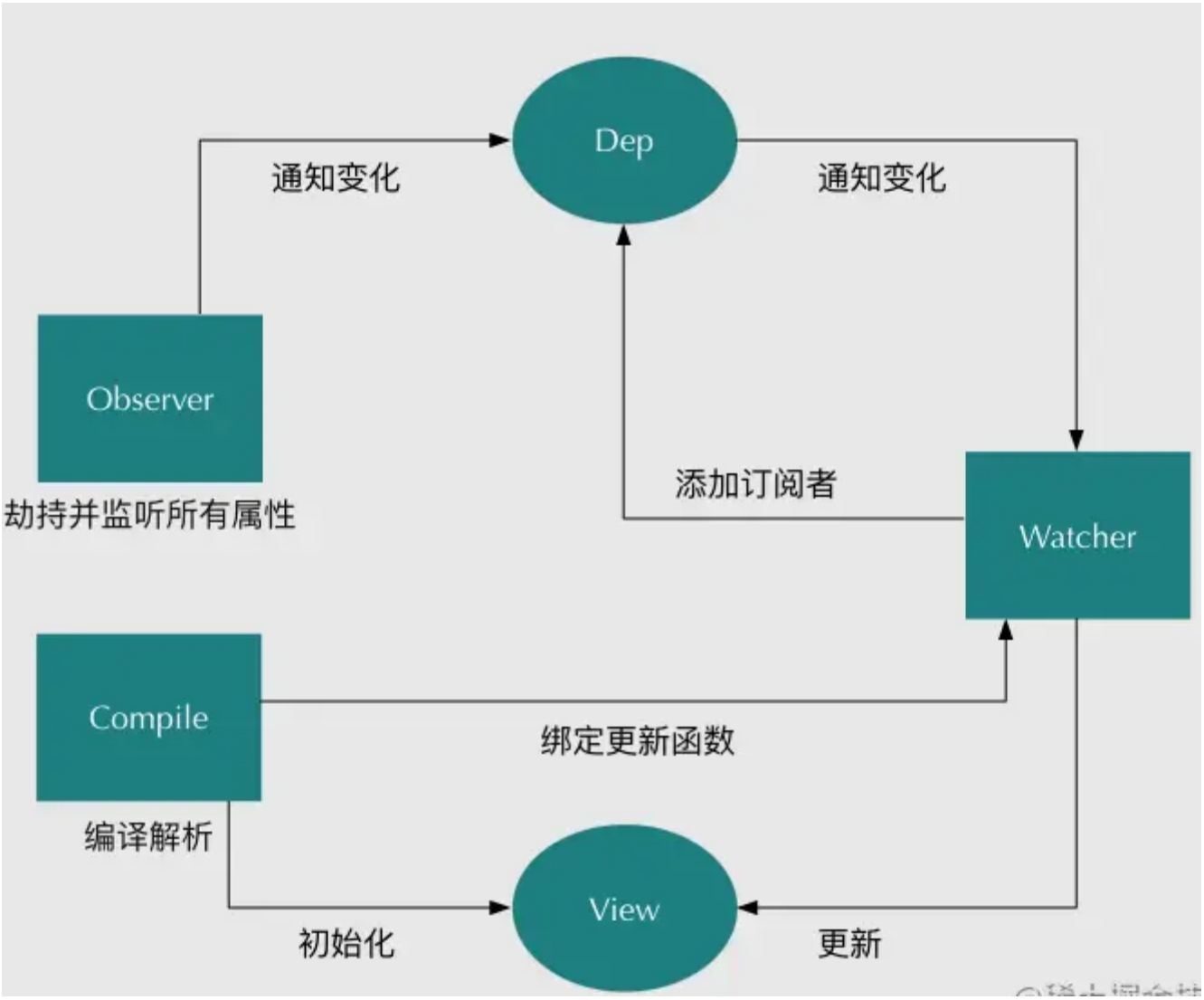
Vue 的响应式原理

`Vue.js` 是采用 **数据劫持** 结合 **发布者-订阅者模式** 的方式，通过 `Object.defineProperty()` 来劫持各个属性的 `setter`，`getter`，在数据变动时发布消息给订阅者，触发相应的监听回调。主要分为以下几个步骤：

1. 使用 `observe` 对需要响应式的数据进行递归，将对像的所有属性及其子属性，都加上 `setter` 和 `getter` 这样的话，给这个对象的某个属性赋值的时候，就会触发 `setter`，那么就能监听到了数据变化。

2. `compile` 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图。
3. `Watcher` 订阅者是 `Observer` 和 `Compile` 之间通信的桥梁，主要做的事情是:

- 在自身实例化时往属性订阅器(`dep`)里面添加自己
- 自身必须有一个 `update()` 方法
- 待属性变动触发 `dep.notice()` 通知时，能调用自身的 `update()` 方法，并触发 `Compile` 中绑定的回调，完成视图更新。 总结：通过 `Observer` 来监听自己的 `model` 数据变化，通过 `Compile` 来解析编译模板指令，最终利用 `Watcher` 搭起 `Observer` 和 `Compile` 之间的通信桥梁，达到一个数据响应式的效果。



v-model 的原理

👉👉 面试官：你真的了解 v-model 吗？(vue2)

`$attrs` 和 `$listens`

一次性传递多个属性和事件

增强一个组件，如何把插槽透传到组件里，引出插槽的原理

不会，面试官给我讲了也没听明白

👉👉 如何在Vue2/3中正确透传插槽，提升组件编写效率？

设计一个组件，`props` 有 `userId`，通过 `userId` 向后台获取数据

这没啥好说的， 都是基础，声明 `props`， `methods`，在 `created` 中发起请求

继续增强这个组件，`props` 发生了变化，重新获取数据

`watch` 这个 `props.userId`，变化的时候发起请求

`props` 变化了两次，但网络不确定哪个请求先回结果，如何保证页面渲染的是后一次请求的数据（数据竞争）

通过返回值结果和 `props.userId` 进行对比，或者拿到当次请求的请求参数和 `props.userId` 做对比

实现一个 `message` 组件，通过命令方式调用

之前实现过，不过有点忘了

👉👉 【流萤书签】基础组件(Button,Overlay,Dialog,Message)

`git` 和 `github` 是什么关系

`Git` 和 `GitHub` 是两个相关但不同的东西。

1. `Git`：是一个分布式版本控制系统，用于跟踪代码的修改和协同开发。它允许开发人员在本地创建和管理代码库，进行版本控制、分支管理和合并等操作。使用 `Git`，开发人员可以在不同的分支上进行并行开发，并轻松地合并代码。 `Git` 是一个独立的软件工具，可以在本地环境中使用。
2. `GitHub`：是一个基于 `Git` 的代码托管平台，提供了在线代码托管、版本控制、代码协作和团队协作的功能。它使用 `Git` 作为底层版本控制系统，并提供了 `Web` 界面和一系列协作功能，如 `Issue` 跟踪、 `Pull Request` 等。 `GitHub` 允许开发人员将代码存储在云端，并与团队成员共享、协作和审核代码。它是一个基于云的服务和社交平台，可以通过网页或桌面客户端访问。

综上所述， `Git` 是版本控制系统，而 `GitHub` 是基于 `Git` 的代码托管平台。开发人员可以使用 `Git` 来管理和跟踪代码的修改，使用 `GitHub` 来存储、分享和协作开发代码。

高频问题

其实高频算不上，只面试了十来家公司，样本容量太小了，但也正因为这么几家都会多次出现的题目，可见是挺热门的，结合实际的面试情况，整理出相对高频的面试题，可以着重看一下，即便是看

不懂，不感兴趣，也要强迫自己背一下。

- 一些常用的页面布局要了解
- `es6` 新特性，遍历数组的 `n` 种方法
- `JS` 原型和原型链
- 浏览器缓存策略
- `ts` 跟 `js` 有什么区别，特点，优点和缺点
- 什么是跨域问题，如何解决
- 项目中有什么亮点或者难点，如何解决的
- `Vue` 响应式原理， `diff` 算法，虚拟 `dom` ，可能是工作经验多了，相比之前更喜欢问一些原理上面的问题了
- `Vue3` 和 `Vue2` 有什么区别，这个是问的最多的，基本每家都会问，可能是上次找工作的时候， `Vue3` 还没怎么用，现在大家都开始用了，后来答的我都烦了

相比于上次找工作的高频问题，这次反而基本没怎么问了

- `CSS` 盒模型
- 前端安全问题
- 浏览器从输入 `url` 到页面渲染之间做了哪些事情
- `promise` 的使用及其原型方法
- 如果你写了自己会 `node` ，可能会问 `express` 和 `koa` 的相关问题
- `webpack` 优化

还有就是自己的项目一定要了解，虽然可能确实是自己做的，但是时间长了会忘，建议面试之前再把自己的项目好好看一下，大多数公司还是会占用一定时间去问一下你之前负责的项目的，这个答不上来大概率会让面试官觉得你项目经历是编的，或者不是自己写的，从而扣分。

必备问题

最终总结了几个面试官常问的非技术类的问题，可以提前准备一下，以免当场卡住，语无伦次导致给面试官留下不好的印象。

- 做个简单的自我介绍
- 为什么从上一家公司离职
- 你觉得上家公司做的怎么样
- 为什么要干前端
- 你平时是怎么学习的
- 看过什么书

- 你觉得你的优势/不足有什么
- 举一个例子，证明自己是乐于学习的
- 你对自己的职业规划
- 希望找一个什么样的工作
- 你还有什么想要问我的么

做个简单的自我介绍

主要介绍一下自己的工作内容，技术栈，稍微带点兴趣爱好什么的也可以。

为什么从上一家公司离职

我回答的主要两个原因，一个是想学习更多东西，另一个是老生常谈的薪资问题。

你觉得上家公司做的怎么样

先大概的夸一夸上家公司，然后再谈一谈让你不想待的原因，千万不要开吐槽大会，抱怨这个，抱怨那个的。比如说我觉得上家公司各种方面做的都不错，无论是公司环境，管理制度，团队氛围都很好，但是技术栈更新比较缓慢，调薪制度不理想。

为什么要干前端

觉得前端比较有意思，可以看到界面，可以做一下好玩的东西，给父母孩子女朋友显摆。

你平时是怎么学习的

看纸质书，电子书，视频，博客论坛。

看过什么书

看过什么就说什么，千万不要瞎编。我听说过有的面试官会问你某本书的封面是什么颜色的，目录是什么等奇葩问题。

你觉得你的优势/不足有什么

优势在于主动学习，乐于分享。不足是不感兴趣的内容学习不下去，比如数据库。

举一个例子，证明自己是乐于学习的

- 掘金社区优秀作者，社区阅读量 `50W+`
- `github` 有多少个仓库，每月有多少次提交，有多少 `star`
- 获得过上家公司的年度进步奖

你对自己的职业规划

四个字，逼格直接上来了，一专多精。

希望找一个什么样的工作

可以看下面的避雷宝典，我想找的就是没有我避雷宝典里面提到的情况的公司。

你还有什么想要问我的么

我一般会问技术团队的规模

- 技术团队有几个人，
- 几个前端，
- 几个后端，
- 高级，中级，初级的分别是多少人

然后就是技术栈

- 目前在使用什么技术栈
- 将来打算使用什么技术栈
- 自己是否可以决定未来技术栈的走向

然后是自己在团队中的角色

- 负责什么工作
- 对于项目的决策度

避雷宝典

除了上面提到的这些基本问题，还有一些结合我自己工作过的公司以及我一些前端朋友的工作经历，给大家整理了一份 **避雷宝典**，有些雷，你没有遇见过，你也不知道它的存在，你也不会去问。所以在入职前一定要确认好以下这些东西，只要是提前确认好的，哪怕比较苛刻，只要你接受就行，千万不要入职以后才发现接受不了的事情，到时候走的话浪费了时间还得重新找工作，不走的话自己待的难受。

- **加班情况**，`首先是有没有加班`，如果有的话，是固定时间加班还是随机加班。如果是固定加班的话，是怎样一个形式，大小周，九九六，还是每周四发版需要加班，加班到几点。如果是随机加班的话，加班的频次和强度，具体在什么场景下会出现加班的情况。`其次是加班福利`，包括但不限于餐补，交通补，调休，加班费。`最后是如何界定加班`，几点以后算加班，加班的最小单位是什么，加班是按照打卡时间自动计算，还是需要提交加班申请，工作日加班和休息日加班是不是一样。我一个朋友入职第一天就加班到十一点多，第二天直接离职了。

- **代码提交规范**，代码规范是什么，有没有自动化工具，有没有规范文档。提交规范是什么，有没有文档和特殊要求，分支如何管理。为什么要列这一条，是因为我有一个朋友的公司要求一个需求必须提若干个 `PR`，每个 `PR` 必须包含若干个 `commit`，同时每个 `PR` 的代码总数不能超过五百行。`Code Review` 到无所谓，毕竟自己写的代码自己有义务改到最好，但是每天除了写代码，大多的时间都用在怎么拆分 `commit` 和 `PR` 上面了就很没必要了。
- **迭代**，这一条要问清楚开发任务是如何迭代的，几周一个迭代，如何发版，需求如何下发到开发手上，工期是不是由自己评估，如果干不完如何处理延期，需求变动是延长工期还是放入下个迭代还是硬塞给你。我有一个朋友的公司就是那种最开始的任务估三天时间，然后在这三天里各种加需求，改设计图，然后工期不变，导致最后不得不加班，还是义务加班。
- **与后端沟通**，开发中是如何跟后端沟通的，接口是提前出，还是跟前端同步出，如果是同步的话，联调单独计算工期还是算在前端开发工期里，这一条我现在公司就挺好的，页面开发时间和联调时间分开的。
- **代码质量**，这个要问测试流程，有几轮测试，如何测试，对 `bug` 数有没有要求，`bug` 流转规范，我上家公司就没有这个要求，只要保证最后上线日没有 `bug` 就行，所以开发和测试相处的还算愉快，现在这个公司要求两周不能超过三个 `bug`，不然就会影响绩效，所以经常看见开发，测试，产品，设计在一块争论这是不是 `bug`，什么原因引起的，这个 `bug` 该给谁，这也是我不喜欢的一个点。
- **电脑权限**，电脑是自己带还是公司给配，自己的电脑基本没有什么问题，公司给配的话看对权限有没有要求，很多公司为了安全都不给你管理员权限，装个软件都要找负责人输入密码，但是这个还好，因为前端需要管理员权限的地方并不是很多，顺便问问是什么电脑，有没有扩展屏之类的。
- **有没有监控**，我见过工位有摄像头的，也见过给电脑屏幕装监控的，还有统计你各个软件使用时间和流量的，这个一定要问，你电脑微信聊个天，老板都知道你聊了什么，先不说涉不涉及法律问题，就说这个行为就真的很恶心。
- **应急处理**，这个是什么意思呢，公司是如何处特殊场景的，比如线上出 `bug` 了，是要随时待命，无论何时何地都要立刻解决，还是说收集整理记录放入下个开发周期。我朋友的公司项目接入了一个监控平台，只要线上有问题就会有机器人给负责人打电话，然后负责人必须在 `15` 分钟内响应（响应不是处理，就是说你已经知道这回事了），如果负责人不接电话或者没在 `15` 分钟内响应的话，机器人就会打到领导甚至老板那里去。每天都要把电脑带回家随时待命，然后他们那个项目还是国际的，有外国人在用，经常半夜两三点给他打电话。
- **开会**，这个要问都有什么会要开，每日站会，部门周会，项目周会，早会，晚会，迭代启动会，迭代中期对齐会，技术分享会，`Code Review` 大会等等，五天工期三天在开会，最气的有的会还是他妈占用你下班时间和假期开，好像把会议安排在工作日的六点前他家里就得着火一样，如果一不小心有这样的领导看到了这里，那不好意思我说的就是你。

- **钱**，普通人打工说白了就是为了钱，大家不要觉得正规公司按月按时按点就会给你卡里打钱了，拖欠工资，拖欠年终奖都是再正常不过的事了，我朋友之前有一家公司都要给你签协议，每个月的 `30%` 都会延后半年给你发，然后稍微给你那么一丢丢利息，我忘了是多少了，到时候搞得你连离职都没法离。所以一定要问清工资什么时候发，有没有年终奖，项目奖等等。公积金缴纳比例是多少，因为好多公司都给你把工资拆的可细可细了，什么绩效工资，保密金，各种奖金，然后你会发现基本工资只有几千，公积金就按照这几千块的基本工资交。
- **涨薪和晋升**，这个要问有没有涨薪和晋升机会，一定要具体，让 `HR` 给你讲清楚，有没有普调，在几月调薪，晋升机制是什么样的。有的公司会告诉你一年两次晋升机会，你以为很容易。结果去了才发现确实有两次机会，但是门槛你很难达到。比如说你现在三年经验在 `P3`，他告诉你晋升 `P4` 需要满五年经验，又或者你现在工资在一万，他告诉你晋升 `p4` 需要工资达到一万五，你就混吧，靠着普调涨到一万五，再熬到五年经验，才只是仅仅给了你一次晋升机会而已，然后第一次不给你过，这第二次机会不就用到了么。

综合上面所说的，你可能不会找到一个完美的公司，但是你一定要提前把这些乱七八糟的事情弄清楚，在不在你的接受范围里，在入职前避雷，而不是入职以后踩雷。大家如果还有什么在工作中遇到的不吐不快的事情，都可以评论一下，我给兄弟们加入到 **避雷宝典** 里。