React 面试题

1. **请介绍你在项目中使用的 React 框架的优势和适用场景**

**组件化：**React 允许开发者将 UI 拆分成独立、可复用的组件，使得开发和测试变得更加容易。

**声明式编程：**React 的声明式范式让开发者更专注于“做什么”而非“怎么做”。    - 高效的更新：通过 Virtual DOM 和 Diffing 算法，React 能够高效地更新 DOM，减少不必要的渲染。

**单向数据流：**React 的状态管理简单直观，数据从父组件流向子组件，易于跟踪和维护。

**社区支持**：React 拥有庞大的社区，提供了大量的第三方库和工具，加速开发流程。

**适用场景**：

* 适用于构建大型、复杂的单页应用（SPA）。
* 适用于需要高性能和频繁更新 UI 的应用。
* 适用于需要高度可定制和可复用 UI 组件的项目。

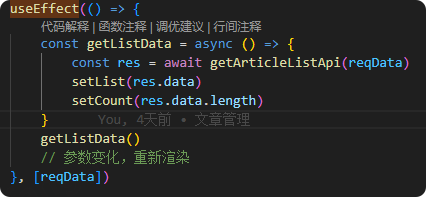
1. **请解释什么是 React 组件的生命周期，以及生命周期函数的执行顺序是怎样的**  
   **生命周期函数**：React 组件的生命周期指的是组件从创建到销毁的过程，包括初始化、更新和卸载三个阶段。

**执行顺序**:

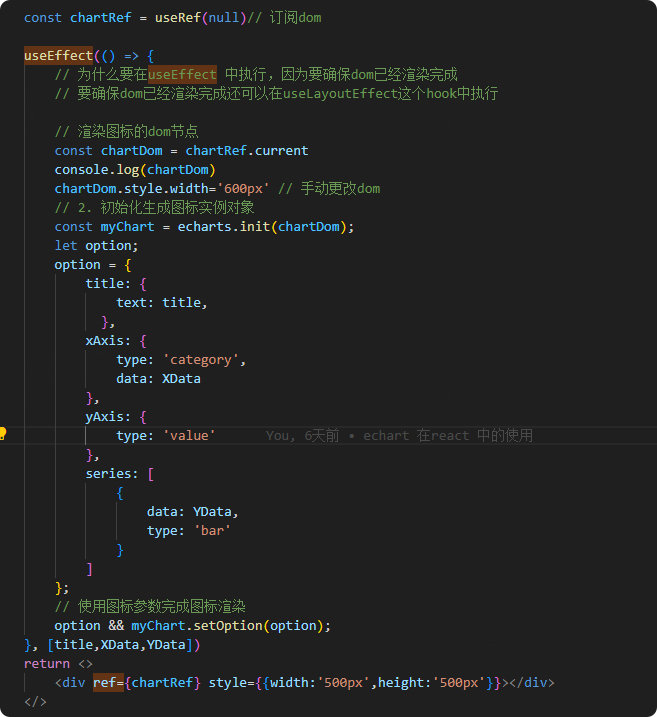
* 挂载阶段：constructor() -> render() -> componentDidMount()
* 更新阶段：render() -> componentDidUpdate()
* 卸载阶段：componentWillUnmount()

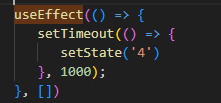
1. **在 React 中，什么是 VirtualDOM?它的作用是什么?  
   定义：**Virtual DOM 是一个轻量级的 JavaScript 对象，它描述了实际 DOM 树的结构和属性。   
   **作用：**Virtual DOM 允许 React 通过比较新旧 Virtual DOM 树的差异来最小化实际 DOM 的变更，提高性能
2. **请解释-下 React Hooks，并举例说明如何使用 useState 和 useEffect 这两个常用的Hooks**

**React hook是react16.8引入的特性，允许在不编写类组件的情况下使用状态和其他react特性，使函数组件也拥有类组件的功能  
useState：**用于在函数组件中添加状态变量。   
**useEffect：连接外部系统（有些组件需要与网络，某些浏览器API或第三方库保持连接，当他们显示在页面上时，这些系统不受react控制，所以称为外部系统）**用于在函数组件中执行常见的副作用操作（如数据获取，订阅，或手动更改dom，设置定时器等），类似于类组件的生命周期方法。

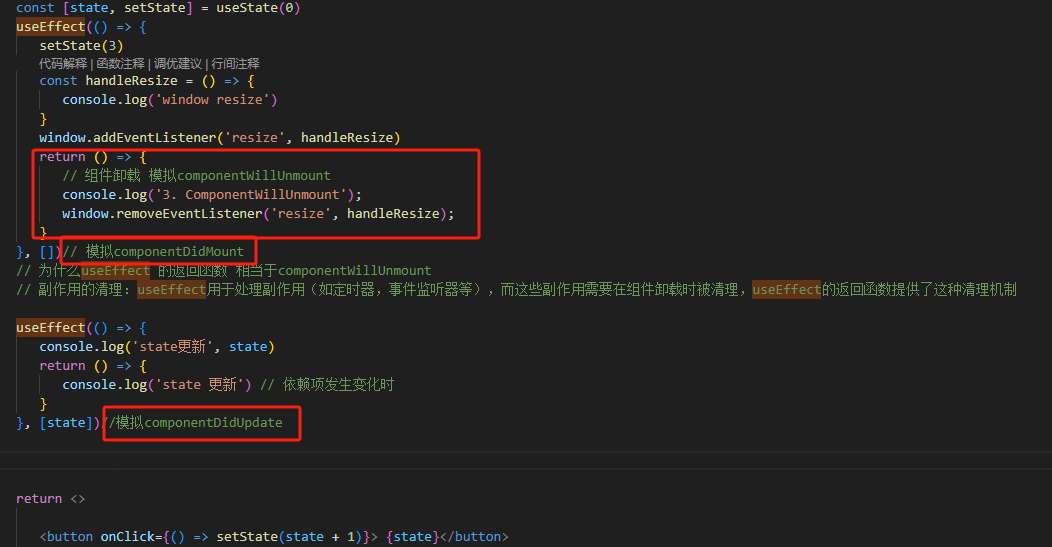
**数据获取：**

**订阅、手动更改dom：**



设置定时器：

类似类组件的生命周期方法：



1. **React 中的路由懒加载是什么?原理是什么?** 定义：在 React 应用中，路由懒加载指的是按需加载组件，而不是在应用加载时一次性加载所有组件。    
   原理：基于Es6的动态 `import()` 语法，结合react的react.lazy 和 Suspense（异步） 来实现，React 可以在需要时才加载组件，减少应用的初始加载时间。
2. **你常用的 React Hooks 有哪些?**useState:用于在函数组件中添加局部状态。他返回一个数组，其中包含当前状态值很一个允许你更改该状态的函数。这是将状态从类组件迁移到函数组件的关键hook  
   useEffect：用于执行副作用操作，比如数据获取，订阅或手动更改dom，他模拟了类组件中的生命周期方法（componentDidMount，componentDidUpdate和componentWillUnmount）。useEffect接受两个参数：一个函数和一个依赖数组。如果依赖项改变，或者组件挂载/卸载时，会执行这个函数  
   useContext：用于在组件树中深层传递上下文数据。    
   useReducer：和useState 作用类似，用来管理相对复杂的状态数据  
   useMemo：在组件每次重新渲染的时候缓存计算的结果  
   useCallback：在组件多次重新渲染的时候缓存函数（注意：useMemo是缓存值，useCallback 是缓存函数）  
   useRef：创建一个可变的引用对象，帮助引用一个不需要渲染的值。
3. **React 组件间怎么进行通信?**1.父子组件通信：父传子props 子传父 回调函数  
   2.兄弟组件通信：通过共同的父组件的状态和回调函数实现  
   3.跨层级组件通信：使用Context API 提供全局上下文 或 使用状态管理库（redux）管理全局状态 或 全局事件总线发布和订阅事件
4. **React.memo()和 useMemo()的用法是什么，有哪些区别**

**React.memo()：**是一个高阶组件，**允许组件在Prop 没有改变的情况下跳过渲染**

，**react 会对每一个prop 使用Object.is 比较新值和老值 ，返回true--表示没有变化**，避免不必要的渲染。React.memo 的第二个参数允许传入一个自定义的比较函数。True===不需要重新渲染，false===重新渲染    
 **useMemo()：**是一个 Hook，用于缓存计算结果，避免重复计算。    
**区别：**React.memo() 用于组件，useMemo() 用于函数

1. **react useMemo和uesCallback**

**为什么要有这些优化：因为默认情况下，当一个组件重新渲染时，react将递归渲染它的所有子组件**

|  |  |  |
| --- | --- | --- |
|  | useMemo | useCallback |
| 用途 | 1.缓存计算结果，减少大量运算 2.解决引用类型地址发生变化的问题，因此可以代替useCallback | 缓存函数引用 |
| 返回值 | 返回计算结果 | 返回函数引用 |
| 优化目标 | 减少复杂计算 | 减少函数引用变化 |
| 使用场景 | 计算量较大的场景 | 函数作为Props 传递给子组件的场景 |

面试题一：

1. **白屏问题：从前端角度来看：**   
   Vue项目打包的路径问题  
   路由重复或者没有配置路由

项目中使用ES6的语法，一些浏览器不支持Es  
首屏加载Js文件过大：

1. 按路由拆分和组件拆分，路由懒加载，组件按需导入
2. 代码压缩： gzip压缩
3. 减少第三方库的使用：按需引入--使用Lodash的单个函数模块，替换为轻量级的库--用dayjs替换moment.js，可以显著减小打包体积
4. **性能优化**  
   代码拆分和懒加载  
   图片和资源优化  
   使用轻量级组件  
   虚拟滚动  
   组件复用  
   减少DOM操作  
   shallowReactive和shallowRef

|  |  |  |
| --- | --- | --- |
| 方法 | Vue | React |
| 代码优化 | 1. 减少不需要的渲染：  * 使用v-memo 指令（用于缓存） * 使用computed和watch  1. 优化JavaScript代码   避免使用全局变量，尽量使用局部变量。使用let/const替代vat，减少作用域链的查找开销  使用array.prototype 方法（如map，filter、reduce）替代传统的for循环。---第二点同样适用于react   1. 使用代码分割和懒加载   使用import 动态导入组件 | 1.减少不必要的渲染：   * 使用React.memo包裹组件 * 使用useCallback 和 useMemo 缓存函数和计算结果 * 避免在组件中直接使用箭头函数，因为会在每次渲染时创建新的引用   2.使用React.lazy 和 Suspense 实现组件的懒加载 |
| 资源管理 | 1. 优化图片资源 使用图片懒加载技术，使用图片cdn 加速图片加载 2. 优化css 和 js 文件 使用tree-shaking 去除未使用的代码 使用代码压缩工具压缩js 和css 文件 3. 减少http请求 合并小图片为精灵图 使用css 预处理器（sass，less）合并样式文件 | 同左 |
| 网络优化 | 1. 使用Gzip 压缩 2. 使用强缓存和协商缓存优化资源缓存 | 同左 |
| 前端框架选择 | 根据项目需求选择合适的框架：react，vue、angular，使用轻量级的库（如Preact 替代react、vue3替代vue2）  使用vite 的原生ESM 支持快速编译和热重载 |  |
| 用户体验优化 | 1. 使用骨架屏在内容加载时提供视觉反馈 2. 减少滚动阻塞 避免在滚动事件中执行复杂的逻辑 使用防抖和节流优化滚动事件 3. 优化交互性能 使用css动画替代 js 动画，减少主线程压力 |  |

1. **https/验证证书**
2. **Vue SSR**  
   SSR首屏加载速度，SEO友好，服务端渲染API  
   或者使用nuxt—但没用过
3. **Vuerouter 动态路由**  
   底层Vue Router 利用Vue的相应系统来监听路由变化  
   渲染（RouterView）跳转（RouterLink）监听（currentRoute）  
   导航守卫  
   懒加载
4. **Vue2/Vue3异同**

|  |  |  |
| --- | --- | --- |
| 方面 | Vue2 | Vue3 |
| 设计理念与架构 | Option api | Composition api |
| 响应式系统 | Object.defineProperty 只能对已经存在的属性进行拦截，无法监听对象属性的动态添加和删除—vue.use() 无法监听数组下标的变化和数组长度的变化—重写数组方法解决 递归对每个属性监听，如果对象结构复杂—导致性能问题 | 基于ES6 proxy 代理整个对象及其嵌套属性，无需递归遍历每个属性，通过proxy里面各个方法可以监听对象的所有操作（例如get，set，deleteProperty，has），无论时属性的读取、设置，还是对象的删改 |
| 性能优化 |  | 在编译阶段进行了优化，如静态提升和diff算法的改进提升了性能，还引入了新的异步组件功能，可以按需加载组件 |
| 新特性 | 无 | Fragment，teleport，suspense ，<script setup>语法糖等新特性  Teleport 可以将组件内部的一部分模板 传送 到该组件的 dom结构外层的位置上 suspense 可以等待整个多层级组件树中的各个异步依赖获取结果时，在顶层展示出加载中、加载失败的状态 |
| Typescript 支持 |  | 完全用typescript 重写，提供更好的类型推断 |
| 生命周期钩子 | beforeCreate、created、mounted | beforeCreate和created 被setup 替代，其他钩子名前加了on，如“onMounted“，且需要在setup中使用 |
| 组件根节点 | 组件必须有唯一的跟标签 | 支持fragment，允许组件有多个根节点 |
| Tree-shaking | 整个库体积较大，即使没用的功能也会被打包进去 | 通过Tree-shaking 优化，使得核心库体积更小 |
| 事件总线 | 使用事件总线（event bus） | 移除了事件总线，推荐使用 provide/inject 或状态管理库（如pinia） 如果vue3 想使用eventBus 可以通过mitt 这个库实现 |
| 过滤器 | 支持过滤器 | 移除过滤器，推荐使用计算属性或方法代替 |
| v-model | 仅支持单个 v-model 绑定 | 支持多个 v-model 绑定 （v-model：title） |

1. **依赖收集----不熟**

实现步骤：数据声明，Proxy代理，依赖收集，响应式效果，触发更新

1. **vue和react的异同**

|  |  |  |
| --- | --- | --- |
|  | vue | react |
| 相同点 | 1. 组件化开发 2. 虚拟dom提高性能，减少不必要的dom操作 3. 响应式数据绑定，使得数据变化能够自动触发视图更新 4. 生态系统丰富 | 同左 |
| 不同点 |  |  |
| 模板语法 | HTML模板 | JSX |
| 响应式机制 | Proxy/object.defineProperty 实现数据的自动追踪和更新。Vue的响应式机制更加自动化 | 通过setState 手动触发组件的重新渲染。  React的手动更新机制更加灵活 |
| 性能优化 | 通过proxy 和 weakMap 实现响应式，性能优化—依赖收集和更新机制上 | 通过React.memo，useMem，.useCallback 手动优化 |
| 社区和生态系统 | 文档清晰，适合中小规模项目快速开发，简洁，快速上手 | 适合各种规模项目，生态系统复杂，需要花费更多时间 |
| 框架/库的定位 | Vue是一个完整的框架，提供了从模板到状态管理的完整解决方案 | React是一个库，主要关注视图层，需要结合其他库（如redux、react router）；来构建完整的应用 |

相同点：组件化开发 虚拟DOM 响应式数据绑定 生态系统丰富  
不同点：**模板语法**(vue-html react-jsx)   
**响应式机制**：vue-通过proxy和object.defineproperty 数据自动更新 react 需要 setState 手动触发  
**性能优化**：vue：通过proxy 和 weakMap 实现响应式，性能优化—依赖收集和更新机制上 react 通过React.memo,useMemo.useCallback 手动优化

**状态管理** vue-- vuex和pinia react-- **Redux**  
**数据流**：vue 通过v-model双向数据绑定 react 单向数据流，回调函数子传父

**2.强缓存协商缓存**  
强缓存：一种不与服务器交互的缓存机制，浏览器在请求资源时，会先检查本地缓存是否存在可用的资源，有则使用缓存资源，不发送请求   
协商缓存：一种与服务器交互的缓存机制，浏览器在请求资源时，会向服务器发送条件请求，服务器根据这些条件判断资源是否被修改过

|  |  |  |
| --- | --- | --- |
| 特性 | 强缓存 | 协商缓存 |
| 是否与服务器交互 | 不交互，直接使用缓存资源 | 需要与服务器交互，验证资源是否更新 |
| 实现方式 | Expires cache-control | Last-Modified ETag |
| 优点 | 减少网络延迟，提高加载速度 | 确保数据一致性 |
| 缺点 | 可能导致数据不一致性 | 增加网络请求开销 |
| 使用场景 | 不经常变动的资源—静态图片，css文件 | 经常变动的资源-动态内容 |

1. **webpack和vite的区别**  
   webpack 适合大型复杂项目，需要高度定制化和强大的插件生态系统  
   vite 使用中小型项目，追求快速开发体验和简洁配置

|  |  |  |
| --- | --- | --- |
| 特性 | Vite | Webpack |
| 构建速度 | 极快 | 较慢，全量打包 |
| 打包方式 | 开发环境不打包，生产环境用Rollup | 全量打包 |
| 热更新 | 快速，基于原生ESM | 较慢，受项目规模影响 |
| 配置复杂度 | 简单，开箱即用 | 复杂，需手动配置 |
| 生态系统 | 较新，正在发展中 | 成熟，插件丰富 |
| 适用场景 | 开发环境、中小型项目 | 大型项目、复杂场景 |
| 浏览器兼容性 | 现代浏览器 | 现代和旧版浏览器 |

1. **4.vite生产环境 开发环境**

|  |  |  |
| --- | --- | --- |
| 特性 | 开发环境 | 生产环境 |
| 构建工具 | 使用esbuild、快速编译和热重载 | 使用rollup，优化打包和压缩 |
| 环境变量 | 加载 .env.development 文件 | 加载 .env.production 文件 |
| 功能 | 实时预览、热更新、调试友好 | 代码压缩、Tree Shaking、资源优化 |
| 启动命令 | Vite 或 vite dev | Vite build |
| 多环境支持 | 支持自定义环境模式 | 支持自定义环境模式 |



1. **webpack file-loader和url-loader**

这两个是用于处理静态资源（图片，字体，文件），webpack本身是一个模块打包器，核心功能是将各种资源（如JavaScript，css，图片等）打包成一个或多个bundle文件，webpack通过以下步骤来完成打包任务：

* 解析入口文件：从配置的入口文件开始解析
* 解析依赖：递归解析入口文件中引入的依赖（如import 或 require 的模块
* 转换模块：对每个模块进行处理，使其符合 webpack 的打包要求
* 生成bundle：将处理后的模块打包成一个或多个输出文件

在这个过程中，webpack 默认只能处理JavaScript文件，对于其他类型的文件（如css，图片，字体等），webpack需要借助loader 来进行转换

|  |  |  |
| --- | --- | --- |
| 特性 | File-loader | url-loader |
| 功能 | 复制文件到输出目录，返回文件url | 小文件转换为Base64，大文件退化为file-loader |
| 适用场景 | 大文件或需要单独存储的资源 | 小文件，减少HTTP请求 |
| 配置 | 需要指定输出路径和文件名模块 | 需要设置limit和fallback |
| 性能优化 | 避免嵌入大文件、减少打包体积 | 减少http请求，适合小文件 |

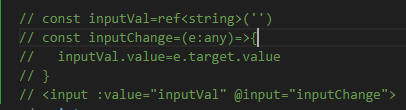
回答要点：

* File-loader 用于处理较大的静态资源文件，将文件复制到输出目录并返回url，适合图片，字体等资源
* url-loader是file-loader 的扩展，适用于小文件，他会将文件内容转换为base64 编码嵌入代码中，减少http请求
* 在配置中，url-loader 的limit 参数用于设置文件大小的阈值，超过阈值时，会退化为file-loader

**7.vue watch和 watchEffect**  
都是vue3 提供的用于响应式的监听数据变化并执行副作用的api

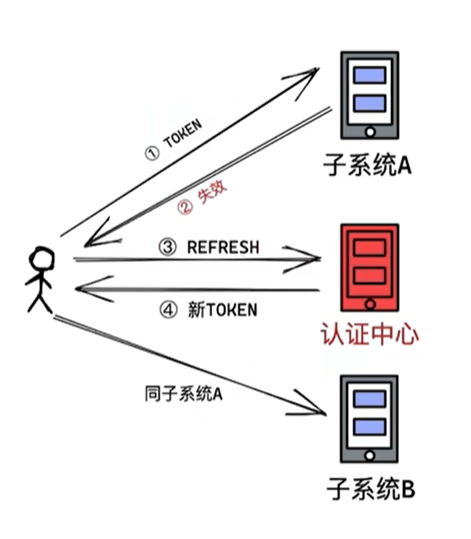
|  |  |  |
| --- | --- | --- |
| 特性 | Watch | watchEffect |
| 用途 | 监听特定数据源的变化 | 自动收集依赖并执行副作用 |
| 回调执行时机 | 默认不立即执行，但可以通过options 参数中的immediate：true 来实现 | 默认立即执行，类似于watch的immediate：true模式 |
| 依赖收集方式 | 显式指定监听的数据源 | 自动收集依赖 |
| 控制能力 | 提供更多控制选项（如immediate，deep） | 更简洁，适合快速实现副作用 |
| 适用场景 | 需要监听特定数据源或控制回调函数执行时机的场景 | 适合不需要显示指定监听数据源的场景 |

1. **vue MVVM**  
   MVVM 是一种软件架构设计模式，主要目标是将应用程序的ui与其底层数据模型分离，通过数据绑定实现数据和ui的自动同步，从而降低代码的耦合度， 开发者只需要专注于数据的处理和业务逻辑的编写，从而无需过多关心ui的更新  
   三个组成部分：  
   **Model（模型）：**  
   代表应用程序的数据模型或业务逻辑，通常包括数据结构，网络请求，处理数据的获取，更新和存储  
   Model 并不直接与UI层交互，它只暴露一些接口共ViewModel 层调用，使得ViewModel 可以获取所需的数据  
   **View（视图）：**用户界面部分，负责展示数据和用户应用的交互。通常由html，css 和 模板组成，不包含业务逻辑  
   **ViewModel（视图模型）**：  
   作为Model和View 之间的桥梁，处理UI逻辑和数据绑定，负责处理数据的变化和视图的更新  
     
   数据绑定：  
   数据绑定是MVVM 框架的核心特性之一。他将View和ViewModel 的数据同步连接
2. **单向数据流**vue，react，angular 都是单向数据流，  
   vue是双向数据绑定 **概念：单向数据流是一种数据管理方式，指的是数据的流向是单向的，从顶层组件流向子组件，通常通过props 传递**  
   子组件不能直接修改父组件传递的props  
   在vue中 通过事件emit 通知父组件来更新数据  
   在react中，通过回调函数通知父组件更新数据  
     
   数据双向绑定：指的是数据在视图（ui）和模型（数据源）之间自动同步，当视图中的数据发生变化时，模型中的数据也会自动更新，反之亦然



v-model：v-bind数据绑定+v-on处理函数绑定的语法糖

1. **寻找字符串数组中的最长共通前缀字符串  
   https://blog.csdn.net/qq\_44062823/article/details/147025841?spm=1001.2014.3001.5501**
2. **前后端分离的好处**前后端分离是现在开发中一种常见的架构模式，他将前端（用户界面）和后端（服务器逻辑）的开发工作分开进行，这样前端后可以并行独立开发，并且前后端技术栈独立，前端代码可以更容易的复用和维护，用户体验页面响应速度快，也会有更好的交互体验
3. **单点登录**一种身份验证和授权机制，是在企业内部多个应用系统场景下，用户只需要登录一次，就可访问多个应用系统  
   token+refreshToken模式  
   首次登录时，服务器会返回两个token，accessToken和refreshToken  
   前端将这两个token 存到localStorage中  
   accessToken过期时间比较短  
   refreshToken 过期时间相对较长  
   请求时请求头中添加accessToken发送验证，如果accessToken 有效—则返回正常请求结果；  
   但是由于token过期时间短，所以容易失效，这时候用户用refreshToken 验证  
   有效--返回请求结果和新的accessToken和新的refreshToken  
   无效—提示用户进行重新登录操作

后端可以HTTP设置cookie时，提供了2个属性，可以增强cookie的安全性，分别是secure属性和httpOnly属性（防止xss攻击）。 **后续复习可参考这个  
https://www.cnblogs.com/Linwei33/p/18493726**

1. **promise 和async/await**promise 时es6引入的异步解决方案，用于代替传统的回调函数  
   promise有三种状态：  
   pending（进行中）：初始状态  
   fulfilled（已完成）：意味着操作成功完成  
   rejected（已拒绝）：意味着操作失败  
   一旦promise 的状态改变，就不可再变  
   promise的链式调用：  
   promise支持链式调用，因为.then()和.catch() 方法都会返回一个新的promise  
   这使得多个异步操作可以按顺序执行，同时避免了回调地狱  
   promise.all 和 promise.race 有什么区别：  
   promise.all 接收一个promise数组作为参数  
   当所有promise 都成功完成时，返回一个新的promise，结果就是一个包含所有promise 结果的数组  
   如果有任何一个promise 失败，promise.all 会立即失败，并返回失败的promise 的错误  
   promise.race 接收一个promise数组作为参数  
   返回第一个完成的promise结果，无论是成功还是失败  
   async/await::  
   是为了简化promise 的使用的语法糖  
   async 关键字用于声明一个函数是异步的，该函数会返回一个promise  
   await 关键字用于等待一个promise 的完成，他只能在async函数中使用  
   async/await 让异步代码看起来像同步代码  
     
   async/await 和 promise 的区别是什么  
   promise 用.then 和 .catch 处理异步操作  
   async/await 使用await 关键字等待promise 完成  
     
   async/await 让异步代码看起来更像同步代码，逻辑更清晰  
   promise 链式调用在处理多个异步操作时可能会显得复杂  
     
   可以使用promise.all或promise.race 来处理并发异步操作  
   在async/await 中可以将promise.all或promise.race 的结果赋值给变量，然后使用await 等待
2. **vue权限控制**
3. **箭头函数和普通函数有什么区别**
4. **Proxy Reflect 是什么？在vue3的作用**
5. **vue3 宏definePropsdefineEmitswithDefaults**
6. **SSR**  
   前端的SSR通常指的是“服务器端渲染”（Server-Side Rendering），这是一种将网页内容在服务器端生成并发送到客户端的技术。与传统的客户端渲染（CSR，Client-Side Rendering）相比，SSR可以带来以下优势：  
   1. SEO优化：搜索引擎更容易抓取服务器端渲染的页面，因为内容在服务器上就已经生成好了。  
   2. 首屏加载速度：由于页面内容在服务器上生成，用户可以更快地看到页面的完整内容，提高用户体验。  
   3. 减轻客户端负担：对于性能较差的设备，服务器端渲染可以减少客户端JavaScript的执行量，从而提升性能。SSR在前端开发中常用于构建单页应用（SPA），例如使用React、Vue或Angular等现代JavaScript框架时，可以通过SSR框架如Next.js（React）、Nuxt.js（Vue）等实现。
7. **20.vue-loader**
8. **21.vue defineAsyncComponent**  
   https://kimi.moonshot.cn/share/ct1egejdf0j8vsgmpjkg  
   **22.vue3 ref如何实现响应式原理和reactive有什么不同**
9. **23.setup 语法糖 和 expose/defineExpose**
10. **24.vue this getCurrentInstance**
11. **25.vue组件作用域**

**26.scoped**  
.read-the-docs[data-v-e17ea971] {    
 color: #888;

}

1. **27.sse协议和实现逻辑**
2. **28.vite 分包基于HTTP长连接**

**29.动态执行 js**  
https://kimi.moonshot.cn/share/ct1f4p21bmk5dtj7m9lg

1. **30.防抖/节流**

<https://kimi.moonshot.cn/share/ct1f7o44aveljhh74pcg>

1. **.vue2和vue3 ref获取到的子组件有什么区别 对应的事件和属性有什么不同？**
2. **css display:none、visibility:hidden 和 opacity:0 之间的区别**
3. **vue3 取消watch**
4. **vue3 watch 和 watchEffect 区别**
5. **vue2 组件传值**
6. **vuex 的各个属性**
7. **vue2 父子组件生命周期的顺序**
8. **v-for v-if 连用**
9. **v-show v-if 区别**
10. **子组件同时监听父组件props的两个值**
11. **watch 中的show 和 组件上的show哪个先变化**
12. **在watch中获取dom**
13. **position属性（如absolute、fixed或relative）**
14. **项目中做了哪些优化**
15. **vueRouter 动态路由**
16. **watch watchEffect computed 区别**
17. **css 水平垂直居中有哪些方式**
18. **React Hooks**
19. **toast 封装**
20. **React 如何监听**
21. **给一个m\*n的网格，左上角是起始位置（0,0），右下角是（m,n）,问走到（x,y）有几种方法？注意：只能向下或向右移动（leetcode）**

大疆面试题

1. **1.水平垂直居中，分别都说出两种**
2. **2.block和inline的区别**
3. **3.选择器的优先级**
4. **4.BFC讲一下：创建条件，作用**
5. **5.relative和absolute的区别**
6. **6.判断js数据类型哪些方法**
7. **7.防抖节流：手撕**
8. **8.读代码输出题：关于this指向的，还有settimeout，async等同步任务、微任务执行顺序之类的**
9. **9.localstorage和sessionstorage的区别**
10. **10.同源策略以及跨域解决方案**
11. **11.cora跨域是在后端还是前端设置**
12. **12.项目中使用过什么跨域方法**
13. **13.CSRF攻击说一下，还有什么其他的攻击？**
14. **14.TCP三次握手和四次挥手，为什么是三次握手而不是两次，为什么是四次挥手而不是三次**
15. **15.为什么项目不用vue3做**
16. **16.webpack了解嘛？描述一下**
17. **17.vue的生命周期，以及ajax请求在哪个周期发送**
18. **1.对低代码的理解**
19. **2.对路由守卫的理解，有哪些，怎么设置，作用是什么**
20. **3.权限的操作；动态路由**
21. **4.你觉得公司开发一个项目和大学课程上有什么区别**
22. **5.组件之间通信**
23. **6.父子传值和vuex的区别和应用场景**
24. **7.100张图片需要加载，怎么优化？（图片懒加载）**
25. **8.怎么获取图片是不是在可视区域内？有哪些方法？**
26. **9.创建一个对象的原理？如果返回的不是对象而是一个数字呢？**
27. **10.xss攻击**
28. **11.事件代理和proxy，为什么要有事件代理**
29. **12。为什么选前端，为什么不选后端或者算法？**
30. **13.有没有实习过？主要做什么工作？**
31. **反问环节。问了什么时候反馈结果，面试官说你是急着对比offer吗，我说不是不是，尴尬，不过面试官挺和蔼的，体验良好**
32. **rem em**
33. **Vue keep-alive 组件**
34. **CSS BFC**
35. **token/Cookie/SessionStorage/LocalStorage**
36. **sass和css 使用变量，有哪些异同**
37. **路由传参如何监听**
38. **css 两列布局 撑满**
39. **ts 常用**
40. **手写call、apply、bind函数**
41. **从数组中查找特定元素**
42. **将对象的值收集到数组中**
43. **判断对象是否为空**
44. **Js拷贝**
45. **CSS垂直居中**
46. **CSS左边固定右边自适应**
47. **Vue父子组件生命周期顺序创建：**父beforeCreate 父created 父beforeMount 子beforeCreate 子created 子beforeMount 子mounted 父mounted更新：父beforeUpdate 子beforeUpdate 子Updated 父updatedv3销毁：父组件 beforeUnmount 子组件 unmounted 父组件 unmountedv2销毁：父beforeDestroy->子beforeDestroy->子destroyed->父destroyed
48. **Vue3.0 和 2.0 的响应式原理区别**
49. **nextTick 使用场景和原理**
50. **toArray 类数组转化为数组**
51. **.once 只执行一次**
52. **.数组扁平化JSON.stringify() 和 JSON.parse()**
53. **Ts中定义类型的方式**
54. **Ts中type 和 interface 的区别**
55. **以下分别打印什么**  
    for (var index = 0; index < 10; index++) {  setTimeout(() => {  console.log(index)  }, 0);}for (let index = 0; index < 10; index++) {  setTimeout(() => {  console.log(index)  }, 0);}
56. **vueRouter 钩子函数**
57. **isObject/isObjectLike/isPlainObject**