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。

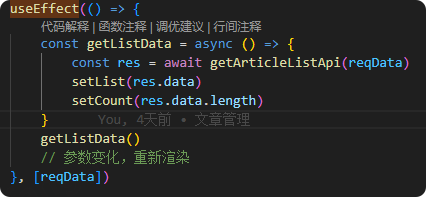
1. **请解释什么是 React 组件的生命周期，以及生命周期函数的执行顺序是怎样的**  
   **生命周期函数**：React 组件的生命周期指的是组件从创建到销毁的过程，包括初始化、更新和卸载三个阶段。

**执行顺序**:

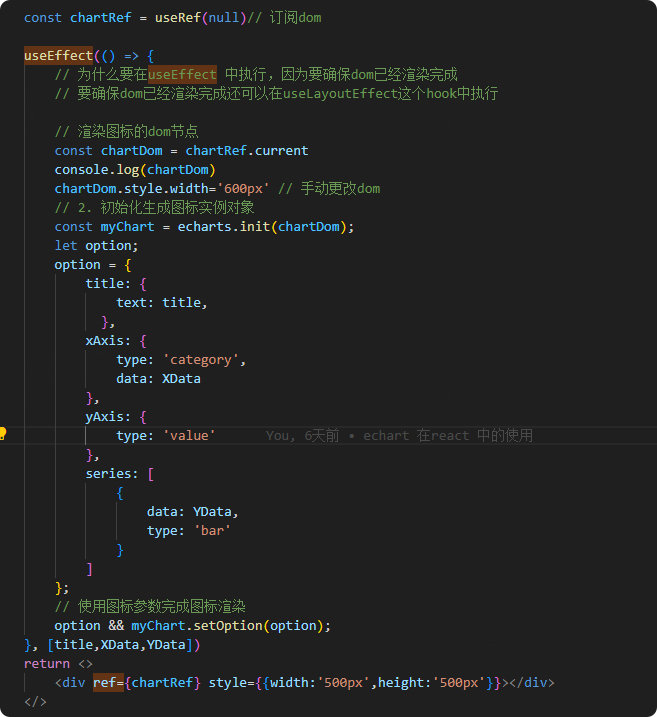
* 挂载阶段：constructor() -> render() -> componentDidMount()
* 更新阶段：render() -> componentDidUpdate()
* 卸载阶段：componentWillUnmount()

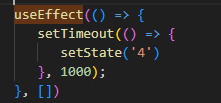
1. **在 React 中，什么是 VirtualDOM?它的作用是什么?  
   定义：**Virtual DOM 是一个轻量级的 JavaScript 对象，它描述了实际 DOM 树的结构和属性。   
   **作用：**Virtual DOM 允许 React 通过比较新旧 Virtual DOM 树的差异来最小化实际 DOM 的变更，提高性能
2. **请解释-下 React Hooks，并举例说明如何使用 useState 和 useEffect 这两个常用的Hooks**

**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。

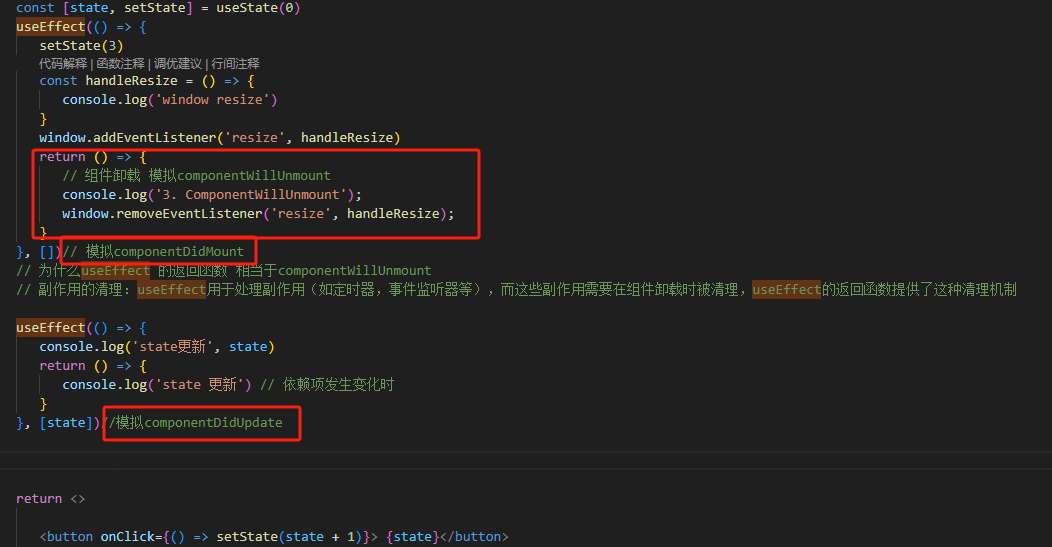
**数据获取：**

**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 | 缓存计算结果 | 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
5. https/验证证书
6. 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
7. 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
8. Vue2/Vue3异同  
   Object.defineProperty / proxy

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，需要手动调用 vue.set()   
无法监听数组下标的变化或数组长度的变化—重写数组方法解决  
object.defineProperty 递归对每个属性监听，如果对象结构复杂==导致性能问题

option api / composition api  
Typescript 的集成

vue3 性能优化---静态提升、diff算法、异步组件可以按需加载组件

vue3 Fragment Tree-shaking 移除了过滤器-用计算属性和方法代替 支持多个v-model  
新特性：如<script setup>语法糖、<Teleport>组件、<suspense>

生命周期钩子  
模板语法  
编译器优化

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  
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 |

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 |



5.webpack file-loader和url-loader

7.vue watch和 watchEffect

8.vue MVVM

9.单向数据流

10.寻找字符串数组中的最长共通前缀字符串

11.前后端分离的好处

12.单点登录

13.promise 和async/await

14.vue2和vue3的区别

15.vue权限控制

16.箭头函数和普通函数有什么区别

17.Proxy Reflect 是什么？在vue3的作用

18.vue3 宏definePropsdefineEmitswithDefaults

19.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。

20.vue-loader

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

23.setup 语法糖 和 expose/defineExpose

24.vue this getCurrentInstance

25.vue组件作用域

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

}

27.sse协议和实现逻辑

28.vite 分包基于HTTP长连接

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

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.水平垂直居中，分别都说出两种

2.block和inline的区别

3.选择器的优先级

4.BFC讲一下：创建条件，作用

5.relative和absolute的区别

6.判断js数据类型哪些方法

7.防抖节流：手撕

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.localstorage和sessionstorage的区别

10.同源策略以及跨域解决方案

11.cora跨域是在后端还是前端设置

12.项目中使用过什么跨域方法

13.CSRF攻击说一下，还有什么其他的攻击？

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.为什么项目不用vue3做

16.webpack了解嘛？描述一下

17.vue的生命周期，以及ajax请求在哪个周期发送

1.对低代码的理解

2.对路由守卫的理解，有哪些，怎么设置，作用是什么

3.权限的操作；动态路由

4.你觉得公司开发一个项目和大学课程上有什么区别

5.组件之间通信

6.父子传值和vuex的区别和应用场景

7.100张图片需要加载，怎么优化？（图片懒加载）

8.怎么获取图片是不是在可视区域内？有哪些方法？

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？

10.xss攻击

11.事件代理和proxy，为什么要有事件代理

12。为什么选前端，为什么不选后端或者算法？

13.有没有实习过？主要做什么工作？

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

1. **rem em**
2. **Vue keep-alive 组件**
3. **CSS BFC**
4. **token/Cookie/SessionStorage/LocalStorage**
5. sass和css 使用变量，有哪些异同
6. 路由传参如何监听
7. css 两列布局 撑满
8. ts 常用
9. 手写call、apply、bind函数
10. 从数组中查找特定元素
11. 将对象的值收集到数组中
12. 判断对象是否为空
13. Js拷贝
14. CSS垂直居中
15. CSS左边固定右边自适应
16. 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
17. Vue3.0 和 2.0 的响应式原理区别
18. nextTick 使用场景和原理
19. toArray 类数组转化为数组
20. .once 只执行一次
21. .数组扁平化JSON.stringify() 和 JSON.parse()
22. Ts中定义类型的方式
23. Ts中type 和 interface 的区别
24. 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);}
25. **vueRouter 钩子函数**
26. **isObject/isObjectLike/isPlainObject**