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。**

**声明式程序设计：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。**

**小区支持：React 拥有庞大的小区，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。**

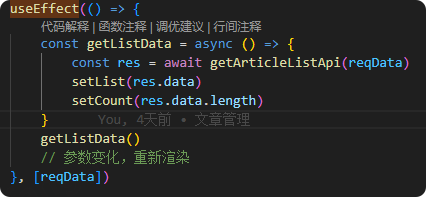
1. **请解释什么是 React 组件的生命周期，以及生命周期函数的执行顺序是怎样的**  
   **生命周期函数：React 组件的生命周期指的是组件从创建到销毁的过程，包括初始化、更新和卸除三个阶段。**

**执行顺序:**

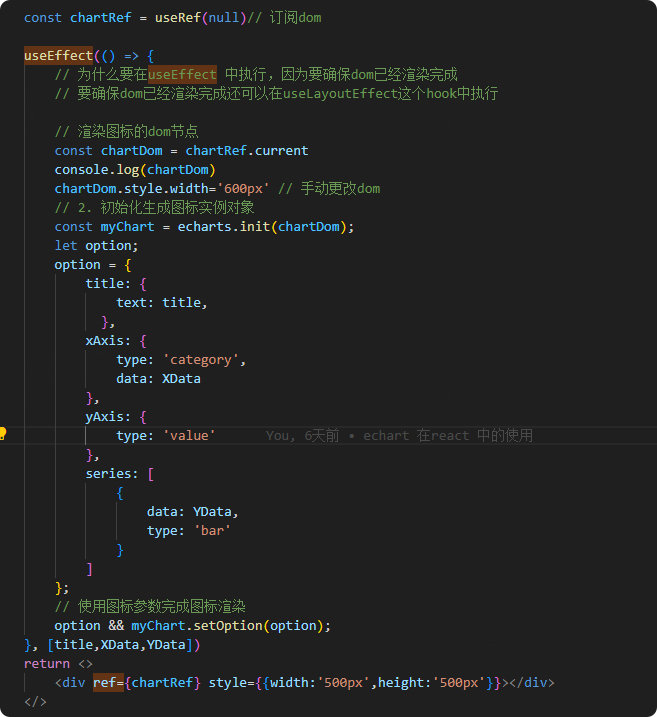
* **挂载阶段：constructor() -> render() -> componentDidMount()**
* **更新阶段：render() -> componentDidUpdate()**
* **卸除阶段：componentWillUnmount()**

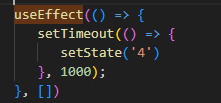
1. **在 React 中，什么是 VirtualDOM?它的作用是什么?  
   定义：Virtual DOM 是一个轻量级的 JavaScript 对象，它描述了实际 DOM 树的结构和属性。   
   作用：Virtual DOM 允许 React 通过比较新旧 Virtual DOM 树的差异来最小化实际 DOM 的变更，提高性能**
2. **请解释-下 React Hooks，并举例说明如何使用 useState 和 useEffect 这两个常用的Hooks**

**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。**

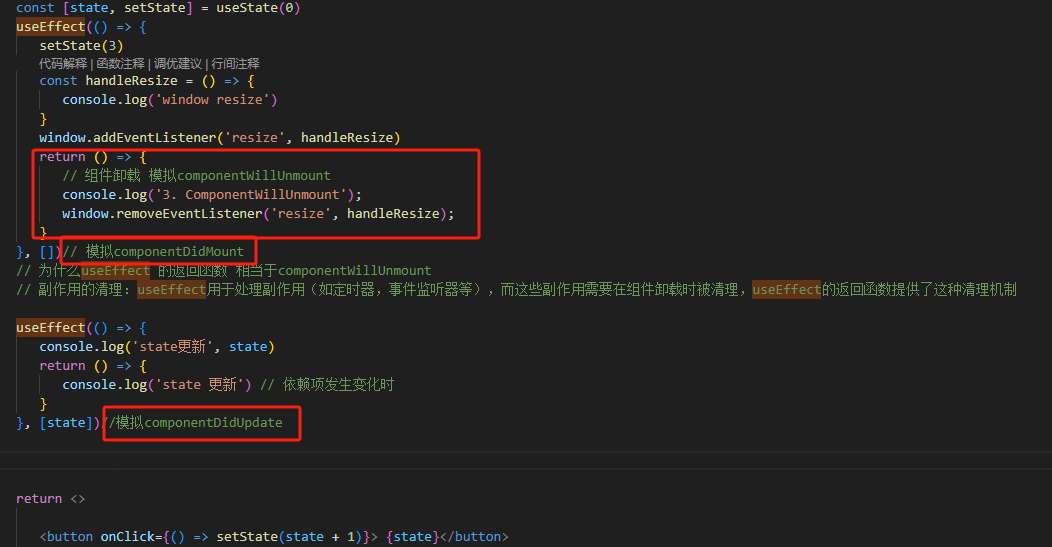
**数据采集：**

**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 中的路由懒载入是什么?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，比如数据采集，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/卸除时，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，说明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** |

1. **React 如何监听  
   使用useEffect 可以监听状态的变化并执行副作用操作**
2. **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  
首屏载入Js档过大：

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**  
   **代码拆分和懒载入  
   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 檔** 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回馈** 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. **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**
2. **Vuerouter 动态路由**  
   **适用场景：  
   1.基于权限的路由管理：不同使用者访问权限不同，使用者登录后，根据其权限动态添加—addRoute  
   2.动态路由参数：需要根据路由参数动态 传值 进行不同的处理  
   3.模块懒加载：动态路由可以和异步组件结合，实现模块懒加载**

|  |  |
| --- | --- |
| **功能** | **示例/说明** |
| **定义动态路由** | **Path: ‘/user/:id’** |
| **获取参数** | **Route.params.id** |
| **动态添加路由** | **Router.addRoute** |

1. **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() 无法监听数组下目标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范本** | **JSX** |
| **回应式机制** | **Proxy/object.defineProperty 实现数据的自动追踪和更新。Vue的回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** |
| **小区和生态系统** | **文文件清晰，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，提供了从范本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)   
回应式机制：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**

1. **vite生产环境 开发环境**

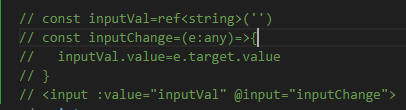
|  |  |  |
| --- | --- | --- |
| **特性** | **开发环境** | **生产环境** |
| **构建工具** | **使用esbuild、快速编译和热重载** | **使用rollup，优化打包和压缩** |
| **环境变量** | **载入 .env.development 文件** | **载入 .env.production 文件** |
| **功能** | **实时预览、热更新、调试友好** | **代码压缩、Tree Shaking、资源优化** |
| **启动命令** | **Vite 或 vite dev** | **Vite build**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



1. **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**

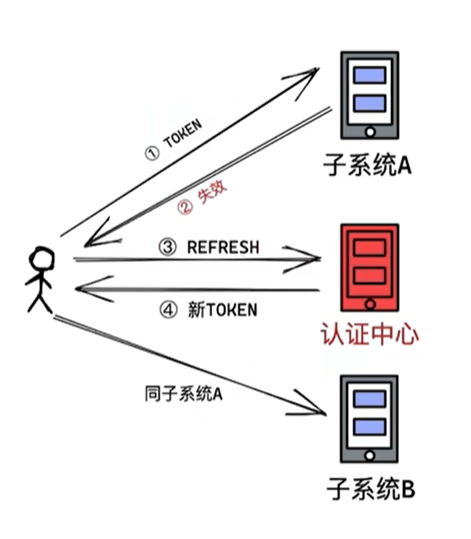
|  |  |  |
| --- | --- | --- |
| **特性** | **Watch** | **watchEffect** |
| **用途** | **监听特定数据源的变化** | **自动收集依赖并执行副作用** |
| **回调执行时机** | **默认不立即执行，但可以通过options 参数中的immediate：true 来实现** | **默认立即执行，类似于watch的immediate：true模式** |
| **依赖收集方式** | **显式指定监听的数据源** | **自动收集依赖** |
| **控制能力** | **提供更多控制选项（如immediate，deep）** | **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）：  
   用户接口部分，负责展示数据和用户应用的交互。通常由html，css 和 范本组成，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是双向资料绑定  
   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  
     
   资料双向绑定：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. **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. **vue权限控制  
   vue 权限控制主要分为以下四部分  
   路由权限：控制使用者可以访问哪些页面  
   方案一：初始化载入全部路由：即在路由定义中添加meta 字段，标记每个路由的权限信息，每次路由跳转时，通过全局路由守卫 beforeEach 进行权限校验  
   方案二：动态加载路由：登录后，根据用户权限动态加载路由，这种方案避免了加载无权限的路由，但需要后端返回用户的权限信息  
   菜单权限：控制用户可以看到哪些菜单项  
   菜单权限通常由后端返回，前端根据用户权限动态生成菜单  
   按钮权限：控制用户可以操作哪些按钮  
   使用v-if指定和自定义指令 实现按钮权限控制**
2. **Proxy Reflect 是什么？在vue3的作用  
   Proxy 是es6 的一个特性，用于创建一个对象的代理，从而实现对对象的拦截，proxy 可以拦截对象的属性读取，设置，删除等操作  
   reflect是es6的一个嵌入式对象，用于拦截和操作js对象的静态方法  
   在vue3中，这两个被用于实现数据的响应式更新，proxy 用于拦截对象的操作，而reflect 用于执行这些操作**
3. **vue3 宏defineProps、defineEmits、withDefaults  
   这三个是<script setup> 语法糖中的编译程序宏   
   defineProps：声明组件的props  
   defineEmits: 触发自定义事件 –通常子传父  
   withDefaults：为props 设置默认值,通常与defineProps 一起用**
4. **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，提高使用者体验。  
   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。**
5. **vue defineAsyncComponent  
   实现组件的异步加载，在需要时才加载某个组件，而不是一次性加载所有数组，减少初始加载时间提升性能**
6. **vue3 ref如何实现响应式原理和reactive有什么不同  
   ref实现原理：主要创建响应式的基本类型资料，也可以用于包装对象或数组，它通过一个包装对象来实现响应式，包装对象里面有value属性，这个属性通过getter和setter 来实现依赖收集和触发更新  
   reactive：创建响应式的对象或数组，对传入的对象进行代理，使其所有属性都具有响应式  
   reactive 使用proxy对象拦截对象的读取和写入操作，通过proxy的get 和set 拦截器，在访问和修改时执行依赖收集和触发更新**

|  |  |  |
| --- | --- | --- |
| **特性** | **Ref** | **Reactive** |
| **使用数据类型** | **基本类型、也可用于对象和数组** | **对象和数组** |
| **访问方式** | **通过.value 访问和修改值** | **直接访问和修改属性** |
| **模块解包** | **在模块中自动解包.value** | **无特殊处理** |
| **内部实现** | **基于proxy，对value 属性进行响应式追踪** | **基于proxy，对整个对象进行递归代理** |
| **性能对比** | **更适合高频词的基本类型操作** | **更适合复杂对象结果** |

1. **setup 语法糖 和 expose/defineExpose  
   setup 用于简化组件的编写方式  
   defineExpose：用于在setup 中显式暴露组件的内部属性和方法  
   父子组件通信：父组件可以通过ref 访问子组件的暴露内容  
   子组件通过defineExpose 暴露方法和属性  
   expose：用在常规的setup 函数中，通过上下文对象调用**
2. **vue getCurrentInstance  
   getCurrentInstance 是一个内部方法，用于获取当前组件的实例，返回了一个对象， 包含组件的上下文信息：proxy，ctx，appContent，props，attrs，alots，emit，refs。  
   适用场景：访问响应式数据：通过getCurrentInstance（）.proxy 可以访问和修改组件的响应式数据、访问生命周期钩子、访问全局属性**
3. **vue组件作用域**

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

}

**通过为组件的样式添加唯一的属性来显示样式的作用域限制，确保组件的样式只应用于当前组件的dom 元素，而不会影响到其他组件的样式，有助于避免样式冲突，提高组件的封装和复用性**

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？  
   vue2:this.$refs.childComponent 可以直接访问子组件的所有data，props，methods，还可以访问子组件的$el(dom元素)，可以调用子组件的方法，监听子组件emit 事件  
   vue3.const childComponent=ref(null)  
   默认情况下，子组件需要使用defineExpose 显示暴露内容，只能访问子组件暴露的属性和方法，可以监听子组件emit 的事件**

|  |  |  |
| --- | --- | --- |
| **特性** | **Vue2** | **Vue** |
| **访问方式** | **This.$refs.xxx** | **ref.value** |
| **默认访问范围** | **全部内容** | **仅暴露内容** |
| **暴露机制** | **自动** | **需要defineExpose** |
| **typescript** | **较差** | **良好** |
| **安全性** | **较低（可访问内部实现）** | **较高（仅能访问暴露内容）** |
| **回应式** | **非回应式** | **回应式** |

1. **css display:none、visibility:hidden 和 opacity:0 之间的区别  
   这三者都是用于隐藏元素的方式**

|  |  |  |  |
| --- | --- | --- | --- |
| **特性** | **Display:none** | **Visibility:hidden** | **Opacity:0** |
| **元素是否占用空间** | **不占用** | **占用** | **占用** |
| **元素是否可交互** | **不可交互** | **不可交互** | **可交互（比如触发点击事件）** |
| **用途** | **完全隐藏并移除元素** | **隐藏但保留空间** | **隐藏但保留空间并且可以交互** |
| **是否影响布局** | **是（重排）** | **否（重绘）** | **否（重绘）** |
| **性能影响** | **高（触发重排）** | **中（触发重绘）** | **低（合成层）** |
| **是否支持过渡动画** | **否** | **是（visibility）** | **是（opacity）** |

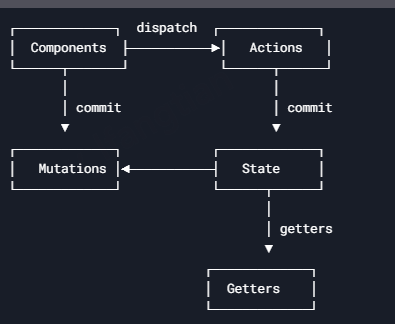
**选择建议：  
需要完全移除元素-使用 display：none  
需要隐藏但保留布局-使用visibility：hidden  
需要动画效果或隐藏但仍需交互-使用opacity：0**

1. **vue3 取消watch  
   1.调用watch 返回的停止函数  
   2.调用watchEffect 返回的 停止函数  
   3.组件卸除时会自动停止监听  
   4.使用显式的清理函数（watchEffect）：接收一个onInvalidate 函数作为参数，可以清理**
2. **vue组件传值  
   vue2:**

|  |  |  |
| --- | --- | --- |
| **方法** | **适用场景** | **特点** |
| **Props 属性传值** | **父传子** |  |
| **This.$emit()** | **子传父** | **通过自定义事件通信，使用$emit 触发事件** |
| **Event bus** | **任意组件间通信** | **全局事件，需注意清理** |
| **Vuex** | **中大型应用状态管理** | **集中式，可追溯** |
| **Ref** | **父组件访问子组件实例** |  |
| **Provide/inject** | **深层嵌套组件** | **依赖注入，响应式需处理** |
| **Attrs/listeners** | **创建高阶组件/透传属性和事件** | **高级用法** |

**$attrs 包含父组件传递但未被props 接收的属性  
$listeners 包含父组件传递的所有事件监听器  
建议：  
优先使用props/emit 实现基础通信  
简单应用可以使用事件总线  
复杂应用建议使用vuex  
避免过度使用ref 直接访问组件实例  
深层嵌套考虑provide/inject  
vue3:**

**父子组件：props和emits自定义事件  
深层嵌套组件：provide和inject  
兄弟组件：共同父组件中转或event bus  
全局状态管理：vuex和pinia**

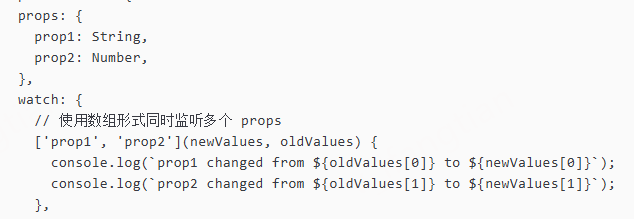
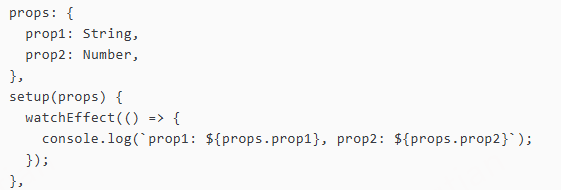
1. **vuex 的各个属性  
   vuex 用于集中式管理应用的所有组件状态  
   state：存储公共数据，只通过mutations 修改  
   getters：从state 派生新数据，类似计算属性，用于复杂的状态计算  
   mutations：同步更改vuex中state的唯一方法  
   actions：异步操作和提交mutations—actions 可以包含任意异步操作，最后通过提交mutations 同步修改state  
   modules：模块化管理store  
   遵循vuex 的单向数据流原则**
2. **vue的生命周期，以及ajax请求在哪个周期发送  
   vue2： beforeCreate-data和methods 还没初始化，无法访问  
   created：data和methods 已经初始化，dom未挂载  
   beforeMount：挂载之前被调用，可以访问el  
   mounted：实例挂载完成之后被调用，可以访问真实的dom元素  
   beforeUpdate：数据更新时，虚拟dom 重新渲染之前被调用，可以访问旧的dom和新的数据  
   updated：数据更新后，虚拟dom重新渲染完成时被调用，可以访问新的dom  
   beforeDestroy：实例销毁前被调用，实例仍然可用，可进行清理操作  
   destroyed：实例销毁后调用，实例所有的绑定和事件监听器已经被移除  
     
   vue3：  
   beforeCreate和created 合并为 setup  
   beforeMount和mount 与vue2 相同  
   beforeUnmount 和unmounted 替代了 vue2 的beforeDestroy和 destroyed  
     
   发送请求的时机  
   1.created ：不依赖dom的初始化数据请求，需要尽早获取数据的情况，优点：比mounted 更早发出请求，适合与组件初始化并行的数据加载  
   2.mounted：需要访问或操作dom的请求。优点：确保dom已经存在，适合基于dom尺寸/位置的请求**
3. **v-for v-if 连用  
   v-for 的优先级高于 v-if，意味着v-if 会在 v-for 之前被解析，因此，v-if的条件会作用于整个循环，而不是循环中的单个元素  
   优化方法：使用计算属性/在外层使用 template 包裹/ 使用method 过滤**
4. **v-show v-if 区别  
   两个都是条件渲染的指令  
   v-if 完全移除元素，减少dom节点，但是切换时需要重新渲染元素，性能开销较大，适用很少改变的场景  
   v-show 切换显示和隐藏的性能开销较小，元素始终存在于dom中，适用频繁切换显示和隐藏的场景 --通过控制元素的display 属性来控制元素的显示和隐藏**
5. **sass和css 使用变量，有哪些异同  
   sass 是一种css 预处理器，扩展css的功能，使得开发更加高效，  
   相同点：变数用于存储可复用的值：可以存储颜色，尺寸，字体等样式值，方便统一管理   
   提高代码可维护性：修改变量值可以全局生效，减少重复代码   
   以下是不同点**

|  |  |  |
| --- | --- | --- |
| **特性** | **Css 变数** | **Sass变数** |
| **语法** | **使用 – 和var（）** | **使用$** |
| **作用域** | **块级作用域** | **全局作用域** |
| **功能** | **支持动态更新和默认值** | **支持嵌套、混合和函数** |
| **编译与运行** | **不需要编译** | **需编译成css** |

**总结：动态主题切换：css 变量（js 可修改）  
复杂样式逻辑：sass变量（配合mixins/函数）  
相容旧浏览器：sass（编译成静态css）  
组件化样式：css变数（作用域更灵活）**

1. **路由传参如何监听  
   vue：  
   使用watch 监听 route.param  
   使用beforeRouteUpdate 导航守卫处理参数变化  
   react:  
   使用useEffect 结合useParams 监听路由参数  
   使用useEffect结合useLocation 监听整个路由变化**

|  |  |  |
| --- | --- | --- |
| **场景** | **Vue方案** | **React方案** |
| **监听动态参数** | **Watch监听 route.params** | **useParams+useEffect** |
| **监听查询参数** | **Watch 监听 route.query** | **useLocation +useEffect** |
| **首次载入触发** | **Mounted** | **useEffect 无依赖项** |

1. **子组件同时监听父组件props的两个值  
   1.使用watch的数组形式：同时监听多个props，并在它们的值发生变化时执行相同的逻辑** **2.使用watchEffect，可以自动监听多个props的变化，无需显式指定依赖**
2. **watch 中的show 和 组件上的show哪个先变化  
   组件上的show 属性先变化，然后再触发watch 回调  
   关键点：vue的响应式更新机制确保props 先更新，再出发watch，如果要确保某些逻辑在watch 之后执行，可以使用nextTick（）--这样是异步代码**
3. **在watch中获取dom   
   dom更新是异步的，所以需要用到nextTick**

|  |  |
| --- | --- |
| **场景** | **解决方案** |
| **监听数据变化后操作dom** | **Watch+ref（获取资料）+nextTick** |
| **任何数据变化后检查dom** | **onUpdated声明周期钩子** |
| **避免直接在watch操作dom** | **用计算属性或指令替代** |

1. **Vue父子组件生命周期顺序  
   vue2 :  
   挂载阶段：(父->子->子完成->父完成)**父beforeCreate 🡪 父created 🡪 父beforeMount 🡪 子beforeCreate 🡪 子created 🡪子beforeMount 🡪子mounted🡪 父mounted  
   **更新阶段：（父更新->子更新->子完成->父完成）**  
   父beforeUpdate 🡪子beforeUpdate🡪 子Updated 🡪父updated  
   **销毁阶段：（父销毁前->子销毁前->子销毁->父销毁）**  
   父beforeDestroy🡪子 beforeDestroy 🡪子 destroyed🡪 父 destroyed  
     
   **vue3 :  
   挂载阶段：**父 setup 🡪 父 onBeforeMount 🡪 子 setup 🡪 子onBeforeMount 🡪子onMounted 🡪 父 onMounted  
   **更新阶段：**  
   父onBeforeUpdate🡪子onBeforeUpdate🡪 子onUpdated🡪父onUpdated  
   **销毁阶段：**  
   父onBeforeUnmount🡪子 onBeforeUnmount🡪子 onUnmouted🡪 父onUnmouted  
    **关键点总结：创建顺序：父组件先创建，子组件后创建  
   挂载顺序：子组件先挂载完成，父组件最后挂在完成  
   更新顺序：父组件先更新，子组件后更新  
   销毁顺序：子组件先销毁，父组件后销毁  
   vue3 变化  
   beforeCreate/created—setup  
   beforeMount—onBeforeMount  
   mounted – onMounted  
   beforeDestroy – onBeforeUnmount  
   destroyed – onUnmounted**
2. **Vue3.0 和 2.0 的响应式原理区别  
   响应式系统的实现方式**

|  |  |  |
| --- | --- | --- |
| **特性** | **Vue2** | **Vue3** |
| **核心实现** | **Object.defineProperty** | **Proxy+reflect** |
| **监听对象属性** | **递归遍历对象，逐个劫持属性** | **直接代理整个对象，动态拦截** |
| **监听数组变化** | **需重写数组方法（如push/pop）** | **原生支持数组操作** |
| **新增/删除属性** | **需用vue.set / vue.delete** | **直接操作即可响应** |
| **Ts支持** | **基于小区支持，类型推断有限，配置复杂** | **原生支持，强大的类型系统，更少的配置** |
| **数据类型** | **不支持Map/Set** | **全面支持** |

**依赖收集与触发更新**

|  |  |  |
| --- | --- | --- |
| **特性** | **Vue2** | **Vue3** |
| **依赖收集** | **通过dep 类和 watcher** | **通过effect + track/trigger** |
| **更新粒度** | **组件级（可能冗余更新）** | **属性级（精准更新）** |

**Vue2 的问题：  
组件级更新：一个数据变化可能导致整个组件重新渲染  
依赖关系复杂：watcher 嵌套可能导致性能问题  
vue3的优化：  
属性级更新：只更新依赖该属性的组件/计算属性  
更高效的依赖追踪：基于weakMap 和set 依赖关系管理  
  
回应式api的差异**

|  |  |  |
| --- | --- | --- |
| **功能** | **Vue2** | **Vue3** |
| **基本响应式** | **Data() 返回对象** | **Ref（）/reactive** |
| **计算属性** | **Computed** | **Computed（）** |
| **监听器** | **Watch** | **Watch/ watchEffect** |
| **回应式工具** | **无** | **toRef/isRef** |

1. **Vue keep-alive 组件  
   vue内置组件，用于缓存动态组件或路由组件，避免重复渲染  
   适用于频繁切换但状态需要保留的组件（如tab切换，spa路由页面）  
   常用props**

|  |  |
| --- | --- |
| **Prop** | **说明** |
| **Include** | **匹配的组件会被缓存（支持字符串。正则、数组）** |
| **Exclude** | **匹配的组件不会被缓存** |
| **Max** | **最大缓存实例数** |

**总结**

|  |  |
| --- | --- |
| **特性** | **说明** |
| **作用** | **缓存组件实例，避免重复渲染** |
| **核心props** | **Include，exclude，max** |
| **生命周期** | **Activated(组件启动，从缓存中恢复) deactivated(组件被停用，进入缓存)** |
| **使用场景** | **动态组件，路由页面，需保留状态的组件** |

1. **nextTick 使用场景和原理  
   是异步队列更新机制，用于在dom更新完成后执行回调函数  
   实现原理：  
   1.nextTick的实现：  
   。vue内部维护一个回调队列，nextTick会把会调函数加入这个队列  
   。通过微任务或宏任务机制，确保回调在dom更新后执行  
   先执行微任务，在执行宏任务  
   2. NextTick 与事件循环  
   。同步代码执行🡪 数据更新（触发watcher）🡪 异步队列更新dom🡪 nextTick 回调执行  
   。nextTick 的会调会在dom更新后、下一个事件循环开始前执行  
     
   为什么nextTick是异步的？  
   性能优化：合并多次数据变更，避免频繁操作dom  
   确保dom更新完成：如果同步执行，可能获取不到最新的dom  
   nextTick和setTimeout（fn,0）的区别  
   nextTick 是微任务（比setTimeout快）  
   什么情况下nextTick 会失效  
   如果数据变更后不处罚响应式更新（如直接修改对象索引或对象属性）nextTick 不会等待dom 更新**

|  |  |
| --- | --- |
| **关键点** | **说明** |
| **作用** | **作用：确保在资料变化后，能获取到最新的dom** |
| **使用场景** | **1.获取更新后的dom 2.在组件渲染完成后执行操作-处理动态内容的尺寸或位置 3.解决v-if/v-show切换后的dom操作问题** |
| **底层机制** | **微任务或宏任务** |

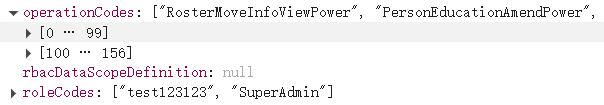
1. **vueRouter 钩子函数  
   全局守卫：会在每个路由导航触发时调用，适用于全局的路由拦截和权限控制：在router/index.js中定义  
   beforeEach：路由进入前触发（常用于登录验证）  
   beforeResolve：路由解析完成后触发（确保异步组件载入完毕）  
   afterEach：路由跳转完成后触发（记录页面访问历史，更改页面标题）  
   路由独享守卫：只针对某个具体的路由生效：在路由配置中定义  
   beforeEnter：在进入某个路由前被调用（常用于对该路由的单独权限验证或数据预加载）  
   组件内守卫：只对该组件的路由跳转生效 ：在 Vue 组件中定义：  
   beforeRouteEnter: 在进入该组件的路由前执行(数据预加载或权限验证)  
   beforeRouteUpdate: 在当前路由改变且该组件被覆用时调用(复用组件时)  
   beforeRouteLeave：离开组件前触发（防止未保存数据丢失）**

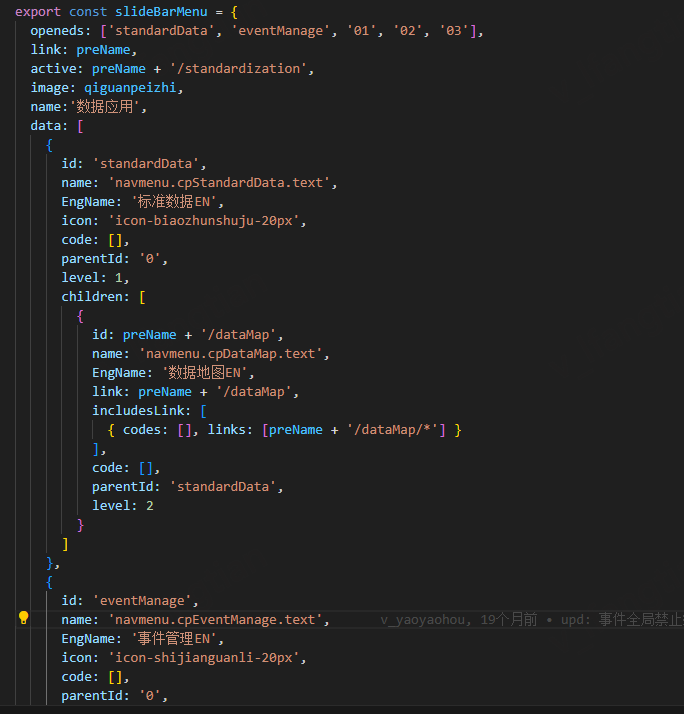
**总结：**

|  |  |  |  |
| --- | --- | --- | --- |
| **守卫类型** | **触发时机** | **作用** | **参数** |
| **beforeEach** | **路由跳转前调用** | **在路由跳转前调用，用于权限验证，登录验证** | **to：即将进入的路由对象 from:当前离开的路由物件 next：控制路由跳转的函数** |
| **beforeResolve** | **在所有组件内守卫和异步路由组件解析后调用** | **确保异步组件和数据加载完成** | **与beforeEach一样** |
| **afterEach** | **导航完成后调用** | **记录页面访问历史，更改页面标题** | **to：已经进入的路由队形 from：已经离开的路由对象** |
| **beforeEnter** | **在进入某个路由前被调用** | **常用于对该路由的单独权限验证或数据预加载** | **与beforeEach一样** |
| **beforeRouteEnter** | **在进入该组件的路由前执行** | **数据预加载或权限验证** | **与beforeEach一样** |
| **beforeRouteUpdate** | **在当前路由改变且该组件被覆用时调用** | **常用于更新组件数据** | **与beforeEach一样** |
| **beforeRouteLeave** | **在离开当前路由时调用** | **常用于保存数据或确认用户是否离开** | **与beforeEach一样** |

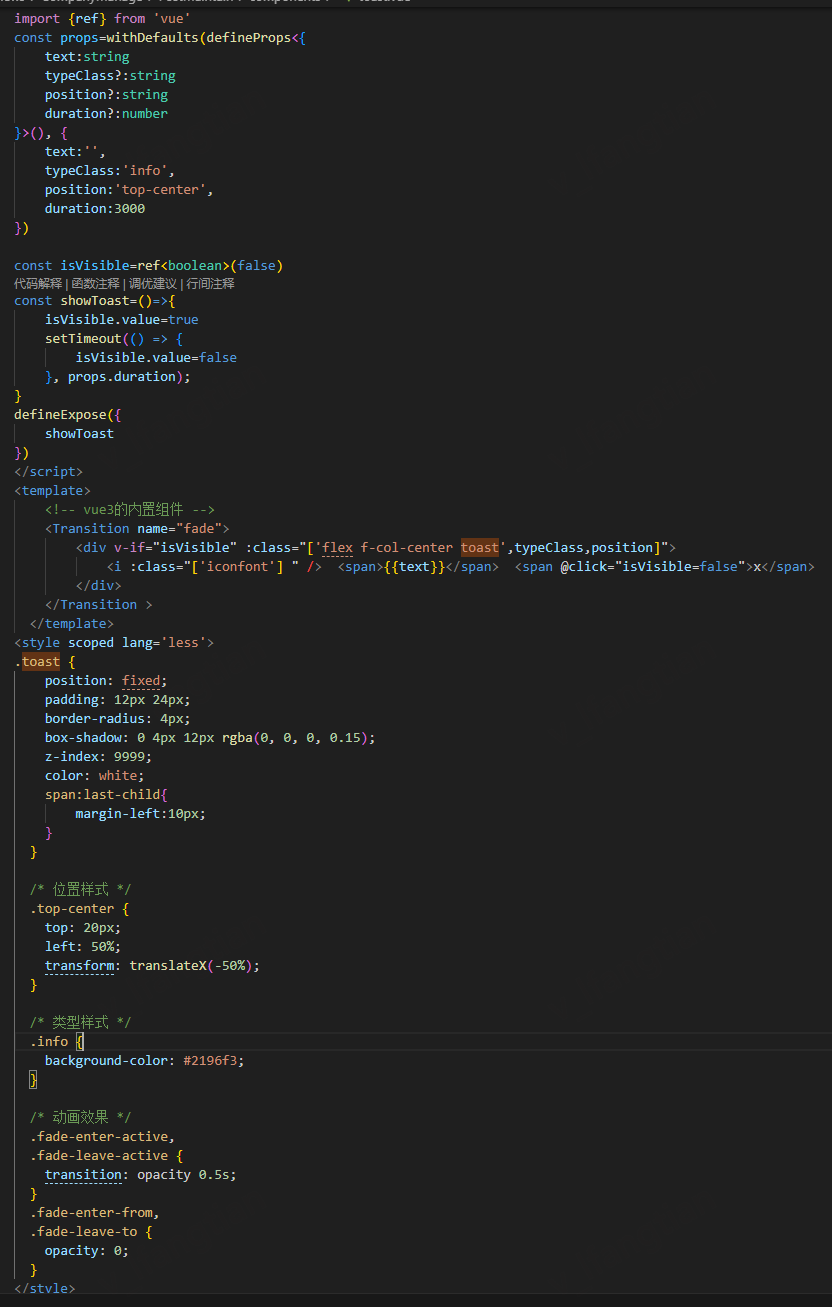
1. **watch watchEffect computed 区别  
   相同点：都是用于响应式处理数据变化的工具  
   不同点：**

|  |  |  |  |
| --- | --- | --- | --- |
| **特性** | **Computed** | **Watch** | **watchEffect** |
| **返回值** | **必须返回一个值** | **无返回值（执行副作用）** | **无返回值（执行副作用）** |
| **执行时机** | **访问时才会计算** | **默认不立即执行，除非设置immediate:true每次变化都会执行** | **立即执行—变化也会执行** |
| **缓存机制** | **有缓存，惰性计算** | **无缓存，每次变化都会触发会调** | **无缓存，每次变化都会触发会调** |
| **适用场景** | **计算派生值（如计算属性）** | **异步操作，复杂逻辑** | **自动追踪多个依赖的副作用** |
| **新旧值** | **无** | **提供** | **不提供** |
| **依赖收集** | **显式依赖** | **显式指定侦听源** | **自动收集回调中的所有依赖** |
| **选择** | **需要基于响应式数据计算出一个新值** | **需要在数据变化时执行异步变化时执行异步操作、复杂逻辑（发送请求，操作dom），并且需要细粒度控制依赖时** | **需要立即执行并自动追踪多个依赖** |

1. **权限的操作；动态路由  
   权限控制指根据使用者的角色或权限级别，控制用户可以访问的路由和功能，一般用户登录后，后端会返回用户的权限信息，前端根据这些信息动态加载路由  
   页面级权限控制：**

 **我们在实际项目中是在router/index 文件里面路由信息完全写死，通过后端返回的用户权限code和前端的路由配置表里面的配置code（就是后端返回的code，根据业务来配置） 一起判断页面权限**

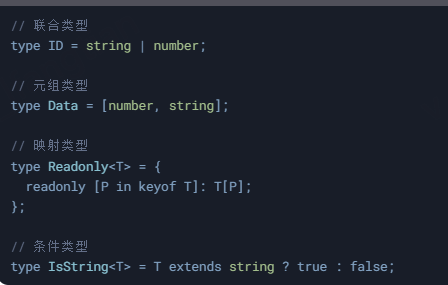
**路由权限：对比看看路由配置表里面有没有，如果没有就是无权限  
其核心思想：当前route.path 递归对比前端的路由配置中是否存在  
如果对比相等--判断后端返回的权限code是否包含了对比相等的路由code，然后返回路由配置表中有权限显示/无配置code（指的是任何用户都可以查看）的路由配置信息，  
如果没有--则跳转无权限页面提示  
如果有，router.replace跳转指定的页面  
如果不相等—返回404页面**

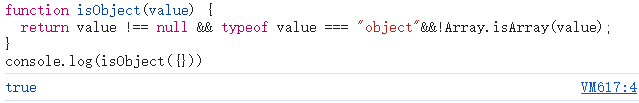
1. **toast 封装**
2. **判断js数据类型哪些方法**
3. **防抖节流：手撕**
4. **读代码输出题：关于this指向的，还有settimeout，async等同步任务、微任务执行顺序之类的**
5. **同源策略以及跨域解决方案  
   同源策略：浏览器的安全机制，限制了一个源的文档或脚本如何与另一个源的资源进行交互  
   同源定义：协议http/https、域名example.com、端口80/443等都相同==同源  
   跨域解决方案：  
   1.后端设置cors（跨域资源共享）  
   2.前端设置代理服务器-让同源服务器转发请求（我们项目就是这样）  
   3.jsonp(仅限get请求)-<script>不受同源策略限制**
6. **CSRF攻击说一下，还有什么其他的攻击？  
   CSRF是一种网络攻击，攻击者通过伪装成用户的身份，向目标网站发送伪造的请求，从而执行用户未授权的操作，比如:攻击者可以利用csrf 转移资金  
   防范措施：服务端使用CSRF令牌验证请求的来源， 客户端防御：敏感操作使用post而不是get  
   其他攻击： XSS：向网页注入恶意脚本，在用户浏览器执行**
7. **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**
8. **对低代码的理解**
9. **100张图片需要加载，怎么优化？（图片懒加载）  
   1.使用interSectionObserver api 手动实现懒加载  
   这个api是检测元素是否进入或离开可视区域  
   2.使用第三方库vue-lazyload  
   是一个vue插件，专门用于实现图片懒加载  
   3.图片压缩、响应式图片—picture元素和srcset属性 来进行优化**
10. **怎么获取图片是不是在可视区域内？有哪些方法？  
    1、getBoundingClientRect：提供元素的大小和相对于视口的位置  
    2、IntersectionObserver 检测元素是否进入或离开可视区域**
11. **创建一个对象的原理？   
    1.使用对象字面量直接创建一个新对象  
    2.使用new object 创建一个对象  
    3.使用构造函数创建对象  
    4.使用object.create 创建对象**
12. **将对象的值收集到数组中  
    1.使用object.values  
    2.for..in  
    3.object.keys + map  
    4.object.keys + reduce  
    5.array.from+object.values**
13. **事件代理和proxy，为什么要有事件代理  
    事件代理：利用事件冒泡机制，将事件监听器绑定到父元素上，而不是直接绑定到每个子元素上，作用：  
    减少内存占用：通过在父元素绑定事件监听器，而不是绑定到子元素上，显著减少内存消耗  
    提高性能：减少事件监听器的数量可降低页面加载  
    简化代码  
    动态元素处理：对于动态添加/删除的子元素，事件代理可以自动处理事件，无需重新绑定监听器   
    proxy：对象代理机制，用于拦截和自定义对象的基本操作**
14. **token/Cookie/SessionStorage/LocalStorage**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **特性** | **Token** | **Cookie** | **sessionStorage** | **localStorage** |
| **存储位置** | **客户端** | **客户端和服务端** | **客户端** | **客户端** |
| **存储大小** | **无限制** | **通常4kb** | **5MB** | **5MB** |
| **自动发送** | **手动设置** | **自动发送到服务器** | **不会自动发送** | **不会自动发送** |
| **安全性** | **高（可加密）** | **中（可通过httpOnly和secure增强）** | **低** | **低** |
| **适用场景** | **身份验证 ，适用于Api请求** | **会话跟踪** | **会话级数据存储** | **持久数据存储** |

1. **ts 常用特性  
   1.类型注解：为变量，函数参数，返回值添加类型信息 “:”  
   2.接口（interface）：定义对象的结构  
   3.枚举：定义一组命名的常量 enum  
   4.泛型：允许定义可复用的组件，保持类型安全 “T“  
   5.类型断言：手动指定一个值的类型，当ts的类型推断无法正确推断类型时适用 “as“  
   7.联合类型：允许一个变量可以是多种类型之一 “|“  
   8.类型别名：type  
   9.模块（module）：用于组织和分割代码，支持导入和导出功能  
   10.交叉类型：”&“**
2. **Ts中定义类型的方式  
   1.基本类型：“：“  
   2.联合类型：”|“  
   3.交叉类型：“&  
   4.类型别名：type  
   5.接口：interface  
   6.数组类型：number[]/Array<number>  
   7.元组类型：[string,number]  
   8.函数类型：定义函数的参数类型和返回值类型  
   9.对象类型：通过接口和类型别名定义  
   10.枚举类型  
   11.类型断言**
3. **Ts中type 和 interface 的区别**

|  |  |  |
| --- | --- | --- |
| **特性** | **Interface** | **Type** |
| **扩展方式** | **Extends** | **&** |
| **声明合并** | **支持** | **不支持** |
| **类实现** | **支持** | **支持** |
| **复杂类型** | **有限** | **强大** |
| **性能** | **优** | **良** |
| **适合场景** | **对象形状** | **复杂类型** |



1. **手写call、apply、bind函数  
   https://blog.csdn.net/qq\_44062823/article/details/145737025?spm=1001.2014.3001.5501**
2. **从数组中查找特定元素  
   1.array.prototype.find—返回满足条件的第一个元素  
   2. array.prototype.findIndex—返回满足条件的第一个元素的索引  
   3. array.prototype.filter—返回满足条件的所有元素组成的数组  
   4. For 循环+判断  
   5. array.prototype.includes ---布尔值 判断数组是否包含某个值  
   6. array.prototype.some ---布尔值 是否有元素符合**
3. **判断对象是否为空  
   1.json.stringify() === ‘{}’  
   2.Object.keys(obj).length===0  
   3.Object.values(obj).length===0   
   4. Object.getOwnPropertyNames(obj).length===0  
   5.object.entries(obj).length===0**
4. **Js拷贝  
   浅拷贝：复制对象、数组的第一层，不会递归复制内部的嵌套对象、数组  
   方法：Object.assign、使用扩展运算符、数组使用slice/扩展运算符  
   深拷贝：深拷贝会递归复制对象或数组的所有层级，包括嵌套的对象和数组  
   方法： json.parse(json.stringify()) 不适用函数、undefined、date特殊类型的对象 、 使用lodash的cloneDeep  
   递归函数实现深拷贝：  
   https://blog.csdn.net/qq\_44062823/article/details/147096221?spm=1001.2014.3001.5501**
5. **.数组扁平化  
   1.array.prototype.flat(Infinity) 展开所有层级,接收表示深度的参数，用于指定需要展平的层级  
   2.递归数组  
   3.reduce+递归  
   4.json.stringify()和json.parse()  
   5.Array.prototype.flatMap():只能展开一层，需要多次调用**
6. **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);}
7. **isObject/isObjectLike/isPlainObject  
   isObject:判断一个值是否是一个对象类型** **isObjectLike：判断一个值是否“类似对象”，即是否可以像对象一样使用属性**
8. **https/验证证书**
9. **vue-loader：  
   webpack 加载器，解析.vue 档、支援预处理器。热重载，代码分割和懒加载，资源加载，但我们项目用的是vite，使用plugin-vue 来处理这些**
10. **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檔，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档

在这个过程中，webpack 默认只能处理JavaScript档，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 | 小档转换为Base64，大档退化为file-loader |
| 适用场景 | 大档或需要单独存储的资源 | 小档，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 | 避免嵌入大档、减少打包体积 | 减少http请求，适合小档 |

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，适用于小档，他会将档内容转换为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

1. **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** | **减少网络等待时间，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是一个模块打包工具，项目中的资源（css，js，图片）打包成浏览器可以识别的格式  
   webpack主要包括以下功能：  
   模块打包：项目的所有模块打包成一个或多个文件  
   代码分割：支持代码分割，按需加载  
   热模块替换：开发过程中，支持热模块替换，无需重新加载整个页面  
   插件系统：通过loader和plugin系统，可以高度定制化构建过程  
   优化：提供代码压缩、缓存等功能  
     
   vite：现代前端开发构建工具  
   vite主要包括以下功能：  
   快速启动：基于原生ESM和依赖预构建，开发服务器几乎瞬间启动  
   热模块替换：同上  
   依赖预构建：自动预构建项目依赖，减少开发过程中的构建时间  
   插件系统：支持插件扩展，可以自定义构建过程  
   内置开发服务器：内置了一个功能强大的开发服务器，支持代理，静态文件服务等功能**  
   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

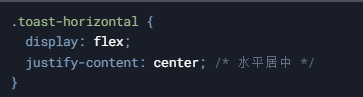
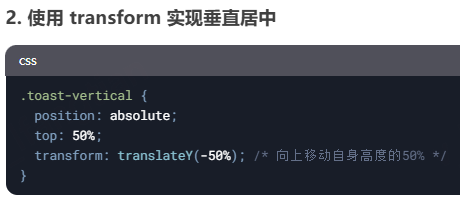
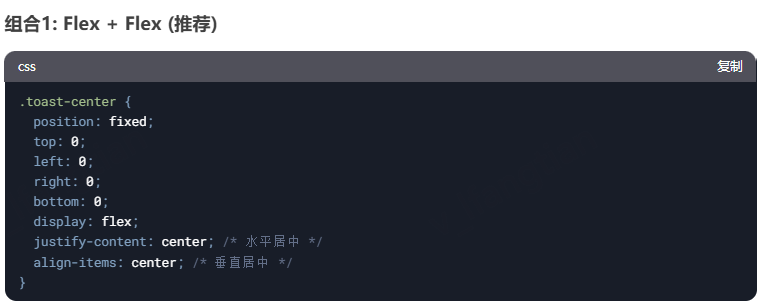
|  |  |  |
| --- | --- | --- |
| 特性 | Vite | Webpack |
| 构建速度 | 极快 | 较慢，全量打包 |
| 打包方式 | 开发环境不打包，生产环境用Rollup | 全量打包 |
| 热更新 | 快速，基于原生ESM | 较慢，受项目规模影响 |
| 配置复杂度 | 简单，开箱即用 | 复杂，需手动配置 |
| 生态系统 | 较新，正在发展中 | 成熟，插件丰富 |
| 适用场景 | 开发环境、中小型项目 | 大型项目、复杂场景 |
| 浏览器兼容性 | 现代浏览器 | 现代和旧版浏览器 |

1. **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（用户接口）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. **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 关键词用于声明一个函数是异步的，该函数会返回一个promise  
   await 关键词用于等待一个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 关键词等待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
4. **箭头函数和普通函数有什么区别  
   类数组对象：具有索引和length属性的对象，但不具备数组的方法，如push、pop、forEach**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **语法** | **This绑定** | **Arguments** | **是否可作构造函数** |
| **箭头函数** | **（）=>** | **不能绑定自己的this，而是继承其所在上下文的this** | **不能绑定自己arguments，但可以通过…操作符访问参数** | **不可以** |
| **普通函数** | **Function关键字** | **取决于函数的调用方式 对象的方法调用：this指向调用它的对象 普通函数调用：this指向全局对象** | **可访问：他是个类数组对象，包含函数调用时的所有参数** | **可以，通过new 关键字创建对象** |

1. **ssr协议和实现逻辑**
2. **vite 分包基于HTTP长连接**
3. **动态执行 js**  
   https://kimi.moonshot.cn/share/ct1f4p21bmk5dtj7m9lg
4. **防抖/节流**

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

1. **项目中做了哪些优化**
2. **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）**
3. **水平垂直居中，分别都说出两种  
   水平居中：** **垂直居中：** **水平垂直居中：**
4. **block和inline的区别**

|  |  |  |
| --- | --- | --- |
| **特性** | **Block（块级元素）** | **Inline（行内元素）** |
| **排列方式** | **独占一行，垂直排列** | **不换行，水平排列** |
| **尺寸设置** | **可以设置宽高** | **不可设置，宽高由内容决定** |
| **内外边距** | **四个方向都有效** | **水平方向有效** |
| **包含关系** | **可以包含任何元素** | **通常只能包含文本和其他行内元素** |

**Inline-block：是两者的混合体  
像块级元素一样设置宽高  
像行内元素水平排列**

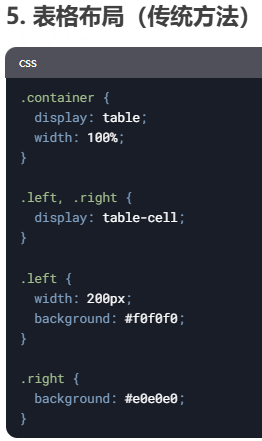
1. **选择器的优先级  
   内联样式：如 style=“”  
   ID选择器：如 #header  
   类/属性/伪类 选择器： 如.class,[type=’text’],:hover  
   元素/伪元素选择器：如 div，：：before  
   !important 会覆盖其他所有的声明**
2. **BFC讲一下：创建条件，作用  
   BFC:是一个独立渲染区域，他决定了内部元素的布局方式，并且内部元素的布局不会影响到外部元素  
   创建条件：  
   根元素（html）  
   设置position:absolute/fixed  
   设置overflow:hidden/auto/scroll  
   设置display：inline-block/flex/grid  
   作用：  
   清除浮动、防止外边距重叠、隔离布局、防止浮动元素重叠、计算高度时包含浮动元素  
   应用场景：  
   清除浮动  
   防止外边距重叠  
   自适应布局  
   隔离元素  
   BFC创建方式的优先选择级：  
   首选：display：flow-root  
   其次：overflow:hidden**
3. **rem em   
   都是css中相对长度单位，主要用于实现响应式和可伸缩的界面设计**

|  |  |  |
| --- | --- | --- |
| **特性** | **Em** | **Rem（root em）** |
| **基准值** | **相对于当前元素的字体大小** | **相对于根元素的字体大小** |
| **继承影响** | **受父元素字体大小影响** | **不受继承影响** |
| **计算方式** | **可以累计计算** | **始终基于根元素固定计算** |
| **使用场景** | **组件内部相对尺寸** | **全局一致的相对尺寸** |
| **稳定性** | **嵌套结构可能不稳定** | **更稳定和可预测** |

1. **position属性（如absolute、fixed或relative）  
   position 是css 中控制元素定位方式的核心属性，主要包括：  
   static、relative、absolute、fixed、sticky**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **属性值** | **定位方式** | **脱离文档流** | **滚动影响** | **场景** |
| **Static** | **正常文档流** | **否** | **受影响** | **默认值** |
| **Relative** | **自身原始位置** | **否** | **受影响** | **微调位置** |
| **Absolute** | **最近非static祖先** | **是** | **受影响** | **精确控制位置** |
| **Fixed** | **视口** | **是** | **不受影响** | **固定位置的导航栏** |
| **Sticky** | **最近滚动祖先** | **否-滚动后固定** | **部分影响** | **粘性定位** |

**父元素高度塌陷  
问题：父元素不包含绝对定位子元素的高度  
解决：给父元素设置position：relative 或明确高度  
sticky定位失效  
原因：父元素由overflow：hidden  
解决：确保父元素可滚动并设置top/bottom值**

1. **CSS左边固定右边自我调整** **首选：flex/grid**