Vue常见面试题

1. Vue基础

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式 JavaScript框架。

- 全称是Vue.js或者Vuejs；
- 它基于标准 HTML、CSS 和 JavaScript 构建，并提供了一套声明式的、组件化的编程模型；
- 帮助你高效地开发用户界面，无论任务是简单还是复杂；

1.1 Vue.js 的特点

- 易用： Vuejs是一个渐进式的框架，相比于其它框架，它更简单，易学，上手快。
- 灵活： （渐进式）不断繁荣的生态系统，可以在一个库和一套完整框架之间自如伸缩。
- 高效： 20kB min+gzip 运行大小；超快虚拟 DOM；最省心的优化。
- 双向绑定：开发效率高。
- 基于组件的代码共享
- Web项目工程化，增加可读性、可维护性

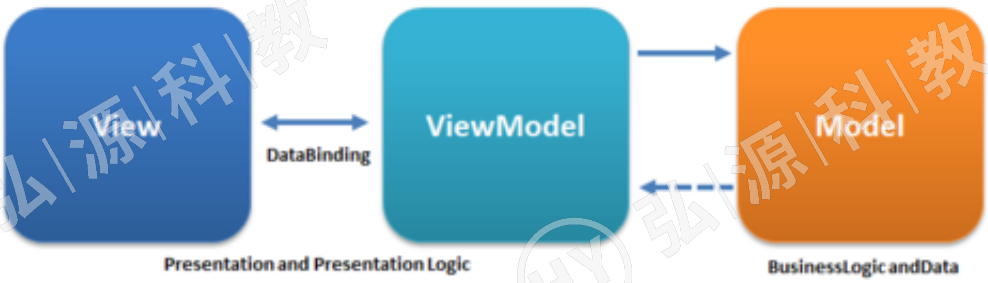
1.2 什么是 MVVM?

MVC和MVVM都是一种软件的体系结构

- MVC是Model – View –Controller的简称，是在前期被使用非常框架的架构模式，比如iOS、前端；
- MVVM是Model-View-ViewModel的简称，是目前非常流行的架构模式；

通常情况下，我们也经常称Vue是一个MVVM的框架。

- Vue官方其实有说明，Vue虽然并没有完全遵守MVVM的模型，但是整个设计是受到它的启发的。



1.3 说说你对 SPA 单页面的理解，它的优缺点分别是什么?

SPA（ single-page application ）仅在 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转；取而代之的是利用路由机制实现 HTML 内容的变换，UI 与用户的交互，避免页面的重新加载。

优点：

- 用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染；
- 基于上面一点，SPA 相对对服务器压力小；
- 前后端职责分离，架构清晰，前端进行交互逻辑，后端负责数据处理；

缺点：

- 初次加载耗时多：为实现单页 Web 应用功能及显示效果，需要在加载页面的时候将 JavaScript、CSS 统一加载，部分页面按需加载；
- 前进后退路由管理：由于单页应用在一个页面中显示所有的内容，所以不能使用浏览器的前进后退功能，所有的页面切换需要自己建立堆栈管理；
- SEO 难度较大：由于所有的内容都在一个页面中动态替换显示，所以在 SEO 上其有着天然的弱势。

1.3 v-show 与 v-if 有什么区别?

首先，在用法上的区别：

- v-show是不支持template；
- v-show不可以和v-else一起使用；

其次，本质的区别：

- v-show元素无论是否需要显示到浏览器上，它的DOM实际都是有存在的，只是通过CSS的display属性来进行切换；
- v-if当条件为false时，其对应的原生压根不会被渲染到DOM中；

开发中如何进行选择呢?

- 如果我们的原生需要在显示和隐藏之间频繁的切换，那么使用v-show；
- □ 如果不会频繁的发生切换，那么使用v-if；

1.4 数组中的哪些方法会触发视图的更新?

Vue 将被侦听的数组的变更方法进行了包裹，所以它们也将会触发视图更新，这些被包裹过的方法包括：

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

上面的方法会直接修改原来的数组，所以它们会触发视图更新。

其它数组的方法:

- 但是某些方法不会替换原来的数组，而是会生成新的数组，比如 filter()、concat() 和 slice()，使用这些方法将不会触发视图更新。

1.5 Vue中v-for的key 有什么作用?

在使用v-for进行列表渲染时，我们通常会给元素或者组件绑定一个key属性。

这个key属性有什么作用呢?

- key属性主要用在Vue的虚拟DOM算法，在新旧nodes对比时辨识VNodes。
- 如果不使用key，Vue会使用一种最大限度减少动态元素并且尽可能的尝试就地修改/复用相同类型元素的算法
- 使用key时，它会基于key的变化重新排列元素顺序，并且会移除/销毁key不存在的元素。

key 是 VNode 的唯一标记，通过这个 key， diff 操作可以更准确、更快速的达到复用节点，更新视图的目的。复用节点就需要通过移动元素的位置来达到更新的目的。

1.6 computed和method有什么区别?

- 计算属性和方法：
  - 都可以通过this来访问
  - 都可以对一些数据进行处理和计算
  - 对于包含响应式数据计算的逻辑，应该使用计算属性，因为计算属性是有缓存。
- computed和method的区别
  - computed底层会缓存, 性能更高
  - 计算属性会基于它们的依赖关系进行缓存;
  - 在数据不发生变化时，计算属性是不需要重新计算的
  - 但是如果依赖的数据发生变化，在使用时，计算属性依然会重新进行计算

1.7 什么是双向绑定？v-model的本质是什么?

双向绑定:

- 即当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化
- v-model 是语法糖，它负责监听用户在表单元素中的输入事件来更新数据

表单元素使用v-model的本质：

- v-bind绑定value属性的值
- v-on绑定input事件监听到函数,函数会获取最新的值赋值到绑定的属性中

```
<input type="text" :value="message" @input="message = $event.target.value" />
```

组件使用v-model的本质：

- 将其 value attribute 绑定到一个名叫 modelValue 的 prop 上；
- 在其 input 事件被触发时，将新的值通过自定义的 update:modelValue 事件抛出(发出)；

```
<Counter v-model="appCounter"/>
<!-- 相当于-->
<Counter v-bind:modelValue="appCounter" @update:modelValue="appCounter =
$event"/>
```

1.8 data选项为什么是一个函数而不是对象?

JavaScript中的对象是引用类型的数据，当多个实例引用同一个对象时，只要一个实例对这个对象进行操作，其他实例中的数据也会发生变化。

而在Vue中，我们更多的是想要复用组件，那就需要每个组件都有自己的数据，这样组件之间才不会相互干扰。

所以组件的数据不能写成对象的形式，而是要写成函数的形式。数据以函数返回值的形式定义。

这样当我们每次复用组件的时候，就会返回一个新的data，也就是说每个组件都有自己的私有数据空间，它们各自维护自己的数据，不会干扰其他组件的正常运行。

1.9 Vue data 中某一个属性的值发生改变后，视图会立即同步执行重新渲染吗?

不会立即同步执行重新渲染。

Vue 实现响应式并不是数据发生变化之后 DOM 立即变化，而是按一定的策略进行 DOM 的更新。

Vue 在更新 DOM 时是异步执行的。只要侦听到数据变化， Vue 将开启一个队列，并缓冲在同一事件循环中发生的所有数据变更。

如果同一个watcher被多次触发，只会被推入到队列中一次。这种在缓冲时去除重复数据对于避免不必要的计算和 DOM 操作是非常重要的。

然后，在下一个的事件循环“tick”中，Vue 刷新队列并执行实际（已去重的）工作。

1.10 sass是什么？如何在vue中安装和使用?

sass是一种CSS预处理器语言，除此之外，less、stylus也是常见的CSS预处理器语言。

sass安装和使用步骤如下：

1. 用npm安装加载程序（ sass-loader、 css-loader等加载程序)。
2. 在 webpack.config.js中配置sass加载程序。

1.11 在 Vue. js开发环境下调用API接口，如何避免跨域

1.在vue.config.js中的devServer选项中的proxy中配置反向代理

2.在vite.config.js中的server选项中的proxy中配置反向代理

3.直接后端开发人员配置cors

1.12 v-if和v-for一起使用的弊端及解决办法

Vue.js 中使用最多的两个指令就是 `v-if` 和 `v-for`，因此开发者们可能会想要同时使用它们。虽然不建议这样做，但有时确实是必须的，于是我们想提供有关其工作方式的指南。

- 2..x 版本中在一个元素上同时使用 `v-if` 和 `v-for` 时，`v-for` 会优先作用。
- 3.x 版本中 `v-if` 总是优先于 `v-for` 生效。

由于语法上存在歧义，建议避免在同一元素上同时使用两者。

比起在模板层面管理相关逻辑，更好的办法是通过创建计算属性筛选出列表，并以此创建可见元素，比如：

1. 在v-for的外层或内层包裹一个元素（template）来使用v-if
2. 用computed处理筛选出列表

1.13 谈谈你对 keep-alive 的了解?

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。

1.14 说说Vue插槽的作用和平时开发中的应用?

插槽的作用:

- 支持在父组件自定义子组件中的个内容
- 让子组件更具有通用性，不必限定死某个内容

插槽平时开发中的应用:

- 在封装组件时，如果组件中的某个内容是动态的或不确定的，就可以使用插槽来代替了。
- 在使用第三方库时，往往会通过使用插槽类自定义第三方组件中的某些内容。

2. Component组件

2.1 父子组件的生命周期顺序

- 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

2.2 组件通讯(传值)式有哪些?

- 父传子：子组件通过props来接收父组件传递的属性 xxx 的值
- 子传父：子组件通过emit触发事件传递，父组件通过监听对应的事件来接收数据
- Provide/Inject：父组件提供内容，子或孙组件可以注入父组件提供的内容。
- 组件实例：通过ref来拿到组件的实例，调用实例的属性或方法进行传值。
- 事件总线：可以自己编写EventBus插件来进行通讯，或世界使用第三方的事件总线库。
- 用Vuex/Pinia: 可以使用全局状态管理来进行全局共享数据。

2.3 什么是生命周期函数？Vue组件的生命周期函数有哪些?

生命周期函数：

- 生命周期函数是一些钩子函数（回调函数），在某个时间会被Vue源码内部进行回调
- 通过对生命周期函数的回调，我们可以知道目前组件正在经历什么阶段

Vue2的生命周期函数:

- beforeCreate :组件实例在创建之前
- created: 组件被创建完成
  - 可以发送网络请求
  - 可以事件监听
  - this.$watch()

- beforeMount : 组件template准备被挂载
- mounted :组件template已经被挂载
  - 可以获取DOM,可以使用DOM
- beforeUpdate: 准备更新DOM
- updated: 更新DOM,根据最新数据生成新的VNode,生成新的虚拟DOM,转换为真实的DOM

- beforeUnmount: 卸载之前
- unmounted: DOM 元素被卸载完成
  - 回收操作(取消事件监听)

Vue3的生命周期函数:

3. Composition API

3.1 什么是Composition API 和 Options API?

Composition API:

- Composition API 是一组 API，允许我们使用导入的函数而不是声明选项来编写 Vue 组件。它是一个涵盖以下 API 的总称：Reactivity API、Lifecycle Hooks、Dependency Injection等等
- 使用Composition API编写组件时可以根据逻辑功能来组织代码。比如可以把一个功能所用到的 API 放在一起，这样可以让代码高内聚和低耦合，进而提高了代码的逻辑的复用性。
- 在 Vue 3 中，它也主要与script setup语法一起使用。

Options API:

- 在对应的属性中编写对应的功能模块, 比如data定义数据、methods中定义方法、computed中定义计算属性、watch中监听属性改变，也包括生命周期钩子
- 弊端: 当我们实现某一个功能时，这个功能对应的代码逻辑会被拆分到各个属性中,当组件变得复杂，导致对应属性的列表也会增长，这可能会导致组件难以阅读和理解

3.2 Composition API和之Options API有什么区别?

- 在逻辑组织和逻辑复用方面，Composition API是优于Options API。
- Composition API几乎是函数，会有更好的类型推断，对于TS的支持更友好。
- Composition API对 tree-shaking 友好，代码也更容易压缩。
- Composition API中见不到this的使用，减少了this指向不明的情况。
- Composition API用起来稍微复杂一点，而Options API就非常简单、易于使用。

3.3 说说Vue3中setup函数的作用?

在Vue3中，`setup()`函数充当了组件编写Composition API 的入口点。

- setup函数参数主要有两个参数：
  - 第一个参数：props , 父组件传递过来的属性会被放到props对象中
  - 第二个参数：context, 它里面包含三个属性
    - attrs：所有的非prop的attribute；
    - slots：父组件传递过来的插槽；
    - emit：当我们组件内部需要发出事件时会用到emit（因为我们不能访问this，所以不可以通过 this.$emit发出事件）
- 可以在setup中可以定义响应式数据、方法、计算属性、侦听器等等。
- 可以通过setup的返回值来替代data选项，让数据可以直接在template中使用。

3.4 ref和reactive有什么区别？开发中如何选择?

- ref和reactive都是响应式的API，都可以用来定义响应式的数据。
- ref可以包裹任意数据类型，reactive只能包裹复杂数据类型，比如对象、数组。

- ref返回一个ref对象，在script中取值需要通过value属性，但是在模板中使用会进行解包不需要调用value。
- reactive包裹的是复杂数据类型，直接取里面的属性即可。

- ref几乎可以应用在任何场景，而且包含reactive适合的场景
- reactive的应用场景比较受限，第一：值比较固定，第二：值与值之间是有联系的。
- 开发中尽量选择ref

3.5 Composition API常见的几个函数与用法?

- ref
  - 包裹任意类型的值，将包裹的值加入响应式
- reactive
  - 包裹复杂类型的值，将包裹的值加入响应式
- computed
  - 把一些复杂逻辑用computed进行包裹，如同Options API中的计算属性一样
  - computed会自动收集相关依赖，当依赖发生变化时，会自动进行更新
- 生命周期
  - Vue3中想要在beforeCreate和created中做的事，直接在setup中做即可
  - Vue3的其他的生命周期函数都要在前面加一个on，然后需要在vue中主动引入
- watch
  - watch可以监听单个数据源，也可以监听多个数据源
  - watch是懒执行，第一次是不会执行的，除非你为其提供第三个参数中的immediate属性为true
  - watch只有等到监听的数据源发生了变化后，才会执行第二个参数（回调）
  - watch可以获取监听数据源的前后变化的值
  - 侦听多个数据源的时候，第一个参数是数组类型
- watchEffect
  - watchEffect会自动收集依赖，收集的依赖是第一个参数，也就是回调函数中有哪些东西是加入响应式的
  - 如果这个值加入了响应式就会被收集起来，当被收集的值发生了变化，就会重新执行这个回调函数
  - watchEffect第一次执行是在DOM挂载前执行的，所以如果你想在第一次执行时拿到DOM元素
  - 需要传入第二个参数，第二个参数是一个对象，让其flush属性的值为post即可
- toRefs

    - 对reactive进行解构后就失去了响应式的效果，因为reactive返回的是一个Proxy对象
    - 对Proxy对象进行解构，拿到的是纯净的值，所以没有了响应式的效果
    - 如果想要对reactive进行解构，需要对其包裹一个toRefs
    - 这么做相当于为reactive中的每一个值包裹了一个ref
- 等等

3.6 Vue3中的watch和watchEffect有什么区别?

- watch和watchEffect都用用来侦听响应式数据的变化，watch可以侦听指定的源，默认第一次不会执行，watchEffect虽不能指定侦听的源，但是会自动收集依赖，并默认会先执行一次。
- watch
    - watch可以监听单个数据源，也可以监听多个数据源
    - watch是懒执行，第一次是不会执行的，除非你为其提供第三个参数中的immediate属性为true
    - watch只有等到监听的数据源发生了变化后，才会执行第二个参数（回调）
    - watch可以获取监听数据源的前后变化的值
    - 侦听多个数据源的时候，第一个参数是数组类型
- watchEffect
    - watchEffect会自动收集依赖，收集的依赖是第一个参数，也就是回调函数中有哪些东西是加入响应式的
    - 如果这个值加入了响应式就会被收集起来，当被收集的值发生了变化，就会重新执行这个回调函数
    - watchEffect第一次执行是在DOM挂载前执行的，所以如果你想在第一次执行时拿到DOM元素
    - 需要传入第二个参数，第二个参数是一个对象，让其flush属性的值为post即可

3.7 说说Vue3中script setup语法糖常见用法?

script setup 是在单文件组件中使用 Composition API 的编译时语法糖，相比与之前的setup函数写法，它具有更多的优势：

- 更少的样板内容，更简洁的代码。
- 能够使用纯 TypeScript 声明 props 和抛出事件。
- 更好的运行时性能 (其模板会被编译成与其同一作用域的渲染函数，没有任何的中间代理)。
- 更好的 IDE 类型推断性能 (减少语言服务器从代码中抽离类型的工作)。

1.script setup

- 当使用 script setup 的时候，任何在 script setup 声明的顶层绑定都能在模板中直接使用
    - 声明的顶层绑定：包括变量，函数声明，以及 import 引入的内容
- 响应式数据需要通过ref、reactive来创建
- 在script setup中导入的组件可以直接使用

2.defineProps

- 在script setup语法糖中必须使用 defineProps API来声明props，它具备完整的类型推断并且在 `<script setup>` 中是直接可用的（不需要额外导入）。

3.defineEmits

- 在script setup语法糖中必须使用 defineEmits API来声明 emits，它具备完整的类型推断并且在 `<script setup>` 中是直接可用的（不需要额外导入）。

4.defineExpose

- 获取组件的实例可以通过ref来获取，接着组件挂载完成后可通过value拿到组件实例。
- 当拿到组件实例后，默认是不可以访问这个实例中的方法和属性，因为默认没暴露任何方法和属性。
- 因此在Vue3组件中可以用defineExpose API来暴露方法和属性给外部访问。
- defineExpose 也是不需要导入，直接使用即可

4. Vue Rouer路由

4.1 vue-router路由的两种模式

vue-router中默认使用的是hash模式

1. hash模式，带#。如：http://localhost:8080/#/page。改变hash，浏览器本身不会有任何请求服务器动作。
2. history模式，不带#， 如：<http://localhost:8080/page ，路径没有#。基于HTML5的pushState、replaceState实现。

hash	history
有 # 号	没有 # 号
能够兼容到IE8	只能兼容到IE10
实际的url之前使用哈希字符，这部分url不会发送到服务器，不需要在服务器层面上进行任何处理	每访问一个页面都需要服务器进行路由匹配生成 html 文件再发送响应给浏览器，消耗服务器大量资源
刷新不会存在 404 问题	浏览器直接访问嵌套路由时，会报 404 问题。
不需要服务器任何配置	需要在服务器配置一个回调路由

4.2 路由跳转时如何传递数据?

动态路由

- path: `/user/:id`
- 获取动态路由的值的方式
    - 在template中，直接通过 $route.params获取值
    - 在created中，通过 this.$route.params获取值
    - 在setup中，使用 vue-router库提供的一个hook useRoute

        - 该Hook会返回一个Route对象，对象中保存着当前路由相关的值

query参数:

- 通过query的方式来传递参数
- 在界面中通过 $route.query 来获取参数
- 在created中，通过 this.$route.query获取值
- 在setup，使用 vue-router库提供的一个hook useRoute 来获取

4.3 什么是路由守卫？路由守卫有什么作用?

什么是路由守卫：

vue-router 提供的路由(导航)守卫主要用来通过跳转或取消的方式守卫导航。有很多种方式植入路由导航中：全局的，单个路由独享的，或者组件级的。

- 全局导航钩子
    - router.beforeEach(to,from,next)
    - router.afterEach(to,from)
    - ....
- 组件内的钩子
    - beforeRouteEnter (to, from, next)
    - beforeRouteUpdate (to, from, next)
    - beforeRouteLeave (to, from, next)
    - ....
- 单独路由独享组件
    - beforeEnter: (to, from, next)
    - afterEach(to,from)
    - ....

路由守卫有什么作用：

- 可以在进入路由之前进行某些判断，比如，检查token是否存在来判断用户是否已经登录。
- 可以在路由守卫中进行页面的权限判断，比如，判断某个用户是否拥有该页面的权限。
- 也可以用来记录页面的某些信息，比如，记录页面的滚动信息等等。

4.4 route和router的区别

- route是路由信息对象，在Vue3中通过 useRoute 来获取。
    - 包括了path，params，hash，query，fullPath，matched，name等路由信息参数。
- router是路由实例”对象，在Vue3中通过 useRouter 来获取。
    - 包括了路由跳转方法、钩子函数等，比如：push、go、back、addRouter、beforeEnter等。

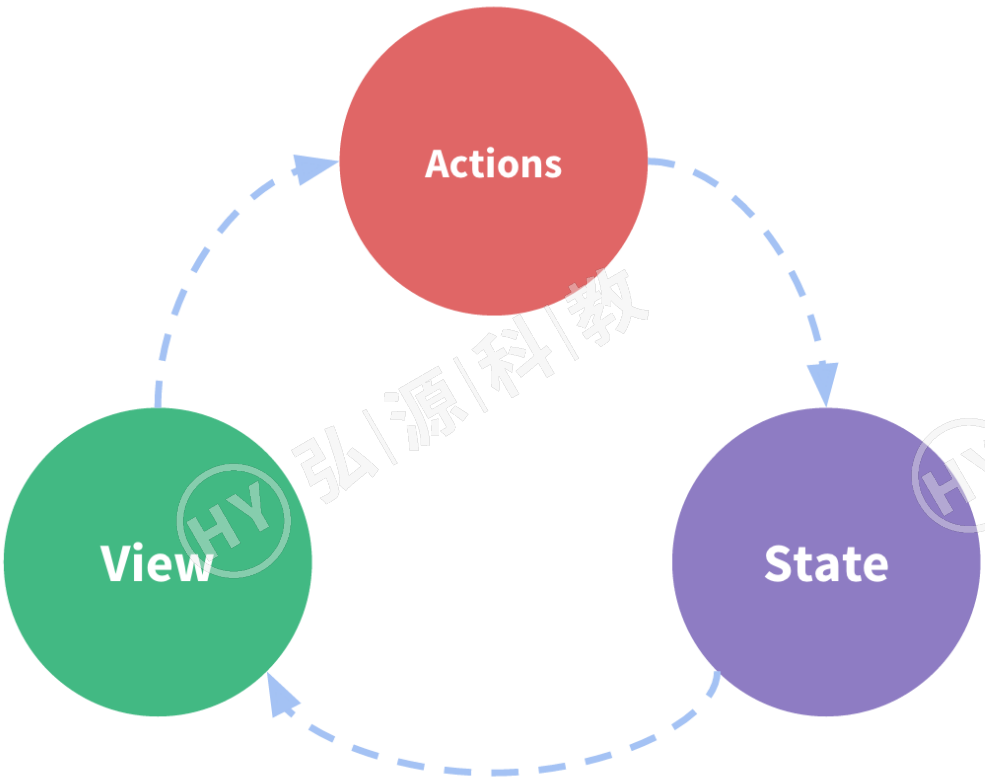
5.Vuex和Pinia状态管理

5.1 什么是状态管理？什么是单项数据流？

在开发中，应用程序是需要处理各种各样的数据，这些数据需要保存在应用程序中的某一个位置，对于这些数据的管理就称之为是 **状态管理**。以前我们是如何管理应用程序的状态？

- 在Vue开发中，我们使用组件化的开发方式。而在组件中我们定义的data或在setup中返回的数据，这些数据我们称之为状态（State）。
- 在模块template中我们可以使用这些数据，模块最终会被渲染成DOM，我们称之为View。
- 在模块中我们会产生一些行为事件，处理这些行为事件时，有可能会修改State，这些行为事件我们称之为Actions。

其实Vue组件内部的数据是以单向数据流的形式来管理数据的。例如，组件的数据定义在State中，接着在View层使用State中的数据，然后View层会产生一些事件Actions，而这些Actions可能会修改State的数据，这就是一个单项数据流的概念。



5.2 什么是Vuex？你使用过 Vuex 吗？

什么是Vuex：

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。

Vuex包括一下几个核心模块：

- 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。

5.3 在哪些场景中会使用到Vuex?

对于简单的项目，是可以不用使用 Vuex，但是对于比较复杂，组件比较多的大项目就需要使用Vuex了。

- 可用于记录收藏、购物车、应用配置信息、用户信息等场景中。
- 可用于记录系统的登录状态、用户权限、部门信息、系统配置信息等场景中。
- 可用于记录城市列表数据、全局枚举、当前坐标等场景中。
- 页面中的组件件嵌套太深了，导致一层层传递数据变的非常麻烦了，也可以将页面数据存到Vuex中。

5.4 什么是Pinia?

什么是Pinia:

- Pinia 是 Vue 的存储库，它允许您跨组件/页面共享状态。
- Pinia适用于Vue2和Vue3，并不需要使用 Composition API。
- Pinia的处理安装之后，它的API也同样适用于SSR的应用程序。

Pinia几个核心概念：

- state
  - state是一个选项，这个选项的值需要是一个函数，函数返回一个对象，对象中存储数据
  - 在组件中拿到当前的store直接使用即可，store.xxx
- getters
  - getters也是一个选项，这个选项的值是一个对象，对象中存储着一个个函数，每个函数可以有一个参数state，通过state可以获取到当前store的state
  - 除此之外每个函数还可以拿到一个this，这个this就是当前的整个store实例
  - 通过这个this，可以想用谁就用谁
  - 在组件中使用也是拿到store直接store.xxx即可
- actions
  - 在actions中，主要存放一个个函数，每个函数最主要的工作发送异步请求，获取到数据后直接修改state
  - 每个action函数并不像getter函数一样，第一个参数是state，它可以没有参数
  - 需要通过this拿到state然后再修改state中的值
  - 在组件中拿到store后直接调用即可，store.xxx()

        - 如果你在此时传递参数，那么就可以在action中拿到参数
- 没有模块modules的概念。

5.5 Pinia 和 Vuex有什么区别?

- Pinia没有Vuex中的mutations选项，因为mutations的出现解决的问题是让devtools进行状态追踪
    - 但是随着技术的发展，Pinia已经解决的这个没有mutation依然可以跟踪状态的问题。
- Pinia可以在任意组件中拿到store然后直接修改state中的任意值
- Pinia不再需要Vuex中的modules这样的嵌套结构，取而代之的是可以创建一个个store
- 使用上的区别
    - 在Vuex中使用某个state时，需要$store.state.xxx
        - 在Pinia中直接拿到store之后store.xxx即可
    - 在Vuex中使用某个getter函数时，需要$store.getters.xxx
        - 在Pinia中拿到store后，store.xxx即可
    - 在Vuex中进行异步请求需要派发action函数
        - 在Pinia中拿到store后，直接调用action函数即可
    - ……

6.Vue其它问题

6.1 什么是虚拟DOM？什么是diff算法?

什么是虚拟DOM：

Virtual DOM 本质上是 JavaScript 对象，是真实 DOM 的描述，用一个 JS 对象来描述一个 DOM 节点。

Virtual DOM 可以看做一棵模拟 DOM 树的 JavaScript 树，其主要是通过 VNode 实现一个无状态的组件，当组件状态发生更新时，然后触发 Virtual DOM 数据的变化，然后通过 Virtual DOM 和真实 DOM 的比对，再对真实 DOM 更新。可以简单认为 Virtual DOM 是真实 DOM 的缓存。

虚拟 DOM 的优缺点:

- 优点：
    - **跨平台与分层设计**(主要原因)：虚拟 DOM 本质上是 JavaScript 对象，而真实 DOM 与平台强相关，相比之下虚拟 DOM 带来了分层设计、跨平台以及 SSR 等特性。至于 Virtual DOM 比原生 DOM 谁的性能好，需要 “控制变量法” 才能比较。这是为什么要设计虚拟 DOM 的主要原因。虚拟 DOM 抽象了原本的渲染过程，实现了跨平台的能力，而不仅仅局限于浏览器的 DOM，可以是安卓和 iOS 的原生组件，也可以是小程序，也可以是各种 GUI。
    - **以最小的代价更新变化的视图**。整棵 DOM 树实现代价太高，能否只更新变化的部分的视图。虚拟 DOM 能通过 patch 准确地转换为真实 DOM，并且方便进行 diff。
    - **保证性能下限**(次要原因)：框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作(分层设计)，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴

的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限。

- **无需手动操作 DOM**：操作 DOM 慢，js 运行效率高。我们可以将 DOM 对比(diff 操作)放在 JS 层，提高效率。我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率。
- **组件的高度抽象化**：Vue.2x 引入 VirtualDOM 把渲染过程抽象化，从而使得组件的抽象能力也得到提升，并且可以适配 DOM 以外的渲染目标。不再依赖 HTML 解析器进行模版解析，可以进行更多的 AOT 工作提高运行时效率：通过模版 AOT 编译，Vue 的运行时体积可以进一步压缩，运行时效率可以进一步提升。**Virtual DOM 的优势不在于单次的操作，而是在大量、频繁的数据更新下，能够对视图进行合理、高效的更新**。为了实现高效的 DOM 操作，一套高效的虚拟 DOM diff 算法显得很有必要.

- 缺点:
  - 无法进行极致优化： 虽然虚拟 DOM + 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。
  - 虽然 Vue 能够保证触发更新的组件最小化，但在单个组件内部依然需要遍历该组件的整个 Virtual DOM 树。
  - 在一些组件整个模版内只有少量动态节点的情况下，这些遍历都是性能的浪费。
  - 传统 Virtual DOM 的性能跟模版大小正相关，跟动态节点的数量无关。

什么是diff算法：

diff 算法是一种通过同层的树节点进行比较的高效算法。diff 整体策略为：深度优先，同层比较。

新旧两个 VNode 节点的左右头尾两侧均有一个变量标识，在遍历过程中这几个变量都会向中间靠拢。当 `oldStartIdx <= oldEndIdx` 或者 `newStartIdx <= newEndIdx` 时结束循环。在遍历中，如果存在 key，并且满足 sameVnode，会将该 DOM 节点进行复用(只通过移动节点顺序)，否则则会创建一个新的 DOM 节点。

`oldStartVnode`、`oldEndVnode` 与 `newStartVnode`、`newEndVnode` 两两比较共有四种比较方法：

1. 当新旧子树的两个开始节点或新旧子树的两个结束节点相同时

   当新旧 VNode 节点的 start 或者 end 满足 sameVnode 时，也就是 sameVnode(oldStartVnode, newStartVnode) 或者 sameVnode(oldEndVnode, newEndVnode) 表示为 true，直接将该 VNode 节点进行 patchVnode 即可（保留）。

2. 当旧子树的开始节点与新子树的结束节点相同时

   当 oldStartVnode 与 newEndVnode 满足 sameVnode，即 sameVnode(oldStartVnode, newEndVnode)。这时候说明 oldStartVnode 已经跑到了 oldEndVnode 后面去了，进行 patchVnode 的同时还需要将真实 DOM 节点移动到 oldEndVnode 的后面。

3. 当旧子树的结束节点与新子树的开始节点相同时

   如果 oldEndVnode 与 newStartVnode 满足 sameVnode，即 sameVnode(oldEndVnode, newStartVnode)。这说明 oldEndVnode 跑到了 oldStartVnode 的前面，进行 patchVnode 的同时真实的 DOM 节点移动到了 oldStartVnode 的前面。

4. 当旧子树中没有新子树中的节点，会将新节点插入到 oldStartVnode 前

如果以上情况均不符合，进入 key 的比较：

- **oldKeyToIdx**：一个哈希表，存放旧节点的 key 与节点的映射关系；如果没有 oldKeyToIdx 则会通过 createKeyToOldIdx 会得到一个 oldKeyToIdx，里面存放旧节点的 key 与节点的映射关系，只不过这个 key 是 index 序列。从 oldKeyToIdx 这个哈希表中可以找到与新节点是否有相同 key 的旧节点，如果同时满足 sameVnode，patchVnode 的同时会将这个真实 DOM（elmToMove）移动到 oldStartVnode 对应的真实 DOM 的前面。
- **idxInOld**：拿新节点的 key 去 oldKeyToIdx 找是否有与旧节点相同的节点，即旧节点中是否有与新节点 key 相同的节点，没有就通过 findIdxInOld 遍历旧节点并通过 sameVnode 判断是否有相同节点，有返回索引。
    - idxInOld 不存在，即新节点在旧节点中都没有找到，说明这是一个之前没有的新节点，需要通过 createElm 创建新节点
    - idxInOld 存在，则进一步通过 sameVnode(vnodeToMove, newStartVnode) 判断是否是同一节点
        - 是同一节点，则通过 patchVnode 更新，并移动节点
        - 不是同一节点，即相同的 key 不同的元素，则通过 createElm 创建新节点

先 `oldStartVnode、oldEndVnode` 与 `newStartVnode、newEndVnode` 两两通过 sameVnode 进行 4 次比较，若成立，则通过 patchVnode 更新节点内容，并移动节点位置。若不成立，再进一步比较 key，idxInOld 判断新节点是否被旧节点复用了。idxInOld 不存在，说明旧节点没有复用新节点，新节点需要 createElm 创建；idxInOld 存在，说明新节点有被复用的可能性，为什么这么说，因为此时我们只知道节点的 key 相同，是否是通过简单的**通过移动节点位置达到复用的目的**，还是说通过创建节点进行原地复用或就地修改，需要进一步通过 sameVnode(vnodeToMove, newStartVnode) 判断是否是同一节点。是同一节点，则直接通过 patchVnode 更新，并移动节点；否则，虽然有相同的 key 但是不同的元素，则通过 createElm 创建新节点，就地修改。从这一点我们就可以思考出为什么 `v-for` 的时候要加上 key？为什么这个 key 不建议 index 来标识？

6.2 Vue2 和 Vue3 响应性原理

Vue.js 2.0 采用数据劫持并结合了发布者-订阅者模式，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。

官网的描述：https://cn.vuejs.org/v2/guide/reactivity.html

Vue.js 3.0放弃了Object.defineProperty API，而使用了更快的Proxy API。Proxy 是在 ES6 中引入，它允许你拦截对该对象的任何交互，也可以避免 Vue 早期版本中存在的一些响应性问题。

官网描述：https://v3.cn.vuejs.org/guide/reactivity.html

6.3 nextTick 方法的实现原理

- 当调用nexttick函数时，nexttick内部会将回调函数使用Promise来包裹，目的是将该回调函数加入到微任务队列中。
- 在队列中的任务都是先进先出的，所以当执行完主程序的代码之后就会执行微任务队列中nexttick的回调函数，那这个过程就称为一次tick。

- 所以nexttick的回调函数将会推迟到下一个 DOM 更新周期之后执行。

6.4 Proxy 与 Object.defineProperty 优劣对比

Proxy 的优势如下:

- Object.defineProperty 只能劫持对象的属性，而 Proxy 是直接代理对象。

由于 Object.defineProperty 只能对属性进行劫持，需要遍历对象的每个属性，如果属性值也是对象，则需要深度遍历。而 Proxy 直接代理对象，不需要遍历操作。

- Object.defineProperty 对新增属性需要手动进行 Observe。

由于 Object.defineProperty 劫持的是对象的属性，所以新增属性时，需要重新遍历对象，对其新增属性再使用 Object.defineProperty 进行劫持。

也正是因为这个原因，使用 Vue 给 data 中的数组或对象新增属性时，需要使用 vm.$set 才能保证新增的属性也是响应式的。

- Proxy支持 13 种拦截操作，这是 defineProperty 所不具有的。
    - get(target, propKey, receiver)：拦截对象属性的读取，比如 proxy.foo 和proxy['foo']。
    - set(target, propKey, value, receiver)：拦截对象属性的设置，比如proxy.foo = v 或 proxy['foo'] = v，返回一个布尔值。
    - has(target, propKey)：拦截 propKey in proxy 的操作，返回一个布尔值。
    - deleteProperty(target, propKey)：拦截 delete proxy[propKey] 的操作，返回一个布尔值。
    - ownKeys(target)：拦截Object.getOwnPropertyNames(proxy)、Object.getOwnPropertySymbols(proxy)、Object.keys(proxy)、for...in循环，返回一个数组。该方法返回目标对象所有自身的属性的属性名，而 Object.keys() 的返回结果仅包括目标对象自身的可遍历属性。
    - getOwnPropertyDescriptor(target, propKey)：拦截 Object.getOwnPropertyDescriptor(proxy, propKey)，返回属性的描述对象。
    - defineProperty(target, propKey, propDesc)：拦截Object.defineProperty(proxy, propKey, propDesc）、Object.defineProperties(proxy, propDescs)，返回一个布尔值。
    - preventExtensions(target)：拦截 Object.preventExtensions(proxy)，返回一个布尔值。
    - getPrototypeOf(target)：拦截 Object.getPrototypeOf(proxy)，返回一个对象。
    - isExtensible(target)：拦截 Object.isExtensible(proxy)，返回一个布尔值。
    - setPrototypeOf(target, proto)：拦截 Object.setPrototypeOf(proxy, proto)，返回一个布尔值。如果目标对象是函数，那么还有两种额外操作可以拦截。
    - apply(target, object, args)：拦截 Proxy 实例作为函数调用的操作，比如proxy(...args)、proxy.call(object, ...args)、proxy.apply(...)。
    - construct(target, args)：拦截 Proxy 实例作为构造函数调用的操作，比如new proxy(...args)。
- 新标准性能红利

  Proxy 作为新标准，从长远来看，JS 引擎会继续优化 Proxy，但 getter 和 setter 基本不会再有针对性优化。

- Proxy 兼容性差

  Proxy 对于 IE 浏览器来说简直是灾难。 并且目前并没有一个完整支持 Proxy 所有拦截方法的 Polyfill 方案，有一个 Google 编写的 proxy-polyfill 也只支持了 get、set、apply、construct 四种拦截，可以支持到 IE9+ 和 Safari 6+。

Object.defineProperty 的优势如下:

- 兼容性好，支持 IE9。

最后总结:

1. Object.defineProperty 并非不能监控数组下标的变化，Vue2.x 中无法通过数组索引来实现响应式数据的自动更新是 Vue 本身的设计导致的，不是 defineProperty 的锅。
2. Object.defineProperty 和 Proxy 本质差别是，defineProperty 只能对属性进行劫持，所以出现了需要递归遍历，新增属性需要手动 Observe 的问题。
3. Proxy 作为新标准，浏览器厂商势必会对其进行持续优化，但它的兼容性也是块硬伤，并且目前还没有完整的 polyfill 方案。

6.5 使用过 Vue SSR 吗？说说 SSR?

> Vue.js 是构建客户端应用程序的框架。默认情况下，可以在浏览器中输出 Vue 组件，进行生成 DOM 和操作 DOM。然而，也可以将同一个组件渲染为服务端的 HTML 字符串，将它们直接发送到浏览器，最后将这些静态标记"激活"为客户端上完全可交互的应用程序。
>
> 即：SSR大致的意思就是vue在客户端将标签渲染成的整个 html 片段的工作在服务端完成，服务端形成的html 片段直接返回给客户端这个过程就叫做服务端渲染。

服务端渲染 SSR 的优缺点如下：

（1）服务端渲染的优点：

- 更好的 SEO： 因为 SPA 页面的内容是通过 Ajax 获取，而搜索引擎爬取工具并不会等待 Ajax 异步完成后再抓取页面内容，所以在 SPA 中是抓取不到页面通过 Ajax 获取到的内容；而 SSR 是直接由服务端返回已经渲染好的页面（数据已经包含在页面中），所以搜索引擎爬取工具可以抓取渲染好的页面；
- 更快的内容到达时间（首屏加载更快）： SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，文件下载等需要一定的时间等，所以首屏渲染需要一定的时间；SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，所以 SSR 有更快的内容到达时间；

(2) 服务端渲染的缺点：

- 更多的开发条件限制： 例如服务端渲染只支持 beforCreate 和 created 两个钩子函数，这会导致一些外部扩展库需要特殊处理，才能在服务端渲染应用程序中运行；并且与可以部署在任何静态文件服务器上的完全静态单页面应用程序 SPA 不同，服务端渲染应用程序，需要处于 Node.js server 运行环境；
- 更多的服务器负载：在 Node.js 中渲染完整的应用程序，显然会比仅仅提供静态文件的 server 更加大量占用CPU 资源 (CPU-intensive - CPU 密集)，因此如果你预料在高流量环境 (high traffic) 下使用，请准备相应的服务器负载，并明智地采用缓存策略。

6.6 你有对 Vue 项目进行哪些优化?

（1）代码层面的优化

- v-if 和 v-show 区分使用场景
- computed 和 watch 区分使用场景

- v-for 遍历必须为 item 添加 key，且避免同时使用 v-if
- 长列表性能优化
- 事件的销毁
- 图片资源懒加载
- 路由懒加载
- 第三方插件的按需引入
- 优化无限列表性能
- 服务端渲染 SSR or 预渲染

（2）打包层面的优化

- Webpack 对图片进行压缩
- 减少 ES6 转为 ES5 的冗余代码
- 提取公共代码
- 模板预编译
- 提取组件的 CSS
- 优化 SourceMap
- 构建结果输出分析
- Vue 项目的编译优化

（3）其它的优化

- 开启 gzip 压缩
- 浏览器缓存
- CDN 的使用
- 使用 Chrome Performance 查找性能瓶颈

6.7 如何实现Vue首屏加载优化的

1. 把不常改变的库放到index.html中，并接入CDN提速
2. Vue路由的懒加载，Vue组件尽量不要全局引入
3. 使用轻量级的工具库
4. 减少首页的资源请求数，减少资源的大小
5. 避免使用大图，图片使用懒加载
6. 等等

6.8 Vue3.0 里为什么要用 Proxy API替代 defineProperty API

- defineProperty API 的局限性最大原因是它只能针对对象的属性做监听。

  Vue2.x中的响应式实现正是基于Object.defineProperty来实现，对 data 中的属性做了遍历 + 递归，为每个属性设置了 getter、setter。这也就是为什么 Vue 只能对 data 中预定义过的属性做出响应的原因。

- Proxy API监听是针对一个对象的，那么对这个对象的所有操作会进入监听操作， 这就完全可以代理所有属性，将带来很大的性能提升和更优的代码。

Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。

- Proxy响应式是惰性的。

  在 Vue.js 2.x 中，对于一个深层属性嵌套的对象，要劫持它内部深层次的变化，就需要递归遍历这个对象，执行 Object.defineProperty 把每一层对象数据都变成响应式的，这无疑会有很大的性能消耗。

  在 Vue.js 3.0 中，使用 Proxy API 并不能监听到对象内部深层次的属性变化，因此它的处理方式是在 getter 中去递归响应式，这样的好处是真正访问到的内部属性才会变成响应式，简单的可以说是按需实现响应式，减少性能消耗。

7. Axios网络请求库

7.1 Axios是什么？怎么使用它，怎么解决跨域?

Axios是什么?

Axios 是一个基于 promise 的 HTTP 库，可以用在浏览器和 node.js 中。前端最流行的 ajax 请求库，react/vue 官方都推荐使用 axios 发 ajax 请求

Axios特点:

- 基于 promise 的异步 ajax 请求库，支持promise所有的API
- 浏览器端/node 端都可以使用，浏览器中创建XMLHttpRequests
- 支持请求 / 响应拦截器
- 支持请求取消
- JSON数据的自动转换

  可以转换请求数据和响应数据，并对响应回来的内容自动转换成 JSON类型的数据
- 批量发送多个请求
- 安全性更高，客户端支持防御 XSRF。

  就是让你的每个请求都带一个从cookie中拿到的key, 根据浏览器同源策略，假冒的网站是拿不到你cookie中得key的，这样，后台就可以轻松辨别出这个请求是否是用户在假冒网站上的误导输入，从而采取正确的策略。

常用语法:

- axios(config): 通用/最本质的发任意类型请求的方式
- axios.request(config): 等同于 axios(config)
- axios.get(url[, config]): 发 get 请求
- axios.post(url[, data, config]): 发 post 请求
- axios.defaults.xxx: 请求的默认全局配置
- axios.interceptors.request.use(): 添加请求拦截器
- axios.interceptors.response.use(): 添加响应拦截器
- axios.create([config]): 创建一个新的 axios(它没有下面的功能)
- axios.CancelToken(): 用于创建取消请求的 token 对象

- axios.all(promises): 用于批量执行多个异步请求

开发环境中解决跨域：

- 方式一：在vue.config.js中的devServer选项中的proxy中配置反向代理
- 方式二：在vite.config.js中的server选项中的proxy中配置反向代理
- 方式三：直接后端开发人员配置cors

部署的时候解决跨域：

- 方式一：直接后端开发人员配置cors
- 方式二：反向代理，比如：nginx 服务器的反向代理。

8. Vue VS React

8.1 Vue和React有什么不同？使用场景分别是什么？

Vue和React的区别：

虽然Vue和React两者在定位上有一些交集，但差异也是很明显的。

- **Vue 使用的是 web 开发者更熟悉的模板与特性。**

  Vue的API跟传统web开发者熟悉的模板契合度更高，比如Vue的单文件组件是以模板+JavaScript+CSS的组合模式呈现，它跟web现有的HTML、JavaScript、CSS能够更好地配合。

- **React 的特色在于函数式编程的理念和丰富的技术选型。**

  Vue 比起 React 更容易被前端工程师接受，这是一个直观的感受；React 则更容易吸引在 FP 上持续走下去的开发者。

- 从**使用习惯和思维模式**上考虑，对于一个没有任何Vue和React基础的web开发者来说， Vue会更友好，更符合他的思维模式。
- React对于拥有函数式编程背景的开发者以及一些并不是以web为主要开发平台的开发人员而言，React更容易接受。
- 这并不意味着他们不能接受Vue，Vue和React之间的差异对他们来说就没有web开发者那么明显。可以说，**Vue更加注重web开发者的习惯。**

各自使用场景：

React：适合构建大型应用程序，其函数编程让React开发应用时非常的灵活，并且React对TS支持非常的友好，也有非常大的生态系统。

Vue：适合使用模板搭建小而快的应用；适合简单、对灵活度要求没那么高的应用。自从Vue3出来后，对TS支持也越来越友好，生态系统也在慢慢的完善起来了。