VUE篇

1、谈谈你对vue的理解，或者问vue的优点有哪些?

- vue是一个轻量级框架：只关注视图层，大小只有几十 `kb` ；
- 简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学习；
- 双向数据绑定、有指令系统，在数据操作方面非常简单；
- 支持定义组件化，在构建单页应用方面有优势；
- 视图，数据，结构分离，很容易就能修改数据
- 内部采用了虚拟DOM：`dom` 操作是非常耗费性能的，不再使用原生的 `dom` 操作节点，极大解放 `dom` 操作，但底层还是避免不了操作DOM的。
- 再就是运行速度更快。

2、v-if和v-show区别

v-if 是真正的条件渲染，因为它会创建和销毁元素或者组件，所以会触发组件的生命周期。

v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 的 “display” 属性进行切换。

所以，v-if 适用于在运行时很少改变条件，不需要频繁切换条件的场景；v-show 则适用于需要非常频繁切换条件的场景。

3、Vue初始化过程或者new Vue发生了什么

- `new Vue`的时候调用会调用`_init`方法
  - 定义 **`$set`**、**`$watch`**、`$get` 、`$delete`、 等方法
  - 定义 `$on`、`$off`、`$emit`、`$off`等事件
  - 定义 `_update`、`$forceUpdate`、`$destroy`生命周期
- 调用`$mount`进行页面的挂载
- 挂载的时候主要是通过`mountComponent`方法
- 定义`updateComponent`更新函数
- 执行`render`生成虚拟`DOM`
- `_update`将虚拟`DOM`生成真实`DOM`结构，并且渲染到页面中

4、Vue实例挂载的过程中发生了什么?

挂载过程指的是app.mount()过程，这个过程中整体上做了两件事：**初始化**和**建立更新机制**

初始化会创建组件实例、初始化组件状态，创建各种响应式数据

建立更新机制这一步会立即执行一次组件更新函数，这会首次执行组件渲染函数并执行patch将前面获得vnode转换为dom；同时首次执行渲染函数会创建它内部响应式数据之间和组件更新函数之间的依赖关系，这使得以后数据变化时会执行对应的更新函数。

5、可以把vue挂载到body或者根元素上吗?

不可以，只能挂载到普通元素上。

是因为所谓的挂载其实是替换，而不是插入。

把数据渲染到视图上，然后替换掉要挂载的元素。

如果挂载到body上，那body就被替换掉了。页面没有body自然就不对了。

> ### 如果非要挂载到body上，怎么办?
>
> 也不是不可以。比如我之前自定义过对话框组件（dialog组件），需要做一个覆盖全屏的遮罩层，这个遮罩层是固定定位，希望参照的父元素时body。所以要挂载到body下。
>
> 解决办法就是在mouted生命周期里，手动的吧this.$el 添加到body下。
>
> ```
> document.body.appendChild(this.$el);
> ```
>
> 然后在组件销毁前再从body上移除
>
> ```
> document.body.removeChild(this.$el);
> ```

6、请描述下你对vue生命周期的理解?

beforeCreate（创建前）： 这时候几乎啥也访问不到，比如data啊、props、方法啊，this

created（创建后）：可以发访问到data，但不能访问到 `$el` 属性。可以在这里做ajax

beforeMount（挂载前）：这时候会编译模板，把data里面的数据和模板生成html。但还没有挂载html到页面上。

mounted（挂载后）：编译好的html内容更新到html 页面上了。可以获取到DOM节点了。

beforeUpdate（更新前）：数据更新了，但是视图还没有更新

updated（更新后）：数据更了，视图也更新了。

beforeDestroy（销毁前）：实例销毁之前调用，this能获取到实例，通常在这里取消事件绑定

destroyed（销毁后）：实例销毁后调用，事件监听会被移除，子实例也会被销毁。

7、v-if和v-for的优先级是什么?

实践中**不应该把v-for和v-if放一起**

在**vue2中**，**v-for的优先级是高于v-if**，把它们放在一起，输出的渲染函数中可以看出会先执行循环再判断条件，哪怕我们只渲染列表中一小部分元素，也得在每次重渲染的时候遍历整个列表，这会比较浪费；另外需要注意的是在**vue3中则完全相反，v-if的优先级高于v-for**，所以v-if执行时，它调用的变量还不存在，就会导致异常

通常有两种情况下导致我们这样做：

- 为了**过滤列表中的项目** (比如 `v-for="user in users" v-if="user.isActive"`)。此时定义一个计算属性 (比如 `activeUsers`)，让

其返回过滤后的列表即可（比如`users.filter(u=>u.isActive)`）。
- 为了**避免渲染本应该被隐藏的列表**(比如 `v-for="user in users" v-if="shouldShowUsers"`)。此时把 `v-if` 移动至容器元素上(比如 `ul`、`ol`)或者外面包一层`template`即可。

文档中明确指出**永远不要把 `v-if` 和 `v-for` 同时用在同一个元素上**，显然这是一个重要的注意事项。

源码里面关于代码生成的部分，能够清晰的看到是先处理v-if还是v-for，顺序上vue2和vue3正好相反，因此产生了一些症状的不同，但是不管怎样都是不能把它们写在一起的。

8、对 SPA 单页面的理解，它的优缺点分别是什么?

SPA（ single-page application ）就是单页面应用。也就是整个应用只有一个HTML文件，然后页面跳转也是在页面内部通过js来实现的，不会跳转到其他HTML页面。

优点：

- 用户体验好、快，内容的改变不需要重新加载整个页面
- 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：

- 首屏加载慢，因为第一次还是要加载很多文件，包括js啊，css啊（解决办法看下一题）

- 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；
- 不利于SEO，页面内容是js生成的，爬虫不容爬取到。（解决办法看SSR）

9、SPA首屏加载速度慢的怎么解决?

减少首屏加载的文件，对一些不需要在首屏显示的文件不去加载，对于图片可以用懒加载。

还有就是用服务端渲染，即能解决加载慢的问题，还能解决SEO的问题。

10、SSR（Server Side Render）解决了什么问题？有做过SSR吗？你是怎么做的?

SSR也就是服务端渲染，也就是将Vue在客户端的渲染工作放在服务端完成，然后再把html直接返回给客户端

SSR的优势：

- 更好的SEO
- 首屏加载速度更快

SSR的缺点：

- 开发条件会受到限制，服务器端渲染只支持beforeCreate和created两个钩子；
- 需要处于Node.js的运行环境；
- 会加大服务端的负载。

11、为什么data属性是一个函数而不是一个对象?

因为在js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，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,当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。

12、动态给vue的data添加一个新的属性时会发生什么？怎样解决?

视图不会更新

可以使用**$set()**

> 原因是： vue的数据响应式原理是，在初始化的时候，会通过数据劫持的方式Object.defefineProperty来给对象的属性，添加监听，这样当data的属性发送变化时，就可以监听到，进而更新视图。
>
> 但是之后动态添加的属性，没有经过劫持，那么新的属性的变化就不会引起视图的更新。
>
> 而$set方法，内部会 调用Object.defefineProperty，来去监听新的属

> 性。

Vue中组件和插件有什么区别?

- 在vue中，组件是把结构、样式和功能封装到一起的模块，比如按钮组件、轮播图等。
- 插件通常用来为 `Vue` 添加全局功能，比如路由啊，vuex状态管理啊

14、Vue组件之间的通信方式都有哪些?

- 父组件传给子组件用**prop**
- 子组件传给父组件用自定义事件 **$emit**,其实也可以用prop，就是父组件先传给子组件一个回调函数，子组件调用回调把参数传给父组件。
- 跨组件可以用**event bus**，**Vue.observable**
- 在就是用**vuex**

15、双向数据绑定是什么，或者问vue中数据绑定的原理是什么?

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。

使用 Object.defineProperty() 来进行数据劫持有什么缺点?

在对一些属性进行操作时，使用这种方法无法拦截，比如通过下标方式修改数组数据或者给对象新增属性，这都不能触发组件的重新渲染，因为 Object.defineProperty 不能拦截到这些操作，只是 Vue 内部通过重写函数的方式解决了这个问题。

在 Vue3.0 中已经不使用这种方式了，而是通过使用 Proxy 对对象进行代理，从而实现数据劫持。使用Proxy 的好处是它可以监听到任何方式的数据改变，唯一的缺点是兼容性的问题，因为 Proxy 是 ES6 的语法。

17、Vue中的$nextTick有什么作用?

- 在数据变化后执行的某个操作，而这个操作需要用到更新之后的DOM结构，那这个操作就需要放到`nextTick()`的回调函数中。
- 它原理其实就是把这个操作放到一个异步任务中去。他内部会先采用微任务，比如**Promise**实现，不行的话改成**setTimeout**（宏任务）。

> ### 又或者问你：想在created生命周期中做DOM操作怎么办?
>
> 在vue生命周期中，如果在created()钩子进行DOM操作，也一定要放在`nextTick()`的回调函数中。
>
> 因为在created()钩子函数中，页面的DOM还未渲染，这时候也没办法操作DOM，所以，此时如果想要操作DOM，必须将操作的代码放在`nextTick()`的回调函数中。

18、说说你对vue的mixin的理解，有什么应用场景?

mixin可以让组件之间的部分代码重用

如果希望在多个组件之间重用一组组件选项，例如生命周期 、 方法等，则可以将其编写为 mixin，并在组件中引用。然后 mixin 的内容就会合并到组件中。

说说你对slot的理解？slot使用场景有哪些?

slot就是插槽，用来做组件嵌套的。通过插槽，可以告诉子组件放在父组件的什么位置。

包括匿名插槽（默认），具名插槽还有作用域插槽。

> 具名插槽，在组件内，给多个slot设置name属性，然后使用的时候，在template标签上，用v-slot来指定slot名称。
>
> 作用域插槽，可以实现组件向插槽内的组件或元素传递数据。

20、Vue.observable你有了解过吗？说说看

有点类似vuex，可以做状态管理。

`Vue.observable`，让一个对象变成响应式数据。`Vue` 内部会用它来处理 `data` 函数返回的对象。

它里面的数据发生变化时，会触发视图的更新。可以用来做跨组件的通信。

21/有没有阅读过vue源码?

看过一点，大概了解**vue初始过程**和**数据响应的原理**

阅读源码后对VUE的理解~

22、你知道vue中key的原理吗？说说你对它的理解

1. 让vue精准的追踪到每一个元素，`高效的更新虚拟DOM`。
2. 触发过渡

```
<transition>
  <span :key="text">{{ text }}</span>
</transition>
复制代码
```

当text改变时，这个元素的key属性就发生了改变，在渲染更新时，Vue会认为这里新产生了一个元素，而老的元素由于key不存在了，所以会被删除，从而触发了过渡。

key的原理分析

说说你对keep-alive的理解是什么？或者问，怎么缓存当前组件

如果需要在组件切换的时候，保存一些组件的状态防止多次渲染，就可以使用 keep-alive 组件包裹需要保存的组件。

可以使用**keep-alive**

24、Vue常用的修饰符有哪些有什么应用场景

> 简单或，只说修饰符的名称就行，详细说的话，解释下用法和作用是什么

修饰符是加载事件绑定或者v-model指令后的，包括：

`.stop`：等同于 JavaScript 中的 `event.stopPropagation()`，阻止事件冒泡；

`.prevent`：等同于 JavaScript 中的 `event.preventDefault()`，阻止默认事件

`.capture`：与事件冒泡的方向相反，事件捕获由外到内；

`.self`：只会触发自己范围内的事件，不包含子元素；

`.once`：只会触发一次。

`.number`把输入框输入的数值型字符串转成数字

`.trim`去除收尾空格

`.lazy`把`v-model`默认的`input`事件换成`change`事件

还有按键修饰符，比如`.enter`,当按回车键在触发事件等。

你有写过自定义指令吗？自定义指令的应用场景有哪些?

在vue中，**需要做DOM操作，可以定义指令**。

定义指令有两种方式，**全局指令和组件内的局部指令**。

比如定义全局指令，通过**Vue.directive()**来定义。

参数是**指令名称**和一个**对象**或者是回调函数。

在这个对象中是一些**钩子函数，有bind，inserted**、update、unbind等。

在钩子函数中可以获取到使用指令的**元素el**和和**绑定在指令上的值**。

比如我定义过**图片懒加载**的指令。

26、Vue中的过滤器了解吗？过滤器的应用场景有哪些?

在vue，过滤器主要用来格式化显示数据的，过滤器不会修改数据，而是改变用户看到的输出

使用场景：

- 需要格式化数据的情况，比如需要处理时间、价格等数据格式的输出显示。
- 比如后端返回一个 **年月日的日期字符串**，前端需要展示为 **多少天前** 的数据格式，此时就可以用过滤器来处理数据。

过滤器是一个函数，它会把表达式中的值始终当作函数的第一个参数。过滤器用在**插值表达式 *`{{ }}` 和 `v-bind`* 表达式** 中，然后放在操作符“ | ”后面进行指示。

27、什么是虚拟DOM？如何实现一个虚拟DOM？说说你的思路

概念：

虚拟DOM其实就是一个描述DOM的js对象。我们知道DOM的属性非常多，有将近300个吧。那频繁操作DOM代价就会比较大，非常耗时。

原理：

比如说：

你用传统的原生去操作`DOM`时，需要更新10个`DOM`节点，浏览器没收到第一个更新`DOM`请求后，并不知道后续还有9次更新操作，因此会马上执行流程，最终执行10次流程

而通过虚拟DOM，同样更新10个`DOM`节点，虚拟`DOM`不会立即操作`DOM`，而是将这10次更新合并成一次，那速度就快很多了。

实现：

它的核心无非就几个关键属性，标签名、数据、子节点等。

简单写一个,定一个对象，然后记录他的标签名，属性，还有子元素等

```
{
  tagname:"div",
  attrs:["id","class"],
  textNode:"",
  children:[
    {
      tagname:"h1",
      attrs:["class"],
      textNode:"hello world"
    }
  ]
}
```

虚拟DOM的好处：

- 将真实元素节点抽象成 VNode，有效减少直接操作 dom 次数，从而提高程序性能
- 方便实现跨平台，针对不同的平台，渲染成不同的控件。

你了解vue的diff算法吗？说说看

diff算法我简单了解一点

组件更新核心是响应式数据监控到数据的改变，重新生成了虚拟dom树，然后通过diff算法计算出前后虚拟dom树的差异点，更新dom时只更新变化的部分。

它其有两个特点：

- 比较只会在同层级进行, 不会跨层级比较，然后采用深度优先的策略
- 在diff比较的过程中，循环从两边向中间比较

[你了解diff算法吗?](#)

29、Vue项目中有封装过axios吗？主要是封装哪方面的?

有封装过。

主要封装设置请求头，超时时间等。

可以通过axios.create方法创建axios实例。然后设置请求头、超时时间、接口前缀等。

```
const service = axios.create({
  baseURL: "http://localhost:3000", // 接口前缀
  timeout: 5000 // 请求超时时间
})
```

然后通过拦截器设置请求头和响应头：

在请求头中添加token，这样请求的时候，就会自动携带token到后端。

```
service.interceptors.request.use(
  config => {
    if (store.getters.token) {
      config.headers['Authorization'] = getToken()
    }
    return config
  },
  error => {
    return Promise.reject(error)
  }
)
```

在响应头中可以根据响应的数据，自动显示相应的提示：

```
// response interceptor
service.interceptors.response.use(
  response => {
    const res = response.data
    if (res.code !== 200) {
      Message({
        message: res.message || 'Error',
        type: 'error',
        duration: 5 * 1000
      })

      return Promise.reject(new Error(res.message || 'Error'))
    } else {
      return res
    }
  }
)
```

然后在api.js中引入，再进一步封装接口：

```
import request from '@/utils/request'

export function login(data) {
  return request({
    url: 'api/login',
    method: 'post',
    data
  })
}
```

这样就可以把`api`统一管理起来，以后维护修改只需要在`api.js`文件操作就行了。

30、你了解axios的原理吗？有看过它的源码吗？

axios其实就是promis封装的ajax，我有自己使用promis封装过ajax。

31、说下你的vue项目的目录结构，如果是大型项目你该怎么划分结构和划分组件呢？

在src下，主要有这么几个目录：

- api： 接口文件
- components：公共的组件，比如header组件
- constant：常量文件
- utils：放一些工具函数
- directives：放全局的自定义指令

- views：放页面组件，每一个页面也都单独定位成一级目录，目录里放页面的根组件和相关组件
- layout：放整体布局组件
- router：放路由
- store：放状态管理的文件
- styles：放公共的样式文件，包括reset.css,公共样式，动画等

32、vue要做权限管理该怎么做？如果控制到按钮级别的权限怎么做？

权限管理一般需求是**页面权限**和**按钮权限**的管理

具体实现的时候分后端和前端两种方案：

前端方案会**把所有路由信息在前端配置**，通过路由守卫要求用户登录，用户**登录后根据角色过滤出路由表**。比如我会配置一个`asyncRoutes`数组，需要认证的页面在其路由**的`meta`中添加一个`roles`字**段，等获取用户角色之后取两者的交集，若结果不为空则说明可以访问。此过滤过程结束，剩下的路由就是该用户能访问的页面，**最后通过`router.addRoutes(accessRoutes)`方式动态添加路由**即可。

后端方案会**把所有页面路由信息存在数据库**中，用户登录的时候根据其角色**查询得到其能访问的所有页面路由信息**返回给前端，前端**再通过`addRoutes`动态添加路由**信息

按钮权限的控制通常会**实现一个指令**，例如`v-permission`，**将按钮要求角色通过值传给v-permission指令**，在指令的`moutned`钩子中可以**判断当前用户角色和按钮是否存在交集**，有则保留按钮，无则移除按钮。

纯前端方案的优点是实现简单，不需要额外权限管理页面，但是维护起来问题比较大，有新的页面和角色需求就要修改前端代码重新打包部署；服务端方案就不存在这个问题，通过专门的角色和权限管理页面，配置页面和按钮权限信息到数据库，应用每次登陆时获取的都是最新的路由信息，可谓一劳永逸！

Vue项目中你是如何解决跨域的呢?

- jsonp
- CORS
- 代理

CORS 后端设置响应头，允许跨域。

代理的话，在开发阶段，可以在vue.config.js 中配置devServer

```
module.exports = {
    devServer: {
        host: '127.0.0.1',
        port: 8084,
        open: true,// vue项目启动时自动打开浏览器
        proxy: {
            '/api': { // '/api'是代理标识，用于告诉node，url前面是/api的就是使用代理的
                target: "http://xxx.xxx.xx.xx:8080", //目标地址，一般是指后台服务器地址
                changeOrigin: true, //是否跨域
                pathRewrite: { // pathRewrite 的作用是把实际Request Url中的'/api'用""代替
                    '^/api': ""
                }
```

```
            }
        }
    }
}
```

生产环境的话，一般是在nginx服务器上配置反向代理。

34、vue项目本地开发完成后部署到服务器后报404是什么原因呢?

主要是**因为采用了history路由的原因**。

由于我们做的是SPA**单页应用，打包完就只有一个index.html文件。nginx的配置在访问域名时，就会打开项目目录下的index.html文件。而如果地址栏不是index.html，就会报错了，因为没有哪个文件嘛**。

解决办法也很简单，就是**修改配置，让访问任意页面的时候，都重定向到index.html,把路由交给前端来处理**。

而hash路由就不会有问题。因为hash值的改变，不会发起http请求，不会重新加载页面。

35、你是怎么处理vue项目中的错误的?

常见的错误主要有两种，接口错误和代码错误。

如果是**接口的错误，可以在axios使用拦截器，拦截响应，根据状态码进行响应的处理**。

如果是代码的错误，可以用**Vue.config.errorHandler**定义全局的处理函数。

```
Vue.config.errorHandler = function (err, vm, info) {
   //在这里做错误处理
}
```

36、vue3有了解过吗？能说说跟vue2的区别吗？

有了解过。

首先：

vue3更新了数据绑定的方式。在**vue2中使用的是Object.definedProperty**方法。

vue2的问题是：

- 检测不到对象属性的添加和删除
- 数组`API`方法无法监听到
- 需要对每个属性进行遍历监听，如果嵌套对象，需要深层监听，造成性能问题

Vue3改成了Proxy（代理）。

`Proxy`的监听是针对一个对象的，对这个对象的所有操作会进入监听操作，这样就完全可以代理所有属性了。

第二：

Vue3新增了组合式API（Composition API）

相对于Vue2来说，可以提高代码的复用性。在逻辑组织和逻辑复用方面优于vue2的。Option API

新的**setup语法糖**，更加方便写组合api了。

第三：

还有就是增加了**Treeshaking**，可以去除无用的代码，减小代码体积。基本原理是：

- 编译阶段利用`ES6 Module`判断哪些模块已经加载
- 判断哪些模块和变量未被使用或者引用，进而删除对应代码

Vue3.0 所采用的 Composition Api 与 Vue2.x 使用的 Options Api 有什么不同?

`Composition API`是一组API，包括：**Reactivity(响应式) API、生命周期钩子，使用户可以通过导入函数方式编写vue组件**。而`Options API`则通过声明组件选项的对象形式编写组件。

`Composition API`最主要作用是能够**简洁、高效复用逻辑**。解决了过去`Options API`中`mixins`的各种缺点；**

另外`Composition API`具有**更加敏捷的代码组织能力**，很多用户喜欢`Options API`，认为所有东西都有固定位置的选项放置代码，但是单个组件增长过大之后这反而成为限制，一个逻辑关注点分散在组件各处，形成代码碎片，维护时需要反复横跳，`Composition API`则可以将它们有效组织在一起。

最后`Composition API`拥有**更好的类型推断**，对ts支持更友好，`Options API`在设计之初并未考虑类型推断因素，虽然官方为此做了很多复杂的类型体操，确保用户可以在使用`Options API`时获得类型推断，然而还是没办法用在mixins和provide/inject上。

Vue3首推`Composition API`，但是这会让我们在代码组织上多花点心思，因此在选择上，如果我们项目属于**中低复杂度的场景**，`Options API`仍是一个好选择。对于那些**大型，高扩展，强维护**的项目上，`Composition API`会获得更大收益。

说说Vue 3.0中Treeshaking（摇树）特性？举例说明一下?

概念：

基于ES6提供的模块系统对代码进行静态分析,并将代码中的死代码（dead code)移除的一种技术。因此，利用Tree Shaking技术可以很方便地实现我们代码上的优化，减少代码体积。

摇树删除代码的原理： webpack基于ES6提供的模块系统，对代码的依赖树进行静态分析，把import & export标记为3类：

- 所有import标记为/* harmony import */
- 被使用过的export标记为/harmony export([type])/，其中[type]和webpack内部有关，可能是binding，immutable等；
- 没有被使用的export标记为/* unused harmony export [FuncName] */，其中[FuncName]为export的方法名，之后使用Uglifyjs（或者其他类似的工具）进行代码精简，把没用的都删除。

条件：

1. 首先源码必须遵循 ES6 的模块规范 (import & export)，如果是 CommonJS 规范 (require) 则无法使用。
2. 编写的模块代码不能有副作用，如果在代码内部改变了外部的变量则不会被移除。

使用方法：

可以在webpack中配置

```
  usedExports: true,
  minimize: true, // 自动压缩代码 负责摇掉枯树叶
```

这样在生产模式下会自动启动。

使用摇树的注意事项：

1. 使用 ES6 模块语法编写代码
2. 工具类函数尽量以单独的形式输出，不要集中成一个对象或者类
3. 声明 sideEffects
4. 自己在重构代码时也要注意副作用

39、用Vue3.0 写过组件吗？如果想实现一个Modal你会怎么设计?

https://vue3js.cn/interview/vue3/modal_component.html

40、实际工作中，你总结的vue最佳实践有哪些?

1. 编码风格方面：
    - 命名组件时使用“多词”风格避免和HTML元素冲突
    - 使用“细节化”方式定义属性而不是只有一个属性名
    - 属性名声明时使用“驼峰命名”，模板或jsx中使用“肉串命名”
    - 使用v-for时务必加上key，且不要跟v-if写在一起
2. 性能方面：
    - 路由懒加载减少应用尺寸
    - 利用SSR减少首屏加载时间
    - 利用v-once渲染那些不需要更新的内容
    - 一些长列表可以利用虚拟滚动技术避免内存过度占用
    - 对于深层嵌套对象的大数组可以使用shallowRef或shallowReactive降低开销
    - 避免不必要的组件抽象
3. 安全：

- 不使用不可信模板，例如使用用户输入拼接模板：`template: <div> + userProvidedString + </div>`
- 小心使用v-html，:url，:style等，避免html、url、样式等注入

41、vue路由的原理

路由是用来实现单页面应用的。

vue的路由的话主要是基于hash哈希路由和history api的路由。

hash路由其实就是锚点链接，哈希值的改变不会引起页面跳转。所以我们可以给 链接点添加哈希值，也就是写个井号，然后当哈希值改变的时候，通过readyStateChange 事件去监听哈希指的改变，然后根据哈希值去更细页面。

第二种是history路由。也就是通过 history的方法pushState去改变路 径。它也可以改变路径而且不会引起页面跳转。哈希路由会有404的问题。 需要服务器做重定向的处理。

42、说说Vuex的理解，或者说Vuex怎么 用的

vuex是一个专为vue.js应用程序开发的状态管理的库，vuex有五个核心属性：state，getter，mutation，action，module。

其中state是用来保存状态的，getter类似组件中的计算属性，可以实现对state的过滤。修改状态的话需要通过 提交 mutation来实现。action里异步操作，比如http请求就可以在action中做。还有一个是module，它是将store分成模块，每个模块都有自己的state、mutation、action，设置是嵌套子模块。

当数据修改时，通过dispatch来调用action，在action里再通过commit来调用mutation。当状态发生改变，那么所有引用这个状态的组件的视图都会发生更新。 但是，一般我们在比较复杂的情况下才回去用到vuex，如果只是简单的父 子组件 传值或者兄弟组件 传值，其实用props、自定义事件或者是event bus就足够了。

43、MVVM

MVVM其实表示的是 Model-View-ViewModel

Model：模型层，负责处理业务逻辑以及和服务器端进行交互 View：视图层：负责将数据模型转化为UI展示出来，可以简单的理解为 HTML页面 ViewModel：视图模型层，用来连接Model和View，是Model和View之 间的通信桥梁

在MVVM的架构下，View层和Model层并没有直接联系，而是通过ViewModel层进行交互。 ViewModel层通过双向数据绑定将View层和Model层连接了起来，使得View层和Model层的同步工作完全是自动的。因此只需关注业务逻辑，无需手动操作DOM。

vue其实就是个类似MVVM的框架。

其中template就是view视图层。 data或者vuex的store就是Model层。然后vue实例或者组件对象是VM层。

负责 View 和 Model 之间通信的桥梁。咱们在使用 Vue 框开发的时候，只需要关心 View 层的 视图层 代码和 数据层的 JavaScript 逻辑就可以了。

44、v-model的原理

v-model只不过是一个语法糖而已,真正的实现靠的还是

v-bind:绑定input的value属性来给他设置值 然后监听input事件或者change事件来获取输入框的值

45、有单独负责过项目吗，有没有自己搭 建过项目

搭建过。如果现在现在让我搭建的话，我会选择比较新的技术。 技术栈选择TS + Vue3 + vite + Pinia 来搭建。我大概说一下流程吧

首先使用vite生成项目，执行一个命令：

```
npm create vite@latest
```

然后选择vue、ts。

2、等项目生成好了之后，安装eslint,统一代码的风格。具体配置使用之前的配置文件拿来改一下或者去查看官方文档。

3、然后还会安装 prettier 进行代码的自动格式化

4、然后安装路由 vue-router,使用createRouter创建路由，在main.ts中引用。路由的配置这里，我会用 import 函数实现路由组件的动态加载。

5、接下安装状态管理。我会选择使用Pinia，用vuex也行。都差不多。安装好了之后在在根目录创建store文件夹，然后在里面创建store。

6、样式的话，我会选用scss

7、网络请求一般都会使用axios。我会对axios进行一些封装。一般是在根目录下创建个utils的目录，在里面去封装axios。主要是封装一些请求头，比如接口前缀，超时时间，拦截器等。 然后再创建个api的目录，用来封装接口。

8、组件库的话，如果是做后台管理系统，会使用Element-Plus，按照步骤就安装他官网的步骤去配置。 差不多就是这样。

46、eslint用过吗？eslint常用配置

eslint是一个代码规范的检测工具，使用eslint可以避免一些低级错误而且课统一代码规范。 安装好后，在项目的根目录下会有一个.eslintrc.js 的配置文件。在这个配 置文件中可以设置一些规则。

这些规则的等级有三种 off表示关闭，warn表示警告，error表示错误。 如果在代码中希望关闭eslint的检测，可以使用注释：

```
/* eslint-disable */
```

常见的eslint的配置项有

- "no-console": 2, //不允许出现console语句
- "no-debugger": //不允许出现debugger语句
- "no-empty": //不允许出现空的代码块
- "no-extra-semi": //不允许出现不必要的分号
- "no-obj-calls": //不允许把全局对象属性当做函数来调用
- "no-regex-spaces": //正则表达式中不允许出现多个连续空格
- "quote-props": //对象中的属性名是否需要用引号引起来
- "no-sparse-arrays": //数组中不允许出现空位置
- "no-unreachable": //在return，throw，continue，break语句 后不允许出现不可能到达的语句

https://blog.csdn.net/qq_33081841/article/details/88575729

官网配合文档：https://eslint.org/docs/latest/rules/

47、如何打包上线一个项目

打包的话，执行 npm run build命令就可以打包了，会生成一个dist目录。dist目录部署到服务器上就可以了。

服务器用的是nginx服务器。用ftp把本地的代码上传到nginx服务器的项目下，把nginx启动起来就行了。