2024年前端项目细节及疑难点

1. 测试和调试

H5 页面需要在多种设备和环境下进行测试，确保其稳定性和兼容性。

解决方法：

· 使用模拟器或真实设备进行测试。

· 利用浏览器的开发者工具进行调试。

· 采用自动化测试框架，如 Selenium 或 Appium ，进行持续集成和测试。

通过了解和解决这些常见的问题，开发者可以提高 H5 页面的质量和性能，为用户提供更好的体验。而

持续学习和适应新技术才是避免和解决这些问题的关键

2. 跨域请求问题

在进行 Ajax 请求或 API 调用时，可能会遇到跨域问题，导致请求失败。

解决方法：

· 使用 JSONP 来绕过同源策略限制。

· 在服务器端设置 CORS（跨源资源共享）策略，允许特定的外部域访问资源。

3. 数据安全和隐私保护

在处理用户数据时，开发者必须确保数据的安全性和用户的隐私权。

解决方法：

· 采用 HTTPS 协议来加密数据传输。

· 遵循数据最小化原则，只收集必要的用户信息。

· 提供清晰的隐私政策，告知用户数据的使用方式和目的

4. 移动端触摸事件处理

在移动端，触摸事件的处理与桌面端的鼠标事件有所不同，需要特别注意。

解决方法：

· 使用触摸事件（touchstart, touchmove, touchend）代替鼠标事件。

· 考虑使用框架或库，如 jQuery

5. 性能优化

H5 页面的性能直接影响用户体验。加载时间过长或运行缓慢的页面会导致用户流失。

解决方法：

· 对图片和视频进行压缩，减少资源文件的大小。

· 利用浏览器缓存和 CDN 服务来加速资源的加载。

· 优化 CSS 和 JavaScript 代码，避免重绘和重排，提高页面渲染效率。

6. 响应式布局的挑战

为了适应不同屏幕尺寸和分辨率，开发者需要创建响应式布局。这可能导致 CSS 和 JavaScript 代码复杂，难以维护。

解决方法：

· 使用媒体查询来适应不同的屏幕尺寸。

· 利用CSS框架，如 Bootstrap 或 Foundation ，它们提供了一套响应式网格系统，简化了布局的开发。

7. 浏览器兼容性问题

在开发 H5 页面时，开发者需要考虑到不同浏览器和设备的兼容性。一些特性在某些浏览器上可能不被

支持，或者在不同浏览器上表现不一致。

解决方法：

· 使用特性检测库，如 Modernizr ，来检测浏览器是否支持特定的功能。

· 采用渐进增强的策略，确保基础功能在所有浏览器上都能正常工作，高级功能则根据浏览器能力逐步增强

8. 获取首页链接里面的参数问题。

获取是可以获取到，只要不跳转出这个项目的页面，都是可以的，但是该项目链接了许多外链，所以，有时候返回的时候，页面就会显示空白，因为获取的参数出了问题。解决办法：将参数设置成了缓存，但是返回的速度快了，首页同样还是会出现拿不到参数，的问题。

解决方法：将获取连接的templateId写在Home页面

```
if (templateId) {
    this.$store.commit('updateTemplatedId',templateId);
    window.localStorage.setItem('templateId', templateId);
    console.log('store===templateId',this.$store.state.templateId)
}else{
    setTimeout(function() {
        window.location.reload()
    }, 1500);
}

```

9. 更新文件缓存的坑。

每次打包好文件给后端更新的时候，用户手机上总会留下，上次版本的信息，而且每次都得清下缓存，才会显示最新版本的数据。后来，我师傅提了个建议，让后端返回一个更新版本的接口，前端每次更新版本的时候，都会给后端传入时间戳，然后后端接收后判断和库里的时间戳是否相同，相同的返回不需要更新，不相同的话，返回要更新，然后前端这边的处理方法是：需要更新的话，清除掉缓存，刷新页面即可。

虽然说给.js .css文件后缀加上时间戳也是可以的，但是页面的入口index.html每次都是一样的，所以。。。就不会更新，，百度一些说在nginx服务器上，写上强制更新，但是由于公司服务器上的文件很多，万一操作失误那就麻烦了

10. 解析后端返回的map格式数据的坑。

之前解析数据的时候，直接就可以拿去，然后直接渲染页面使用即可。但是这次遇到后端返回的是map格式的数据，这就得解析下了。 例如：body['1'] 。根据返回的格式，自己解析成自己需要的数据格式。

11. vue组件动态加载的坑。

由于首页的排版不确定，然后想着，让组件动态显示，根据后端传入的数据，传入那个组件的数据，就显示那个组件。解决办法：和后端商量好，做个标识。前端根据标识判断，动态显示组件。 使用到了vue中的 ,刚开始想着是不是和原生js一样使用append直接可以插入进入呢，但是后来发现根本不可以，思路是可以的，但是实现起来是行不通的。因为append后面插入的必须是个节点，而不是组件。后来就去查阅vue文档。

12. 页面缓存的坑。

有个填写信息的页面，需要填写一部分信息，进入查新协议页面，返回的时候，页面上填写的信息还需要留存。 解决办法：使用vue提供的keep-alive，来完成页面的缓存的。

13. 路由传参的功能的坑。

之前一直使用路由传参，但是当本页面刷新的时候，页面上是没有参数的，因为参数是从上个页面传入进来的。 解决办法：使用了缓存，和vuex状态管理。但是由于项目并不是很大型的项目，所以使用最多的是缓存。

14. axios请求中post请求的坑。

刚开始的坑是，使用axios的post方法请求数据，数据被拦截，数据一直传不到后端那边。后来查文档才得知 axios对于post请求是有拦截功能的，需要自己判断，或者使用提请的 qs 方法，将传给后端的数据进行下处理。

15. 为什么需要二次封装组件库?

> 实际工作中，我们在项目中需要自定义`主题色`， `更改按钮样式`， `自定义图标`,自定义`table组件`等等，这些都可以基于antd组件库进行二次封装，减少重复工作，提升开发效率。

所以我们在封装的时候按照下面这四个原则进行思考就行了，另外本身封装组件库对于项目来说也是没有任何风险，因为一开始我们把`PropsType`直接进行`转发`，内部再进行增加业务的功能，这样就是达到完全的`解耦`

- **统一风格**：在一个大的项目或者多个相关的项目中，保持一致的界面风格和交互方式是非常重要的。通过二次封装，我们可以定义统一的样式和行为，减少不一致性。
- **降低维护成本**：当底层的组件库更新时，我们可能需要在项目的多个地方进行修改。但是如果我们有了自己的封装，只需要在封装层面进行更新即可，这大大降低了维护成本。
- **增加定制功能**：有些时候，我们需要在原有组件库的基础上增加一些特定的功能，如特定的验证、错误处理等。二次封装提供了这样的可能。
- **提高开发效率**：在一些常用的功能（如表单验证、全局提示等）上，二次封装可以提供更方便的API，提高开发效率。

16. 请结合一个组件库设计的过程,谈谈前端工程化的思想

当我们结合一个组件库设计的过程来谈论前端工程化的思想时，需要理清这些要点：

1. **使用 Lerna 进行多包管理：通过 Lerna 来管理多个包（组件），实现组件级别的解耦、独立版本控制、按需加载等特性。**

```
# 安装 Lerna
npm install -g lerna

# 初始化一个 Lerna 仓库
lerna init

# 创建 "Button" 组件包
lerna create button --yes
```

2. **规范化提交：使用规范化的提交信息可以提高 Git 日志的可读性，并且可以通过 conventional commits 自动生成 CHANGELOG。可以使用 commitizen、commitlint 等工具来配置。**

```
# 安装相关工具
npm install commitizen cz-conventional-changelog --save-dev
```

```
// package.json
{
  "scripts": {
    "commit": "git-cz"
  },
  "config": {
    "commitizen": {
      "path": "cz-conventional-changelog"
    }
  }
}

```

3. **代码规范化：通过 ESLint、Prettier 等工具实现代码规范化和格式化，并封装为自己的规范预设。**
4. **组件开发调试：需要考虑热更新编译、软链接引用等问题，以方便在开发过程中进行组件的调试。**
5. **文档站点：可以基于 dumi 搭建文档站点，并实现 CDN 加速、增量发布等优化。可以使用 surge 实现 PR 预览。**
6. **单元测试：需要考虑 jest、enzyme 等工具的配合使用，生成测试覆盖率报告。**

7. **按需加载：需要配合 babel-plugin-import 实现按需加载，即在编译时修改导入路径来实现组件的按需加载。**
8. **组件设计：需要考虑响应式、主题、国际化、TypeScript 支持等问题，以保证组件的灵活性和可扩展性。**
9. **发布前的自动化脚本：需要编写自动化脚本来规范发布流程，确保发布的一致性和可靠性。**
10. **发布后的处理：考虑补丁升级、文档站点同步发布等问题，以便及时修复问题并提供最新的文档。**
11. **制定 Contributing 文档：制定 Contributing 文档可以降低开源社区贡献的门槛，并确保社区成员了解如何参与项目。处理 issues 和 PR 需要有专人负责。**

17. 如何对一个组件库进行测试？

> 首先需要明确，组件库的测试大致可以分为两类：一类是针对组件本身的功能和性能的测试（例如，单元测试、性能测试），另一类是针对组件在集成环境下的行为和性能的测试（例如，集成测试、系统测试）。

1. **功能测试（单元测试）**

通常来说，组件的功能测试可以通过单元测试来完成。单元测试的目的是验证组件的单个功能是否按照预期工作。这通常可以通过编写测试用例来完成，每个测试用例针对一个特定的功能。

2. **边界测试**

边界测试是一种特殊的功能测试，用于检查组件在输入或输出达到极限或边界条件时的行为。

3. **响应测试**

响应测试通常涉及到 UI 组件在不同的设备或屏幕尺寸下的行为。这可能需要使用端到端（E2E）测试工具，如 Puppeteer、Cypress 等。

4. **交互测试**

交互测试也可以通过端到端（E2E）测试工具来完成。

5. **异常测试**

异常测试用于验证组件在遇到错误或非法输入时能否正确处理。这通常可以通过在测试用例中模拟错误条件来完成。

6. **性能测试**

性能测试用于验证组件的性能，例如，加载速度、内存消耗等。

7. **自动化测试**

单元测试、集成测试和系统测试都可以通过自动化测试工具进行。例如，Jest 和 Mocha 可以用于自动化运行 JavaScript 单元测试，Puppeteer 和 Selenium 可以用于自动化运行端到端测试。

18. 在使用 `qiankun` 时，你如何处理多个子项目的调试问题?

在使用 `qiankun` 处理多个子项目的调试问题时，通常的方式是将每个子项目作为一个独立的应用进行开发和调试。每个子项目都可以在本地启动，并通过修改主应用的配置，让主应用去加载本地正在运行的子应用，这样就可以对子应用进行调试了。这种方式的好处是，子应用与主应用解耦，可以独立进行开发和调试，不会相互影响。

对于如何同时启动多个子应用，你可以使用 `npm-run-all` 这个工具。 `npm-run-all` 是一个CLI工具，可以并行或者串行执行多个npm脚本。这个工具对于同时启动多个子应用非常有用。

19. 你能解释一下 `qiankun` 如何实现 `keep-alive` 的需求吗?

> 在 `qiankun` 中，实现 `keep-alive` 的需求有一定的挑战性。这是因为 `qiankun` 的设计理念是在子应用卸载时，将环境还原到子应用加载前的状态，以防止子应用对全局环境造成污染。这种设计理念与 `keep-alive` 的需求是相悖的，因为 `keep-alive` 需要保留子应用的状态，而不是在子应用卸载时将其状态清除。

然而，我们可以通过一些技巧来实现 `keep-alive` 的效果。一种可能的方法是在子应用的生命周期函数中保存和恢复子应用的状态。

20. 在使用 `qiankun` 时，你如何处理 `js` 沙箱不能解决的 `js` 污染问题?

> `qiankun` 的 `js` 沙箱机制主要是通过代理 `window` 对象来实现的，它可以有效地隔离子应用的全局变量，防止子应用之间的全局变量污染。然而，这种机制并不能解决所有的 `js` 污染问题。例如，如果我们使用 `onclick` 或 `addEventListener` 给 `<body>` 添加了一个点击事件，`js` 沙箱并不能消除它的影响。

对于这种情况，我们需要依赖于良好的代码规范和开发者的自觉。在开发子应用时，我们需要避免直接操作全局对象，如 `window` 和 `document` 。如果必须要操作，我们应该在子应用卸载时，清理掉这些全局事件和全局变量，以防止对其他子应用或主应用造成影响。

21. 在项目中有没有单独封装组件

答：有的，在項目的 commen 文件下会存放项目公用组件（如：页面的头组件、页面底部组件等）项目里的feature 文件下则是放项目的功能组件（如：轮播图、分页器、上拉加载这些功能组件）把这些页面重复的部分，抽离出来进行单独的封装，有效的减少了代码量，提升了项目的开发效率。解决了传统项目中效率低、难维护、复用性低的等问题。

22. 在项目中发送请求怎么携带token

答：当用户登陆的时候，后端会把用户的信息和token返回给我们的，我们进行存储token，把存储的token，放在请求拦截器中，这样所有的请求都可以直接通过请求拦截器将token传给服务器。

23. 工作中有用到git吗

答：有的，在之前的公司里，基本上用的都是git进行多人开发，git是一个分布式版本管理工具，首页是获取项目：访问远程仓库复制项目链接，下载项目到本地。使用 git init 方法在项目根目录下，创建本地 git 仓库，进行一个开发，开发完毕后使用git add.方法将文件都加载到暂存区，再使用 git commit -m 提交代码到本地仓库，先pull远程仓库的代码（避免远程仓库与本地代码不一致时发生冲突），最后push 本地仓库的代码提交到远程仓库。还可以使用 git merge 将子分支代码合并到主分支代码仓库里。

24. git 版本发生冲突你怎么解决的

版本冲突基本出现在合并操作（合并远程仓库代码或者合并分支代码）中。如果出现版本冲突，需要具体分析出现冲突的代码区，手动进行代码合并，然后再进行提交。

25. 如何快速让一个盒子水平垂直居中

纯flex布局实现：：给元素设置flex布局，添加flex的alignltems:center垂直居中,justifycontent:center水平居中，实现居中效果；

flex布局+外边距（margin）实现：给父级设置为display: flex；布局，/给子级添加 margin:auto；

就可以实现水平垂直居中显示；

绝对定位+外边距（margin）实现：：绝对定位的元素将上下左右都设置为O，再设置margin:auto 即可实现居中；

绝对定位+外边距（margin）+C3的转换transform 中的translate实现：：绝对定位元素，利用 margin 偏移外容器的50%，再利用translate 回补自身宽高的50%；

26. 首屏加载白屏怎么进行优化

答：

（1）使用CDN 减小代码体积，加快请求速度；

(2)SSR通过服务端把所有数据全部渲染完成再返回给客户端；

(3) 路由懒加载，当用户访问的时候，再加载相应模块；

(4) 使用外链CSS，JS文件；

(5) 开启 GZIP压缩；（是一种 http 请求优化方式，通过减少文件体积来提高加载速度。

html、js、css 文件甚至json数据都可以用它压缩，可以减小60%以上的体积。（需要后端支持））

(6) 项目打包不生成.map 文件；

(7)使用骨架屏 在首页加载时，显示页面结构轮廓；

27. 路由传参 query与 params区别

答：query类似 get，跳转之后在地址栏中显示拼接参数，类似？id=1；params类似 post，跳转之后不在 url 中显示参数；相对于query比较安全；

注意点：（params 在进行路由传参时，要在路由中配置占位。如果不在路由中配置占位，跳转过去以后刷新页面会数据丢失，query参数数据不会丢失）；

28. 项目基础优化

(1)减少 HTTP 请求数（避免重复请求）

(2)减少 DOM元素和 DOM 操作

(3)使用雪碧图

(4)压缩合并 JS、CSS代码

(5)避免图片 src为空

(6)把样式表放在 link 中

(7)css样式放置在文件头部

(8)js脚本放置在文件末尾

29. 对 $nextTick异步渲染的理解

答：Vue 采用了数据驱动视图的思想，但是在一些情况下，仍然需要操作DOM。有时候，DOM1的数据发生了变化，而DOM2需要从DOM1中获取数据，那这时就会发现DOM2的视图并没有更新，这时就需要用到了nextTick

了；

作用：在下一次 DOM更新结束后执行其指定的回调；（当数据更新后，要基于更新后的新DOM进行某些操作时，要在nextTick指定的回调函数中执行）。如果不采用异步更新，在每次数据的更新后，都会对当前组件进行重新渲染。所以为了性能考虑，vue 会在本轮数据更新后，再去异步更新视图；

30. 你是怎么对axios进行二次封装的，有什么作用

答：

1.添加请求拦截器

在请求发送前进行必要操作处理，例如添加tokan、cookie 请求体加验证、设置请求头等，相当于是对每个接口里相同操作的一个封装；

2.添加响应拦截器

同理，响应拦截器也是如此功能，只是在请求得到响应之后，对响应体的一些处理，通常是数据统一处理等，也常来判断登录失效等。

31. 用户token 失效你是怎么处理的

答：当用户登陆的时候，接口的响应信息中是有三个和token相关的信息的，1.当前使用的token，用于访问需要授权的接口 2.字段就是token的过期时间3.刷新获取新的token字段；

方式一：在请求拦截器，根据接口响应的过期时间判断token是否过期，如果过期则会刷新后再继续请求接口；缺点：1.后端需要提供Token

过期时间字段，2.需要过期时间与本地时间进行判断，如果计算机时间被篡改时，拦截就会失败的；

方式二：在每次请求数据的时候都会携带

token，正常情况下接口返回的状态码是200，我们会在响应拦截器中，走成功的回调把接口中data数据进行返回，如果tokan失效后端会返回401的状态码，就表示没有tokan/tokam无效或者过期；

具体处理步骤：在响应拦截器的失败回调中，if判断 error.response状态码，等于401就代表着token无效/过期，那么我们会if判断 vuex的sdata.user 信息，如果说没有就跳转到登陆页面（可以把router.push（）封装为跳转函数，进行调用），有的话，就响应拦截器外定义一个变量标记Token的刷新状态（避免多次请求刷新Token） 默认为false，和一个数组用来存储因为token刷新而挂起的请求，在有user信息的逻辑基础上，去if判断这个变量，为true表示正在进行刷新token的请求，另那么我们就return；

push函数，把失败请求存储到列表中去;else就是变量为false，先设置变量true，然后获取到用户信息中的续期 token的字段;作为参数发送获取新的token请求(实现用户无感知刷新tokan)，使用if判断code字段!==200，失败就使用vuex的commit方法在 mutation 中清除无效的用户信息，然后调用跳转函数跳转到登陆页面去;code字段==200，成功就使用vuex的commit方法在mutation 中存储新的token,然后重新发送失败的请求(使用forEach遍历请求存储列表，进行自调用)，然后就清空数组，将请求return 出去request(error.config);最后无论请求成功失败就把这个变量字段设置false;

32. 说一下项目的流程

答：

1.项目立项

2.BA整理项目需求，定项目周期；

3. 需求评审（过项目需求）

4.后端设计数据库，ui设计原型图

5.ui评审

6. 前端静态页面开发（mock数据测试），后端做接口，测试做测试用例；

7. 前后端联调

8.测试

9.项目上线

33. 商品和增删改查

答：根据接口文档获取到已有的商品列表信息（商品名称、商品价格、商品信息），进行列表渲染展示，可以对商品进行增加、删除、修改。

添加：根据接口文档，发送添加请求，判断接口返回结果中 code 字段的 ==200就使用

Element-UI 的message消息提示，添加成功，关闭遮罩层为false，调用请求数据函数更新列表；else 就抛出一个错误信息；

修改：当点击修改时，弹出 Element-UI遮罩层收集用户修改数据，这里进行了一个lodash浅拷贝赋值操作（因为数据在页面上正在展示，直接修改数据会导致页面数据发生变化），点击确定的时候会，就把用户信息作为参数，发送修改请求，判断接口返回结果中code字段的 ==200 就使用 Element-UI 的message消息提示，修改成功，关闭遮罩层为false，调用请求数据函数更新列表；else 就抛出一个错误；

删除：在删除的回调中可以拿到当前商品的id，那么就把id作为参数发送删除请求，判断接口返回结果中code 字段的== 200就使用Element-UI 的message消息提示，删除成功，调用请求数据函数来更新列表；

34. Element-UI 的form表单验证你是怎么使用的

答：首先是在form表单配置 rules 校验规则，可以在组件data中自定义（或者统一封装到一个Js文件中），使用required: true，来设置form表单为必填项，message设置提示信息，通过trigger 定义触发校验方式；当全部表单都符合校验规则，在点击确定的时候调用 validate进行最终的表单校验。validate 参数为一个回调函数，回调函数有一个 config参数（是否校验成功）我们可以对config进行判断，成功就携带form表单的信息提交后台；

35. 项目中的商品详情页你是怎么实现的

答：当用户点击商品名字或者图片的时候，就进行this.$router.push 进行路由跳转并传递商品的id 到详情页，再通过后台提供的相关详情接口，把id 作为参数，在详情组件的 mounted钩子函数中派发

action 调用Vuex中的请求函数，获取到商品的详细数据存储Vuex中，通过getters 配置对象，对数据进行简化，通过mapgetters 映射给组件，使用插值语法或者v-for进行数据的渲染；

36. 什么是数组扁平化

答：所谓的数组扁平化就是将多维数组转化为一维数组；

一般数组扁平化，数组中存储的多维数据都是数组 不会是对象或者函数；

具体实现：最常用的方法就是使用toString（）方法将数组中的每一个数据，转成一个字符串，使用逗号进行间隔，再使用字符串中的split方法，以逗号进行分割将字符串再转化为一维数组；

37. 你这个项目登陆是怎么实现的

答：登陆的方式有很多种，我们这个选择两种方式进行回答。

第一种：手机号+验证码实现登陆

（1）在用户选择登陆的时候，会收集用户输入的信息，进行一个form的校验(正则或者Element-U方式，校验字段是否合法，不合法就给予错误提示)

（2）当用户点击获取验证码按钮的时候，发送获取验证码请求，后端会发验证码至用户手机，并对验证码设置一个过期时间，这时把按钮disabled设置为true变为禁用状态，使用setinterval定时器设置一个倒计时的效果，把按钮文本内容与倒计时进行拼接显示，判断这个倒计时等于0就清除这个定时器clearlnterval,解除按钮的禁用状态disabled设置为false,按钮文本恢复为获取验证码，计时变量再设置为60秒；用户收到验证码后进行输入。

(3)在点击登录的回调中，为了防止在输入验证码后手机号更换了或者删除了，发送无效请求，所以这里还要对手机号进行校验，不合法就提示用户输入正确手机号，然后return退出，合法就发送登陆请求，if判断返回请求结果成功就$router.push()跳转到首页；else登陆失败，就直接return一个Promise.reject()new一个error错误进行抛出；

第二种：账户+密码实现登陆

(1)收集数据，进行form表单验证与上面相同；

(2)通过监听属性调用后台获取公钥接口，(该接口中后台会通过算法生成一对钥匙公钥、私钥，并存到session中，公钥则返回给前端)前端拿到公钥后，再通过JS加密库中的方法对用户输入的密码进行加密。

(3)点击登录时，发送登录请求，传递加密好的密文；登录成功后，就提示用户登陆成功，本地存储token令牌，使用$router.push()跳转至首页；

38. 项目中的权限管理怎么实现的

答：权限管理有三个很重要的模块；

(1)用户模块：可以给用户分配不同的角色

(2)角色模块：可以授于用户不同的角色，不同的角色有不同权限

(3)权限模块：用于管理系统中的权限接口，为角色提供对应的权限

具体实现步骤

根据用户管理中给用户分配角色的不同，就有对应的菜单权限。

(1)当用户登录后，后端会把用户token和相应的用户信息(用户名/菜单权限),返回给前端，前端会先把用户信息先进行保存。

(2)在routes.js路由文件中把路由拆分为静态路由和动态路由，在vuex的user.js模块中，通过数组的filter方法和indexOf定义一个递归函数，在请求用户信息的时候，再进行commit提交mutations一个计算函数同时调用递归函数传递参数(参数一是异步路由，参数二是用户的权限信息)进行递归过滤，留下当前用户有权限访问的路由。

(3)在mutations中，通过concat方法合并静态路由和过滤后的动态路由，使用路由的addRouter方法注册为新的路由，实现根据不同的用户展示不同的菜单侧边栏； (后端也会进行一些权限接口的判断和拦截，通过非法途径是访问不了的)

39. Vue的$nextTick()方法有用到过吗

答：有，比如说数据渲染更新后，我们要更后的数据进行某些操作时，就可以在nextTick指定的回调函数中去执行；

40. 深拷贝与浅拷贝

浅拷贝：只是复制某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存；深拷贝：会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会影响到原对象；

实现浅拷贝方式：1.0bject.assign;2.es6扩展运算符

实现深拷贝方式：1.JSON.parse(JSON.stringify(object));2.lodash插件；

实际工作中：(1)当数据类型比较简单时(对象/数组)的时候，可以用浅拷贝；(2)当数据结构中存在对象里面套数组，数组里面套对象(结构较为复杂),需要使用深拷贝来解决这类问题；

lodash插件；

实际工作中：(1)当数据类型比较简单时(对象/数组)的时候，可以用浅拷贝；(2)当数据结构中存在对象里面套数组，数组里面套对象(结构较为复杂),需要使用深拷贝来解决这类问题；

41. 在Vue是项目中如何打开新的页面

答：使用this.$router.resolve()和window.open(xxx.href,'_blank')在进行路由跳转的时候就会打开新的窗口。但是这种方式有个问题就是不能用params方式进行传递参数，在页面跳转过去后，获取的params方式传递参数为undefined。

解决办法：使用query方式传参，本地存储或者Vuex;

42. 请解释一下微前端的概念以及它的主要优点和挑战?

微前端是一种将不同的前端应用组合到一起的架构模式。这些应用可以独立开发、独立部署、独立运行，然后在一个主应用中进行集成。这种模式的主要目标是解决大型、长期演进的前端项目的复杂性问题。

主要优点：

1. **解耦：** 微前端架构可以将大型项目分解为多个可以独立开发、测试和部署的小型应用。这种解耦可以提高开发效率，减少团队间的协调成本。
2. **技术栈无关：** 不同的微前端应用可以使用不同的技术栈，这为使用新技术、升级旧技术提供了可能。
3. **并行开发：** 因为微前端应用是独立的，所以多个团队可以并行开发不同的应用，无需担心相互影响。
4. **独立部署：** 每个微前端应用可以独立部署，这意味着可以更快地推出新功能，同时降低了部署失败的风险。

主要挑战：

1. **性能问题：** 如果不同的微前端应用使用了不同的库或框架，可能会导致加载和运行的性能问题。
2. **一致性：** 保持不同的微前端应用在用户体验、设计和行为上的一致性可能会比较困难。
3. **状态共享：** 在微前端应用之间共享状态可能会比较复杂，需要使用特殊的工具或模式。
4. **复杂性：** 尽管微前端可以解决大型项目的复杂性问题，但是它自身也带来了一些复杂性，比如需要管理和协调多个独立的应用。
5. **安全性：** 微前端架构可能会增加跨域等安全问题。

43. 你能详细描述一下 qiankun 微前端框架的工作原理吗?

> qiankun 是一个基于 single-spa 的微前端实现框架。它的工作原理主要涉及到以下几个方面：

1. **应用加载**：qiankun 通过动态创建 script 标签的方式加载子应用的入口文件。加载完成后，会执行子应用暴露出的生命周期函数。

2. **生命周期管理**：qiankun 要求每个子应用都需要暴露出 bootstrap、mount 和 unmount 三个生命周期函数。bootstrap 函数在应用加载时被调用，mount 函数在应用启动时被调用，unmount 函数在应用卸载时被调用。
3. **沙箱隔离**：qiankun 通过 Proxy 对象创建了一个 JavaScript 沙箱，用于隔离子应用的全局变量，防止子应用之间的全局变量污染。
4. **样式隔离**：qiankun 通过动态添加和移除样式标签的方式实现了样式隔离。当子应用启动时，会动态添加子应用的样式标签，当子应用卸载时，会移除子应用的样式标签。
5. **通信机制**：qiankun 提供了一个全局的通信机制，允许子应用之间进行通信。

44. 在使用 qiankun 时，如果子应用是基于 jQuery 的多页应用，你会如何处理静态资源的加载问题？

> 在使用 qiankun 时，如果子应用是基于 jQuery 的多页应用，静态资源的加载问题可能会成为一个挑战。这是因为在微前端环境中，子应用的静态资源路径可能需要进行特殊处理才能正确加载。这里有几种可能的解决方案：

方案一：使用公共路径

在子应用的静态资源路径前添加公共路径前缀。例如，如果子应用的静态资源存放在 `http://localhost:8080/static/`，那么可以在所有的静态资源路径前添加这个前缀。

方案二：劫持标签插入函数

这个方案分为两步：

1. 对于 HTML 中已有的 img/audio/video 等标签，qiankun 支持重写 getTemplate 函数，可以将入口文件 index.html 中的静态资源路径替换掉。
2. 对于动态插入的 img/audio/video 等标签，劫持 appendChild、innerHTML、insertBefore 等事件，将资源的相对路径替换成绝对路径。

45. 在使用 `qiankun` 时，如果子应用动态插入了一些标签，你会如何处理？

在使用 `qiankun` 时，如果子应用动态插入了一些标签，我们可以通过劫持 DOM 的一些方法来处理。例如，我们可以劫持 `appendChild`、`innerHTML` 和 `insertBefore` 等方法，将资源的相对路径替换为绝对路径。

46. 在使用 `qiankun` 时，你如何处理老项目的资源加载问题？你能给出一些具体的解决方案吗？

> 在使用 `qiankun` 时，处理老项目的资源加载问题可以有多种方案，具体的选择取决于项目的具体情况。以下是一些可能的解决方案：

1. **使用 `qiankun` 的 `getTemplate` 函数重写静态资源路径**：对于 HTML 中已有的 `img/audio/video` 等标签，`qiankun` 支持重写 `getTemplate` 函数，可以将入口文件 `index.html` 中的静态资源路径替换掉。
2. **劫持标签插入函数**：对于动态插入的 `img/audio/video` 等标签，我们可以劫持 `appendChild` 、 `innerHTML` 、 `insertBefore` 等事件，将资源的相对路径替换成绝对路径。例如，我们可以劫持 jQuery 的 `html` 方法，将图片的相对路径替换为绝对路径：
3. **给老项目加上 webpack 打包**：这个方案的可行性不高，都是陈年老项目了，没必要这样折腾。
4. **使用 iframe 嵌入老项目**：虽然 `qiankun` 支持 jQuery 老项目，但是似乎对多页应用没有很好的解决办法。每个页面都去修改，成本很大也很麻烦，但是使用 iframe 嵌入这些老项目就比较方便。

47. 你能解释一下 `qiankun` 的 `start` 函数的作用和参数吗？如果只有一个子项目，你会如何启用预加载？

> `qiankun` 的 `start` 函数是用来启动微前端应用的。在注册完所有的子应用之后，我们需要调用 `start` 函数来启动微前端应用。

`start` 函数接收一个可选的配置对象作为参数，这个对象可以包含以下属性：

- `prefetch`：预加载模式，可选值有 `true`、`false`、`'all'`、`'popstate'`。默认值为 `true`，即在主应用 `start` 之后即刻开始预加载所有子应用的静态资源。如果设置为 `'all'`，则主应用 `start` 之后会预加载所有子应用静态资源，无论子应用是否激活。如果设置为 `'popstate'`，则只有在路由切换的时候才会去预加载对应子应用的静态资源。
- `sandbox`：沙箱模式，可选值有 `true`、`false`、`{ strictStyleIsolation: true }`。默认值为 `true`，即为每个子应用创建一个新的沙箱环境。如果设置为 `false`，则子应用运行在当前环境下，没有任何的隔离。如果设置为 `{ strictStyleIsolation: true }`，则会启用严格的样式隔离模式，即子应用的样式会被完全隔离，不会影响到其他子应用和主应用。
- `singular`：是否为单例模式，可选值有 `true`、`false`。默认值为 `true`，即一次只能有一个子应用处于激活状态。如果设置为 `false`，则可以同时激活多个子应用。
- `fetch`：自定义的 `fetch` 方法，用于加载子应用的静态资源。

48. axios-请求中post请求的坑

刚开始的坑是，使用axios的post方法请求数据，数据被拦截，数据一直传不到后端那边。后来查文档才得知 axios对于post请求是有拦截功能的，需要自己判断，或者使用提请的 qs 方法，将传给后端的数据进行下处理。

49. 路由传参的功能的坑

之前一直使用路由传参，但是当本页面刷新的时候，页面上是没有参数的，因为参数是从上个页面传入进来的。 解决办法：使用了缓存，和vuex状态管理。但是由于项目并不是很大型的项目，所以使用最多的是缓存。

50. 页面缓存的坑

有个填写信息的页面，需要填写一部分信息，进入查新协议页面，返回的时候，页面上填写的信息还需要留存。 解决办法：使用vue提供的keep-alive，来完成页面的缓存的。

51. vue组件动态加载的坑

由于首页的排版不确定，然后想着，让组件动态显示，根据后端传入的数据，传入那个组件的数据，就显示那个组件。解决办法：和后端商量好，做个标识。前端根据标识判断，动态显示组件。 使用到了vue中的<component :is=""></component> ，刚开始想着是不是和原生js一样使用append直接可以插入进入呢，但是后来发现根本不可以，思路是可以的，但是实现起来是行不通的。因为append后面插入的必须是个节点，而不是组件。后来就去查阅vue文档。

但是现在还有个问题，首页是通过动态组件添加的，数据得从后端接口返回，但是接口请求也是需要时间，所以，刚开始进入页面的是，页面先会是空白，但是这样的体验并不友好，会让用户感觉到页面就是一片空白，但是好的解决办法现在暂时没有想出来。好的解决办法还在寻找ing。

52. 解析后端返回的map格式数据的坑

之前解析数据的时候，直接就可以拿去，然后直接渲染页面使用即可。但是这次遇到后端返回的是map格式的数据，这就得解析下了。 例如：body['1'] 。根据返回的格式，自己解析成自己需要的数据格式。

53. 更新文件缓存的坑

每次打包好文件给后端更新的时候，用户手机上总会留下，上次版本的信息，而且每次都得清下缓存，才会显示最新版本的数据。后来，我师傅提了个建议，让后端返回一个更新版本的接口，前端每次更新版本的时候，都会给后端传入时间戳，然后后端接收后判断和库里的时间戳是否相同，相同的返回不需要更新，不相同的话，返回要更新，然后前端这边的处理方法是：需要更新的话，清除掉缓存，刷新页面即可。

虽然说给.js .css文件后缀加上时间戳也是可以的，但是页面的入口index.html每次都是一样的，所以。。。就不会更新，百度一些说在nginx服务器上，写上强制更新，但是由于公司服务器上的文件很多，万一操作失误那就麻烦了。

54. h5页面打开调试日志

h5页面不像小程序那样，直接可以打开控制台，在手机上查看日志，得需要自己安装vConsole的插件来实现。

55. 获取首页链接里面的参数问题

获取是可以获取到，只要不跳转出这个项目的页面，都是可以的，但是该项目链接了许多外链，所以，有时候返回的时候，页面就会显示空白，因为获取的参数出了问题。解决办法：将参数设置成了缓存，但是返回的速度快了，首页同样还是会出现拿不到参数的问题。

解决办法还在寻找ing。

56. h5里面的搜索

h5里面input实现在手机上按下“搜索”，“go”，“前往”等按钮的时候，同时会触发像PC端的Enter。 input标签需要设置属性：type="search" 。

57. 登录接口bug

需要判断 errCode 10001状态的情况。如果出现 errCode出现 10001，则清空原来的session，重新请求该网络请求。

58. 封装的请求方法不需要在传入相同的参数。

封装的方法，每个方法里面都得传入sessoin，但是里面需要有个版本的方法不需要传入session，那么就得在封装的方法里面进行判断。

```
(config.method == 'get') {
    console.log('config.params.version',config.params.version)
    if(config.params.version == 'v'){   // 在更新版本的接口里面会用到
      config.url = config.url
    }else{
      config.url = config.url + '?session='+localStorage.getItem('session');
    }
  }
```

省份地区判断的话，尽量不使用name判断，会有bug的。通过市区匹配省份的话，使用areaCode，有些文件是不一样的。

59. 进来不在index.html文件里面引入公共的文件

因为每次更新版本的时候，index.html 都是相同的，如果修改了公共文件，是会有缓存，不会更新。因为该公共文件后面没有添加时间戳。

60. 你做的项目中都使用过那些中间件呢?

中间件可以理解为在请求和响应之间进行响应数据的处理。

分类： 内置中间件

app.use(express.static('托管目录地址'))

静态资源管理中间件

第三方中间件 body-parser// cookie-parser// cookie-seesion Ø

自定义中间件 记录访问日志

61. 你在开发过程中有什么困难点(或者使用了什么技术)

在开发的过程中我没有太多的问题但是我的同事遇到了一个问题，怎么在序列化器中获取request的值。通过源码的研究发现self.context["reqeust"].xx 可以获取到值 大大提高了数据的粘性和开发效率 在小程序端 使用模块的引用设置url值 方便url路径的更改

62. 会写接口吗？ 项目你负责什么？

会写drf源码 在小程序的项目中我通过自定义方法 重构drf内部方法 实现更高的扩展性。

开始：小程序、api（主）

63. 你之前做过小程序吗？主要包括哪些功能？

没做过小程序，这次公司需求。

了解vue.js ，类似。

小程序就是前端技术：html、css、js

64. 这个小程序的具体设计逻辑是怎么样的？有几个人开发？周期多长？在你离职时， 这个小程序项目是否还在开发？或者你离 职时，你们公司在开发什么？

他分为两大模块，有用户动态模块，拍卖模块，用户动态模块又分为发布功能，动态展示功能以及常见的用户交互功能，拍卖模块又分为专场-拍品，由后台管理人员维护更新，连我总共有3个人开发，总共开发了三个月，我离职时，项目还在开发，正在开发xx

65. 你说你负责支付环节，那么微信支付的 流程是否可以简单说一下？

- 临时凭证 获取 wxid
- 生成repayid
- json，返回给小程序，小程序调支付窗口。
- 支持成功提示POST回调。

66. 那好，既然流程说完了，那么，我问你， 你这个项目涉及到了rmb，它在用户点击 支付之后的逻辑关系和表关系你是怎么 理解的？ 有多少张表？表关系？

首先，在做这个功能的时候，会先在纸上画出来其中的逻辑，然后再把后端需要的值传过去。做这个后端api时，需要拿到前端的数据，判断它是否使用优惠券了，是否使用保证金了，是否有地址了，选择什么方式支付了，余额还是微信支付，还有该用户的支付价格，传到后端之后进行数据校验，首先判断地址是否存在，然后判断订单是否合法，是否使用了优惠券，是否使用保证金，用户的支付价格和后端计算的价格是否相符，由于支付环节必须保证一次完成，所以给它加了事务，订单表的查询加了锁，通过一系列校验，然后进行支付。支付成功之后，订单表记录，保证金抵扣记录，优惠券使用记录，退保证金记录，都需要进行相应的修改

67. 你觉得这个小程序项目的细节之处有哪些?

- 闭包
- drf，认证组件自定义。
- drf序列化 嵌套
- 重复订单处理 & 数据库锁（InnoDB 行级锁）

68. 如何自定义tabbar?

创建一个compent文件夹，里面创建一个tabbar页面，取消原有的tabbar页面。需要在app.json中输入compent:ture，在自定义的pages页面中写入<tabbar selected="{{0}}"></tabbar>进行选中效果显示

69. 你刚才提到了异步，在这个项目中你是否被异步坑过，最后又是如何解决的?

算是坑过吧，我在做发布功能时，需要将用户选中的图片和其他信息发送上去，图片发送到桶中，其他信息需要放到数据库中，而其他信息需要包含图片的路径地址，这个路径地址需要上传到桶中返回，而异步执行会造成图片上传和其他信息发送时间不一致，造成的结果是图片路径不能保存到数据库中。

70. 你能描述一下渐进增强和优雅降级之间的不同吗?

渐进增强：主要是针对低版本浏览器进行页面构建，保证最基本的功能，然后再针对高级浏览器进行效果或交互功能的改进来达到更好的用户体验。

优雅降级：主要是针对高级浏览器构建完整的功能，并视图降低用户的体验。然后再考虑低版本的兼容问题。

71. 为什么利用多个域名来存储网站资源会更有效?

1，节省主域名的连接数，使页面响应速度提高。

2，由于同一时刻对同一域名的请求数有一定限制，使用多个域名就可以突破浏览器的并发限制。

3，静态和动态请求分别放在不同的服务器上更加方便CDN缓存

4，避免不必要的安全问题(上传js窃取主站cookie之类的)

72. 一个页面上有大量的图片（大型电商网站），加载很慢，你有哪些方法优化这些图片的加载，给用户更好的体验。

我对不同的情况进行不同的处理。

1，如果一个页面图片很多，并超出了浏览器窗口的可视区域，我会采用图片懒加载。

图片**懒加载**，在页面上的未可视区域可以添加一个滚动条事件，判断图片位置与浏览器顶

端的距离与页面的距离，如果前者小于后者，优先加载。

2，如果为幻灯片、相册等，可以使用图片**预加载**技术，将当前展示图片的前一张和后一张优先下载

3， 果图片为 css 图片，比如一些小图片，logo，二维码可以整合到一起，使用**精灵图**技术。

4，如果图片过大，可以使用特殊编码的图片，**加载时会先加载一张压缩的特别厉害的缩略图**，以提高用户体验。

73. 一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么?

过程：浏览器输入域名->浏览器查询dns是否有缓存->DNS查询到域名->TCP/IP链接（三次握手）->建立连接->浏览器发出请求->服务器响应（1.2.3.4）->浏览器会先获得响应头然后在获得响应体

1. 当发送一个 URL 请求时，浏览器都会开启一个线程来处理这个请求，同时在远程 DNS 服务器上启动一 个 DNS 查询，看这个请求的url是否存在，这能使浏览器获得请求对应的 IP 地址。
2. 浏览器与远程 Web 服务器通过 TCP 三次握手来建立一个 TCP/IP 连接。
3. 一旦 TCP/IP 连接建立，浏览器会通过该连接向远程服务器发送 HTTP 的 GET 请求。远 程服务器收到请求并找到资源并使用 HTTP 响应返回该资源。
4. 响应返回之后，收到客户端的应答之后，服务器会关闭TCP/IP连接
5. 客户端接受到响应之后就会开始解析下载响应数据

74. 前端如何进行登录身份的判断

前台发送登录请求 后台返回 token，前台得到后台返回的 token，将其写入到 localStorage 中，后续请求中都携带 token 后台判断 token 是否过期，如果过期就对前台的请求响应过期标识或者状态码 前台得到过期标识后，清除 localStorage 中的 token，然后重定向到 login 路由

75. 电商项目跟其它项目有什么不同?

电商网站重点在于支付购物环节 其他类型的网站更多在于内容上面的建设

76. 实践题

经过二个月“测评项目“开发，测试，上线，”测评项目“服务于学校实施的第一站，参与测评学校的学生上午8点集中开测。市场反馈有如下情况：

1）打开网站出现502

2）登录进不去系统

3）提交数据一致反复

4）有时出现白屏现象

对于市场反馈情况谈谈你的认识?

首先，502错误，意思是worker忙不过来，参与测评学校的学生上午8点集中开测，在短时间内访问人数过多导致积累了大量的请求，后台服务器忙不过来。解决方法贵公司可以看一下PHP到底在忙些啥，如果是 CPU 密集型计算（应该不会），看看CPU和内存满没满，没满就多开一些worker，满了就多加一些机子。我猜测可能是数据库响应缓慢，然后人多了数据库是否能撑住也是一个考虑，多用一些缓存吧。防流量攻击真没啥好办法，找牛逼一些的机房/CDN 吧。

第二个问题，结合502报错来看，登录不进系统的原因估计也是短时间内访问人数过多导致服务器崩了。

提交数据一直反复就相对复杂了，可能出现的使用场景也有很多。

比如说我们在提交表单的时候，有可能是网络延迟或内存不足，导致点击一次提交按钮，页面没有反应，结果强迫症就犯了，就疯狂的点提交恨不得把手机屏幕点烂来。也有可能是有些毛孩子，点完提交按钮立马点击刷新页面，或直接返回上层页面想卡bug，不为别的，哎~就是玩儿。

对于这种情况，我以前工作的做法就是设置蒙版层，不管你怎么点，只要你点提交，我就给你弹出蒙版层。其实这个问题最根本的原因是程序没有进行重复判断，导致数据库重复写入。前端后端都可以添加一个重复判断逻辑，判断后台数据库中是否已经存在当前提交的数据，避免重复添加。但是这种判断只解决了两次表单依次提交的问题，如果不同用户同时提交表单，数据就不一定会正确了。这种问题就是我们常说的并发问题，并发问题的解决方案也有很多，比如：加锁排队处理等。

最后提到解决白屏问题，我个人在白屏优化实践上尝试过SSR、预渲染、还有骨架屏等一些方案，每个方案都个有自己的优劣，需要根据实际的业务场景进行取舍。

SSR服务端渲染，这个方案可以让页面直接在服务端渲染，但是不利于前后端分离，开发的效率也比客户端渲染低，同时也加大了服务器的压力。而且由于地理位置的不同，不同用户看到的页面也是不一样的，也就是所谓的千人千面，这也为缓存造成了一定困难。

使用预渲染的话，预渲染插件会在编译阶段就将对应的路由编译好插入到app节点，这样就能在js文件解析过程中有内容展示，js解析完成后，项目使用的单页面应用框架会将app节点内的内容替换成它渲染好的内容。但是当真实访问页面的时候，真实数据可能已经和预渲染的数据有了很大的出入，会导致涉及到一些价格、金额、地理位置的地方甚至会导致用户做出一些错误的决定。

使用骨架屏的话，实现原理和预加载类似，也可以很好的完善预渲染的不足。但是也存在与业务深度耦合，页面复杂度变高的问题。当应对需求变更时，骨架屏结构成本，工具实现成本，配置成本，项目的维护成本，都比较高。

77. 项目开发中有遇到什么挑战没?

1.问题描述：1）简单介绍这个项目规模、背景2）什么情况下遇到什么样的问题

2.你处理这个问题的过程及结果：1）遇到问题你如何思考；2）你如何执行的；3）处理结果如何

3.通过处理这个问题，你学到了什么或者说通过这个问题，你看到了你们什么不足，后续动作（采用什么样的方式，在以后的项目中避免再出现这类问题）

1.（使用缓存处理）

在做Vue开发移动端APP时，有个页面比较常见，左边是对所有菜谱品类的展示，右边是对对应菜谱的展示，一开始在开发的时候没注意，就直接在mounted()前面使用async,里面使用await调了两边接

口后，又通过watch监听了品类索引变化，调了一遍接口。后面发现每次切换品类时，页面都有一闪而过的感觉，发现每次都调了一遍接口，这对性能消耗挺大。所以就尝试的走了Vuex做缓存处理。当时解决这个缓存问题用了一些巧妙的方法，首先缓存的数据，采用的是对象形式，不是数组。这样写起来更快，因为用品类的下标直接做缓存数据的key值，非常好写。不过后面又有点坑，就是我监听的是引用数据类型，不管是vue还是react，引用数据类型发生变化时，页面可能不会更新。后面又在mutation中对vuex中的数据更新时，做了一下深复制，就做好了缓存处理。因为我是根据判断这个对象中有没有数据去调接口的，如果存在，就不调接口。现在做了缓存没错，那如果以后后台的数据发生了变化的话，那我这里也不调接口了，后台数据就没有在页面上实现更新，所以在跳出这个页面的时候还要清下缓存。清缓存要在生命周期结束的时候清，如果碰到了动态组件把APP页面下的 Tap 栏包住的时候， Keep-alive。就不能用 destroyed 生命周期清除，要用 deactivated 生命周期去清除。这样才有始有终，完成缓存处理。

2.（解决关键词高亮问题---->原理和字符串敏感词替换一样，但是用在关键词高亮上算是一个技术亮点吧）

用户配置一堆关键词，在页面上将这些关键词高亮，也许你会觉得这有什么难度？用正则匹配一下出来高亮不就行了吗？但是，一开始，用户的词不多，我确实使用的是遍历，时间复杂度为n2。后来用户会配置100w量级的词，使用遍历就会使页面卡死崩溃。解决的方法就是：优化性能，高亮分三步，生成字典树，遍历页面文字，取出文字进行匹配。使用字典树代替遍历，整个页面100w量级的词绘制可以实现在1秒以内。

3.（封装自定义组件）

在一个小程序项目需求中，要求页面头部tab栏切换的同时，对应tab栏品类的页面也要展示出来。你可能会觉得这个需求用个taroUI组件库中的tabs标签页组件不就可以完成吗？但是这个需求想满足用户的沉浸式体验，切换页面时需要有独特丝滑的专场特效。所以当时面临的问题就是，使用的UI库的组件默认样式生硬，满足不了需求。我当时是对小程序原生swiper组件进行了二次封装，主要实现了几点自定义需求：头部Tab栏品类样式使用flex动态布局，实现品类数量可变；使用 slot 插槽来动态渲染 Tab区块中的内容，配合原生swiper组件使用定义插槽；小程序原生组件<swiper>是有默认高度的，必须手动设置其高度，这里使用wx.getSystemInfo来动态获取屏幕尺寸。自己封装组件，踩了不少坑，但从中我学习到了：使用小程序的原生组件，并修改其默认的样式；学会使用 slot 插槽，实现组件内容的差异化；学会了使用小程序原生 api 获取手机信息，用 js 改变组件样式等等

4.（遇到了难用的轮子）

在写XXX小程序项目中需要实现音频播放的功能，官方已经推荐使用新的API了，然而这个小程序官方文档文档写得并不好，很多时候我们会遇到一些需求，或者改变而这些需求文档里又写得非常模糊，这就比较头疼了。小程序官方文档更新十分频繁，坑非常多。所以很多时候，我就要去不断地从文档的字里行间猜测，并结合源码一步一步地去跟踪，去尝试解决这个音频播放问题。但有时候确实

超出了我的能力范围，那么我就会把我的问题提炼成一个小demo，到知乎、segmentfault思否去问，或者提问一些同样用这个轮子的作者，最终找到了新API的使用规范。

78. 项目研发流程中作为前端开发一般扮演的啥角色?

前端开发一般扮演着一个“团队核心废物”的角色。为什么这么说，首先在项目的研发中，UI小姐姐总感觉前端开发的页面满足不了她们的设计理念，说你没品味；后端小哥哥觉得前端开发就像只会写写样式的js交互工程师，像极了一个破美工的；测试小哥哥拿着测试报告说：这锅谁他娘的来背一下。还记得当年那个请你根据手机壳的颜色，来实现APP启动的颜色的产品经理嘛？这些对前端开发的误会难道不能折射出前端是团队里最应该学会沟通的人嘛?

界面有问题需要和UI沟通,数据有问题需要和后台沟通,功能有问题需要和产品沟通,测试的时候给你提bug你还需要和测试沟通……毕竟前端是最接近用户的人,用户对一个网站,软件最直观的感受是反映到前端；交互体验更是前端项目的核心点。

和UI的沟通,在工作中我们不应该是被动的实现UI的设计,而是应该合理化的提出自己的想法,不然日后返工浪费的是双方的时间。比如通用组件的设计，每次页面的提示弹窗设计，再比如你需要做一个图表,用到了echarts,你完全可以让UI基于echarts去设计样式,而不是让她在那里自由发挥，因为你永远不知道设计师的脑子里装了多少创意,这样节省的是两个人的时间,不会出现他做好样式而你实现不了的尴尬。

和后端联调接口前，先要对业务需求了解透彻，需要哪些数据，有时候明明后台来处理某个事件很简单,后台非要你来做,这就需要我们对一个需求,一个任务的要有清晰认识了,如果对任务含糊不清,自己都没搞明白,你只能受后台摆布了.最后可能也会因为任务没有完成而备受责难了。有理有据,后台开发人员是不会说什么的,否则,后台会很不耐烦的,甚至骂你的可能都有,本身做后台比较难,尤其在查询数据,取数据,封装数据方面都比较难处理。

面对产品经理的需求，前端应该深刻理解需求，毕竟工作性质影响了一个人的思维逻辑，前端能站在一个产品经理的角度去思考每一个需求，便显得尤其重要。不放过每一个细节也很重要。产品经理在设计一个产品的时候，都是从大方向去想问题的，大方向没有错就行了，细节脱离不了大方向。这是他们想的。但是对于程序来说，却万万不能。因为一个细节的逻辑往往决定了整个大方向。

举个例子：有一个需求，用户的作品需要提交审核，经过审核才可以让所有人看到。当产品经理交这个需求给你的时候，你能察觉到什么问题了吗？这里面有几个细节：

1.用户提交审核后，用户可以不可以再编辑作品；

2.作品是否会多次审核；

3.需不需要记录审核历史；

4.用户作品是否需要有版本的控制，如要产生版本，版本又是如何产生的；

5.审核通过后，用户可以不可以再修改作品，若不可以，那么是不是其他人就看不见用户作品等等。

我认为前端开发是团队里最应该学会沟通的人，一个好的前端开发模式可以推动整个项目的进步的，尽管有时可能会被误解成废物，但是这何尝不是大家公认的核心呢？

79. 现在有的项目中觉得哪些项目可以继续优化，为啥没有优化？

之前用vue做了一个动态官网项目，后期客户要求seo，百度上之前搜索不到官网地址，后来在项目的入口文件index.html页面加上了，固定的meta标签，加上name名为keywords、description的meta标签。以上做了个简单的seo优化，这个项目有几个官网，但是其中只有一个官网要求seo，也就是在百度能够搜索到，当时为了应急，就写死了，但是，其它的网站也就会受到干扰了，也就是对于一个项目对应几个官网，写死的meta标签做seo是不科学的。

如果这个项目要解决seo优化，可以用服务端渲染（ssr），如果项目刚开始就考虑到seo，采用服务端渲染，那么就用服务端渲染就得了。但是一般来讲，项目做到后期才会考虑到seo的问题，这时再去搞服务端渲染，相当于重头写项目，非常耗费人力物力。

所以先只考虑在首页加入 meta 标签提供一些元数据，使用简单、具有表意性的 title 以及使用 h5 提供的具有语义化的标签（不要一堆 div），生成对 search engine 友好的 sitemap，使用合理的 html 结构（比如按标题、内容、页脚这样的顺序、或者将重要的内容放在 html 前，其他放在后）

80. 平时写项目总结么，一般总结哪些东西？

平时在写项目的时候，会写项目总结，常常会遇到什么技术难点后，先用自己积累的知识点去尝试解决，如果遇到不会的，我就会在网上查看文档，通过翻阅资料解决问题。在解决问题后，一般我会将我认为有坑的地方记录下来。

比如，最近我做的一个比较有特点的项目，用的是 React + taro + koa框架，完成类似听书小程序。里面有个书城页面，是自己封装了品类筛选的组件。用的是小程序 scroll-view 的原生API，这里面有好多个坑，首先他有一个API，是scrollintoView,这个API的值不能数字开头，但是后端给的数字一般都是id，数字嘛。这只能自己把后端传的数字处理成string类型。最后还有一个样式的坑，你一定要按照官方文档中给的CSS样式来，给scroll-wrap加上white-space：nowrap和宽度百分百，才能实现滚动的效果。

除了总结遇到的问题，还会总结一下自己负责模块的一些规范，比如打包技术，图片的使用，自己使用了那些UI库的组件实现了什么具体的功能。自己负责的模块中联调了哪些接口。最后总结下大概完成了哪些需求。

81. 请简单绘制登录场景的业务流程图，如不熟悉登录业务，也可以选择自己之前项目的业务简单说明。

React做的后台管理系统的登录。【鉴权，不同角色的权限管理】

82. 项目上线后，会将 index.html 给后端，在地址栏上输入 www.abc.com，当在地址后面缀上 /layout 回车后，页面会报 404，是否遇见过这个问题，又该如何去解决?

修改配置文件，把默认首页从index.html改成5ox.html就可以了。

83. 项目中由谁定接口，公司文档如何管理，由谁负责上传代码，怎么上传代码的，项目发布都是怎么做的?

首先，接口文档不应以前端作为主导方提出，而是由服务端进行接口文档的编写工作，在编写完成后，和前端一起协商补充，这样做是最好的。前端只负责数据的展示，所以没必要由前端进行接口定义，因为服务端的逻辑处理如果完全按照前端提出的来，那还咋设计。。最重要的是沟通，并且熟悉业务逻辑。

公司文档，

项目上传由运维上传

84. 请你说说高级前端工程师和初级以及中级有什么区别?

初级前端工程师：能熟练使用HTML、CSS、JS主要工作还是搭建静态页面。一套代码能适配PC+手机端。以外，还需要会使用一些框架之类的东西，像bootstrap、jquery之类的。会ajax了解怎么与后台交互是学习Ajax的关键点。

中级前端工程师：首先就是前端工程化有一定的了解，框架angular、vue、react 。那它和jquery有着很大区别。vue是数据控制页面渲染及状态，而jquery是DOM节点控制渲染，vue渲染页面更容易更优雅。vue能够把前端项目彻底工程化，有配置文件、可以安装第三方模块、配合webpack打包、可以实现模块化开发..等等，当然简单是它最大的优势。熟练es6 7 语法、vuex、Element_ui （开发pc端框架）、vux（开发手机端框架）、Mint UI（开发手机端框架）、Nodejs（后端语言，js语法）。

高级前端工程师：要求编写的模块更具有可复用性，不用怎么修改，后续要加新功能时，也不需要改动原有代码，就能让项目逐渐的稳定下来。有卓越的技术方案设计和架构能力了。能从原理层面来解决问题。对前端工程化、数据可视化、Webkit、可视化搭建、跨端、移动端动画等领域极为精通。来自于深耕，基本上有什么问题、需求或者技术评审，叫上他们就行。钻研和解决问题的能力。时刻关注业界新动态，能快速研究一项新技术，了解和思考如何应用以及对已有业务的帮助。如何提升体验以及赋能。遇到问题就解决问题，不抱怨。执行力和沟通能力。跨部门、团队协调合作，做一件事情能有始有终做好并且拿到成绩，拿到之后还可以汇报展示出来。

85. 用过echars与highchars么，你遇到哪些问题及如何解决的

在用React项目中集成过echars图标表，首先引入之后，用window来访问，如果访问成功，就说明安装成功。指的注意的是echars图表初始化实例其实是dom操作，当用React的hooks写法时，这个初始化实例的操作要放在副作用里，就是用useEffect包起来，不然的话就影响性能。有时候还要注意样式，如果不给，图表就会没有高度，就不会显示出来。一般是从后端动态数据，先做一层处理，在一般是叫 options 生成器，把后端数据处理成 echars 的规范的数据，在引入到视图展示。还有注意调数据是异步的，所以要确保拿到数据后在显示图表。如果做大量图表，我们就需要封装一个个的 option，不然数据会非常复杂。我们调接口肯定是走状态管理，一般的做法会使用两个useEffect，一个用来初始化，一个用来触发调接口。因为图表的实例化是dom操作，所以我们还有可以用hooks中useRef这个API去优化一下。

在工作中也是用过Ant-V，不过大都是看着文档来，思路都是一样的，实在有搞不定的问题，只能面向百度编程了。

86. 项目开发中是用什么工具来管理代码的；说一下你是用过的工具用法（git、svn）

git包管理工具，加入一个项目组，正式开始开发，一般有以下几个步骤：

1、git clone git@gitlab.com ——使用git clone克隆远程项目到本地，克隆到本地后只有一个分支master

2、git checkout -b test ——本地创建一个test分支并为测试环境代码分支并切换至test，这里根据原项目仓库的测试环境代码分支命名，有些使用dev，这里假定原项目测试环境代码分支为test，测试test分支代码跟master分支代码相同

3、git pull origin test ——拉取远程test分支代码并合并到本地test分支，经过本步操作，本地test分支就是测试环境代码了，以后开发分支的代码都要先合并到test分支推到远程测试环境经过测试后才能进行线上部署

4、git checkout master ——再切回master分支

5、git checkout -b develop-branch ——创建开发分支develop-branch并切换至develop-branch分支，这一步就是要进入开发了，一个需求一个开发分支，直到功能开发完毕

6、git add new_file.php ——把开发过程中新建的文件添加进版本管理

7、git commit -m '备注' ——提交所有变更

8、git checkout test ——切换至测试分支

9、git merge develop-branch ——合并新开发的变更到test分支

10、git push origin test ——test推到远程分支，此时测试环境可以拉取test分支代码，然后进行测试

11、git checkout master && git merge develop-branch ——测试通过后，切换到master分支然后合并develop-branch到master分支

12、git push origin master ——master分支推到远程仓库，进行后续的部署上线

87. 讲一下最近的这个项目中都负责什么

1. 这个项目的列表展示与详情页是我负责的。我对于后端传递的数据进行了怎样的处理，在异步请求中选择了一定的异步分割处理数据，拆分一次性阻塞主线程的时间，可以减少用户的等待，页面滚动时选择节流，减少无效的axios请求等等，对自己模块所负责的内容，进行梳理。

2. 这个项目的登录，注册，模块是我负责的。我对不同角色的鉴权是怎么实现的。。。

3. 这个项目的表单提交页是我负责的。。。。用了什么UI组件。。。

【问法同一、二大题类似，展开自己最拿手的回答即可】

88. 怎么判断是开发环境 生产环境

在node中，我们有一个对象process对象，它里面包括的一些信息，env和它的一些属性，当然NODE_ENV是我们自己加上去的自定义属性，用来区分环境变量，也就是通过这个变量来进行区别是开发环境还是生产环境；但是有个问题，不同电脑上设置的方式是不一样的，所以cross-env就来了，它可以跨平台设置环境和使用环境变量。

npm install cross-env

我们在webpack.base.conf.js文件中修改代码：

```
const NODE_ENV=process.env.NODE_ENV;
console.log(NODE_ENV);
```

然后我们修改package.json文件：

//--config是可以设置我们执行哪个webpack文件，默认是执行webpack.config.js,但是我们现在修改文件名了，所以我们要设置一下

```
"build": "cross-env NODE_ENV=production webpack --config webpack.config.prod.js",
"dev": "cross-env NODE_ENV=development webpack-dev-server --config webpack.config.dev.js"
```

就这样，我们就实现了利用webpack来区分开发环境和生产环境，当我们用npm run dev运行的时候就是开发环境，当我们运行npm run build的时候就是构建生产环境打包；

89. Vue如何在用户没登陆的时候重定向登录界面?

现在 我们需要实现这样 一个功能，登录拦截，其实就是 路由拦截，首先在定义路由的时候就需要多添加一个自定义字段requireAuth，用于判断该路由的访问是否需要登录。如果用户已经登录，则顺利进入路由， 否则就进入登录页面。在路由管理页面添加meta字段

```
{
 path:'/manage',
 name:'manage',
 component:manage,
 meta:{requireAuth:true}
}
```

在入口文件中，添加路由守卫

1. 先判断该路由是否需要登录权限
2. 判断本地是否存在token，如果存在token就next()，不存在token重定向到登录页

90. Vue项目常见优化点

1、首屏加载优化

2、路由懒加载

```
{
 path: '/',
 name: 'home',
 component: () => import('./views/home/index.vue'),
 meta: { isShowHead: true }
}
```

3、开启服务器 Gzip

开启 Gzip 就是一种压缩技术，需要前端提供压缩包，然后在服务器开启压缩，文件在服务器压缩后传给浏览器，浏览器解压后进行再进行解析。首先安装 webpack 提供的compression-webpack-plugin 进行压缩,然后在 vue.config.js：

```
const CompressionWebpackPlugin = require('compression-webpack-plugin')
const productionGzipExtensions = ['js', 'css']......plugins: [
 new CompressionWebpackPlugin(
  {
   algorithm: 'gzip',
   test:    new RegExp('\\.('productionGzipExtensions.join('|') +            ')$'),
   threshold: 10240,
   minRatio: 0.8
   }
)]....
```

4、启动 CDN 加速

我们继续采用 cdn 的方式来引入一些第三方资源，就可以缓解我们服务器的压力，原理是将我们的压力分给其他服务器点。

5、代码层面优化

- computed 和 watch 区分使用场景
- computed： 是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值。当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；

- watch：类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作；当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 (访问一个 API)，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。
- v-if 和 v-show 区分使用场景 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。这里要说的优化点在于减少页面中 dom 总数，我比较倾向于使用 v-if，因为减少了 dom 数量。
- v-for 遍历必须为 item 添加 key，且避免同时使用 v-if v-for 遍历必须为 item 添加 key，循环调用子组件时添加 key，key 可以唯一标识一个循环个体，可以使用例如 item.id 作为 key 避免同时使用 v-if，v-for 比 v-if 优先级高，如果每一次都需要遍历整个数组，将会影响速度。

6、Webpack 对图片进行压缩

7、避免内存泄漏

8、减少 ES6 转为 ES5 的冗余代码

91. 异步解决方案有哪些?

解决方案：

/*---------1.回调函数callback：----------*/

被作为实参传入另一函数，并在该外部函数内被调用，用以来完成某些任务的函数。如setTimeOut，ajax请求，readFile等。

例：

```
function greeting(name) {
    alert('Hello 'name);
}

function processUserInput(callback) {
  var name = prompt('请输入你的名字。');
  callback(name);
}

processUserInput(greeting);
```

优点：

解决了异步的问题。

缺点：

回调地狱：多个回调函数嵌套的情况，使代码看起来很混乱，不易于维护。

/*---------2.事件发布订阅：---------*/

当一个任务执行完成后，会发布一个事件，当这个事件有一个或多个‘订阅者’的时候，会接收到这个事件的发布，执行相应的任务，这种模式叫发布订阅模式。如node的events,dom的事件绑定

例：

```
document.body.addEventListener('click',function(){
  alert('订阅了');
},false);
document.body.click();
```

优点：

时间对象上的解耦。

缺点：

消耗内存，过度使用会使代码难以维护和理解

/*---------3.Promise：---------*/

Promise是es6提出的异步编程的一种解决方案。

Promise 对象有三种状态：

pending: 初始状态，既不是成功，也不是失败状态。

fulfilled: 意味着操作成功完成。

rejected: 意味着操作失败。

promise的状态只能从pending变成fulfilled，和pending变成rejected，状态一旦改变，就不会再改变，且只有异步操作的结果才能改变promise的状态。

例：

```
let promise = new Promise(function (resolve, reject) {
```

```
  fs.readFile('./1.txt', 'utf8', function (err, data) {
    resolve(data)
  })
})

promise
  .then(function (data) {
    console.log(data)
  })
```

优点：

解决了回调地狱的问题，将异步操作以同步操作的流程表达出来。

缺点：

无法取消promise。如果不设置回调函数，Promise内部抛出的错误，不会反应到外部。当处于Pending状态时，无法得知目前进展到哪一个阶段（刚刚开始还是即将完成）。当执行多个Promise时，一堆then看起来也很不友好。

/*---------4.Generator：---------*/

Generator是es6提出的另一种异步编程解决方案，需要在函数名之前加一个*号，函数内部使用yield语句。Generaotr函数会返回一个遍历器，可以进行遍历操作执行每个中断点yield。

例：

```
- functioncount() {
 yield 1
 yield 2
 return 3
}
var c = count()
console.log(c.next()) // { value: 1, done: false }
console.log(c.next()) // { value: 2, done: false }
console.log(c.next()) // { value: 3, done: true }
```

```
console.log(c.next()) // { value: undefined, done: true }
```

优点：

没有了Promise的一堆then(),异步操作更像同步操作，代码更加清晰。

缺点：

不能自动执行异步操作，需要写多个next()方法，需要配合使用Thunk函数和Co模块才能做到自动执行。

```
/*---------5.async/await：---------*/
```

async是es2017引入的异步操作解决方案，可以理解为Generator的语法糖，async等同于Generator和co模块的封装，async 函数返回一个 Promise。

例：

```
async function read() {
 let readA = await readFile('data/a.txt')
 let readB = await readFile('data/b.txt')
 let readC = await readFile('data/c.txt')

 console.log(readA)
 console.log(readB)
 console.log(readC)
}

read()
```

优点：

内置执行器，比Generator操作更简单。async/await比*yield语义更清晰。返回值是Promise对象，可以用then指定下一步操作。代码更整洁。可以捕获同步和异步的错误。

92. 移动端点击事件 300ms 延迟如何去掉？原因是什么？

300毫秒原因：

当用户第一次点击屏幕后，需要判断用户是否要进行双击操作，于是手机会等待300毫秒，

解决方法：FastClick.js

FastClick 是 FT Labs 专门为解决移动端浏览器 300 毫秒点击延迟问题所开发的一个轻量级的库。FastClick的实现原理是在检测到touchend事件的时候，会通过DOM自定义事件立即出发模拟一个click事件，并把浏览器在300ms之后的click事件阻止掉。

93. 如何实现函数的柯里化？比如 add(1)(2)(3)

```
/*-----解决方法1：-----*/
function add () {
 var args = Array.prototype.slice.call(arguments);

 var fn = function () {
  var sub_arg = Array.prototype.slice.call(arguments);
      // 把全部的参数聚集到参数的入口为一个参数： args.concat(sub_arg)
  return add.apply(null, args.concat(sub_arg));
 }

 fn.valueOf = function () {
 return args.reduce(function(a, b) {
    return a + b;
  })
 }

 return fn;
}
console.log(add(1,2)) // 3
console.log(add(1)(2)) // 3
console.log(add(1)(2)(3)) // 6
console.log(add(1,2,3)(4)) // 10
```

```
/*-----解决方法2：-----*/

function add () {
  var args = Array.prototype.slice.call(arguments);

  var fn = function () {
    // 把参数都放在一个相当于全局变量的 args 里面
    args.push(...arguments)
    return fn;
  }

  fn.valueOf = function () {
    return args.reduce(function(a, b) {
      return a + b;
    })
  }

  return fn;
}
console.log(add(1,2)) // 3
console.log(add(1)(2)) // 3
console.log(add(1)(2)(3)) // 6
console.log(add(1,2,3)(4)) // 10
```

94. 什么是反柯里化

在JavaScript中，当我们调用对象的某个方法时，其实不用去关心该对象原本是否被设计为拥有这个方法，这是动态类型语言的特点。可以通过反柯里化(uncurrying)函数实现，让一个对象去借用一个原本不属于他的方法。

95. 如何避免回调地狱?

1. Promise 对象就是为了解决这个问题而提出的。它不是新的语法功能，而是一种新的写法，允许将回调函数的嵌套，改成链式调用。

promise只有两个状态resolve和reject，当它触发任何一个状态后，它会将当前的值缓存起来，并在有回调函数添加进来的时候尝试调用回调函数，如果这个时候还没有触发resolve或者reject，那么回调函数会被缓存，等待调用，如果已经有了状态(resolve或者reject)，则立刻调用回调函数。并且所有回调函数在执行后都立即被销毁。

2. ES6 co/yield方案

yield: Generator 函数是协程在 ES6 的实现，而yield是 Generator关键字， 异步操作需要暂停的地方，都用yield语句注明。

co: co 模块是著名程序员 TJ Holowaychuk 于 2013 年 6 月发布的一个小工具，用于 Generator 函数的自动执行。

3. ES7 async/await 方案

async/await是es7的新标准，并且在node7.0中已经得到支持。

它就是 Generator 函数的语法糖，async函数就是将 Generator 函数的星号（*）替换成async，将yield替换成await，仅此而已。可以理解官方对co和Generator 封装方案。

96. 开发过程中遇到内存泄漏的问题都有哪些?

1. 当页面中元素被移除或替换时，若元素绑定的事件仍没被移除，在IE中不会作出恰当处理，此时要先手工移除事件，不然会存在内存泄露。

2. 由于是函数内定义函数，并且内部函数--事件回调的引用外暴了，形成了闭包。闭包可以维持函数内局部变量，使其得不到释放。

97. 浏览器有哪些兼容问题，你封装过什么插件

```
//1.滚动条到顶端的距离（滚动高度）
var scrollTop = document.documentElement.scrollTop || document.body.scrollTop;

//2.滚动条到左端的距离
```

```
var scrollLeft = document.documentElement.scrollLeft || document.body.scrollLeft;

//3. IE9以下byClassName
function byClassName(obj,className){
  //判断是否支持byClassName
  if(obj.getElementsByClassName){
    //支持
    return obj.getElementsByClassName(className);
  }else{
    //不支持
    var eles = obj.getElementsByTagName('*'); //获取所有的标签
    var arr = []; //空数组，准备放置找到的对象
    //遍历所有的标签
    for(var i = 0,len = eles.length;i < len;i ++){
      //找出与我指定class名相同的对象
      if(eles[i].className === className){
        arr.push(eles[i]); //存入数组
      }
    }
    return arr; //返回
  }
}

//4. 获取非行内样式兼容  IE:currentStyle  标准：getComputedStyle
function getStyle(obj,attr){
  return window.getComputedStyle ? getComputedStyle(obj,true)[attr] : obj.currentStyle[attr];
}
//div.style.width = '';设置样式
//obj['属性']： 对象是变量时，必须用对象['属性']获取。

//5. 获取事件对象的兼容
```

```
evt = evt || window.event

//6. 获取鼠标编码值的兼容
function getButton(evt){
  var e = evt || window.event;
  if(evt){
    return e.button;
  }else if(window.event){
    switch(e.button){
      case 1 : return 0;
      case 4 : return 1;
      case 2 : return 2;
    }
  }
}

//7. 获取键盘按键编码值的兼容
var key = evt.keyCode || evt.charCode || evt.which;

//8. 阻止事件冒泡的兼容
e.stopPropagation ? e.stopPropagation() : e.cancelBubble = true;

//9. 阻止超链接的默认行为的兼容
evt.preventDefault ? evt.preventDefault() : evt.returnValue = false;

//10. 添加事件监听器的兼容
function addEventListener(obj,event,fn,boo){
  if(obj.addEventListener){
    obj.addEventListener(event,fn,boo);
  }else if(obj.attachEvent){
    obj.attachEvent('on'event,fn);
```

```
    }
}

//11. 移除事件监听器的兼容
function removeEventListener(obj,event,fn,boo){
    if(obj.removeEventListener){
        obj.removeEventListener(event,fn,boo);
    }else if(obj.detachEvent){
        obj.detachEvent('on'event,fn);
    }
}

//12. 获取事件源的兼容
var target = event.target || event.srcElement;
```

98. 假如A页面我定义了一个定时器，然后跳到B页面如果让A页面的定时器暂停

方法1：在beforeDestroy()等生命周期结束阶段内清除定时器：

```
beforeDestroy() {
    clearInterval(this.timer);
    this.timer = null;
}
```

方法2：通过$once这个事件侦听器器在定义完定时器之后的位置来清除定时器。

```
const timer = setInterval(() =>{
    // 某些定时器操作
}, 500);
// 通过$once来监听定时器，在beforeDestroy钩子可以被清除。
this.$once('hook:beforeDestroy', () => {
```

```
    clearInterval(timer);
})
```

99. 深拷贝是什么？项目哪里是用到了深拷贝？

// 答案：

1，在拷贝构造函数中假如只完成了数据成员本身的赋值则称为“浅拷贝”；编译器提供的默认拷贝构造函数就已经可以完成这个任务。

而假如要复制的数据除了属性值本身以外，还要复制附加在数据属性值上的额外内容，那就要自己来写拷贝构造函数了，来完成所谓的“深拷贝”。

举个例子：

若在构造函数中new了一个新的空间存放数据，并且用指针记录了首地址；若是浅拷贝，则在拷贝构造函数中指针值将复制给另一个数据成员，这样就会有两个指针指向同一个空间；这样的话在析构函数里将会对指针所指向的空间进行释放，由于两个指针指向的是同一个空间，在释放第一个指针指向的空间时不会出现什么问题，而释放第二个指针指向的空间时就会因为空间已经被解析过而导致解析的空间不存在的情况，就会造成程序无法终止。

而解决上面这种情况的办法就是使用“深拷贝”，深拷贝是在拷贝构造函数里再new一个新的空间。将数据复制在新空间里，并将拷贝的指针记录这个新空间的首地址，这样在析构函数里就不会有问题了。

2，在某些引用类型值不更新的情况下用深拷贝

100. swiper 插件从后台获取数据没问题，css 代码啥的也没问题，但是图片不动，应该怎么解决？

主要原因：

swiper提前初始化了，而这个时候，数据还没有完全出来。

解决方法

从swiper 入手，在swiper中写 observer:true/observeParents:true

```
1  let myswiper = new Swiper(".swiper-container" , {
```

```
    autoplay: true,
    loop: true,
    // observer 修改swiper子元素时自动初始化swiper
    observer:true,
    // observeParents 包括当前父元素的swiper发生变更时也会初始化swiper
    observeParents:true,
})
```

从 Vue 入手，vue中专门提供了提供了一个方法nextTick() 用于解决dom的先后执行问题。

```
mounted(){
    this.$nextTick(function(){
        // ...操作
        let myswiper = new Swiper(".swiper-container" , {
            autoplay: true,
            loop: true
        })
    }) }
```

101. 常见内存泄漏

1、静态集合类，如HashMap、LinkedList等等。如果这些容器为静态的，那么它们的生命周期与程序一致，则容器中的对象在程序结束之前将不能被释放，从而造成内存泄漏。简单而言，长生命周期的对象持有短生命周期对象的引用，尽管短生命周期的对象不再使用，但是因为长生命周期对象持有它的引用而导致不能被回收。

2、各种连接，如数据库连接、网络连接和IO连接等。在对数据库进行操作的过程中，首先需要建立与数据库的连接，当不再使用时，需要调用close方法来释放与数据库的连接。只有连接被关闭后，垃圾回收器才会回收对应的对象。否则，如果在访问数据库的过程中，对Connection、Statement或ResultSet不显性地关闭，将会造成大量的对象无法被回收，从而引起内存泄漏。

3、变量不合理的作用域。一般而言，一个变量的定义的作用范围大于其使用范围，很有可能会造成内存泄漏。另一方面，如果没有及时地把对象设置为null，很有可能导致内存泄漏的发生。

4、内部类持有外部类，如果一个外部类的实例对象的方法返回了一个内部类的实例对象，这个内部类对象被长期引用了，即使那个外部类实例对象不再被使用，但由于内部类持有外部类的实例对象，这个外部类对象将不会被垃圾回收，这也会造成内存泄露。

5、改变哈希值，当一个对象被存储进HashSet集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，造成内存泄露

6、缓存泄漏

内存泄漏的另一个常见来源是缓存，一旦你把对象引用放入到缓存中，他就很容易遗忘，对于这个问题，可以使用WeakHashMap代表缓存，此种Map的特点是，当除了自身有对key的引用外，此key没有其他引用那么此map会自动丢弃此值

7、监听器和回调

内存泄漏第三个常见来源是监听器和其他回调，如果客户端在你实现的API中注册回调，却没有显示的取消，那么就会积聚。需要确保回调立即被当作垃圾回收的最佳方法是只保存他的若引用，例如将他们保存成为WeakHashMap中的键。

102. 插入几万个 dom ，如何实现页面不卡顿?

让创建插入节点的工作分批进行：

```
setTimeout(() => {
    // 插入十万条数据
    const total = 100000;
    // 一次插入 20 条，如果觉得性能不好就减少
    const once = 20;
    // 渲染数据总共需要几次
    const loopCount = total / once;
    let countOfRender = 0
    let ul = document.querySelector("ul");
    function add() {
        // 优化性能，插入不会造成回流
        const fragment = document.createDocumentFragment();
        for (let i = 0; i < once; i++) {
            const li = document.createElement("li");
            li.innerText = Math.floor(Math.random() * total);
            fragment.appendChild(li);
        }
        ul.appendChild(fragment);
        countOfRender += 1;
        loop();
    }
    function loop() {
        if (countOfRender < loopCount) {
            window.requestAnimationFrame(add);
```

```
        }
    }
    loop();
}, 0);
```