1. vue的双向数据原理是什么

**vue**双向数据绑定是**通过**数据劫持**并结合**发布-订阅模式**的方法来实现的，**也就是说数据和视图同步，数据发生变化，视图跟着变化，视图变化，数据也随之发生改变。

**数据劫持**：通过 **Object**.**defineProperty**() 对 **data** 中数据进行数据劫持，分别给每一条数据添加 **get** 和 **set** 方法；

**发布订阅者模式**：

数据劫持时，会给每一条数据生成一个 **Dep**（订阅者）；

当数据被节点使用时，数据的 get 方法执行，生成一个对应的 **Watcher**（观察者），并将 **Watcher** 保存到该数据对应的 **Dep** 中；

当数据被修改时，数据的 **set** 方法执行，数据的 **Dep** 机会调用自己的 **notify** 方法，通知当前保存的所有 **Watcher** 更新页面；

1. react、vue、angular的区别你是怎么看待，用于哪些场景

**React**：**React** 是一个基于组件的 JavaScript 库，广泛应用于单页应用程序 (SPA)、移动应用程序和桌面应用程序。它的虚拟 DOM 架构支持快速、高效的 UI 更新，并具有出色的性能。React 是高度可定制和扩展的,与其他库和框架可以很好地集成。但 React 本身只关注视图层，因此它需要结合其他库和框架来实现更全面的应用开发。

**Vue**: **Vue** 是一个渐进式的框架，支持逐步增强的应用程序开发。Vue具有与React相似的虚拟 DOM 架构，但它更加容易学习和使用。Vue 具有丰富、强大的指令集，易于管理状态、调试，并且提供了更多的生态系统支持。Vue 还可以与其他库和框架轻松集成。

**Angular**: **Angular** 是一个完整的前端框架，它提供了更多的分层、抽象、可重用组件，适用于大型、复杂的应用程序。Angular 自带了许多功能来支持模板、数据绑定、路由、依赖注入等等。Angular 的学习曲线比React和Vue要高，但可以在更高的级别上提高应用程序的可维护性。

针对不同的场景，这些框架有各自适用的领域，如下：

**React** 适合于需要高效、响应式更新视图的中大型项目，例如单页 Web 应用、电商平台等需要支持对组件化视图进行模块化、高度重用的开发场景。

**Vue** 适合与相对简单的前端项目开发，以及需要前端与后端打通，是快速开发Web 页面的最佳选择，它提供的模板语法和指令非常容易上手使用，支持组件化视图模块化的开发场景。

**Angular** 通常用于大型企业级项目，比如管理后台或者电商平台，因为它的结构比较灵活，依赖注入和事件处理很容易统一管理和维护，为后期的大型项目发展保驾护航。

1. 在项目中你会把哪些模块抽取出来？为什么要这样设计？

**类型检查模块**：JavaScript是一门动态语言，类型不是在编译时确定的。因此，为了确保代码的正确性，我们可以使用类型检查工具（如TypeScript或Flow），将静态类型添加到代码中。这个模块可以帮助我们提前捕获类型错误，减少运行时错误和调试时间。

**工具函数模块**：在项目中，有很多常用的功能可能在多处使用，我们可以将这些功能抽取出来作为工具函数模块。这些函数可能涉及字符串处理、数组处理、日期处理等。将这些常用功能抽象为可重用的模块，在需要使用时直接引入即可。

**状态管理模块**：对于大型的应用程序，为了方便管理状态，我们通常使用状态管理框架（如Redux或MobX）。这个模块可以帮助我们将状态抽象为单个存储区，结构清晰，方便管理和维护。

**组件库模块**：对于UI相似的组件，我们可以将它们封装为可重用的组件库，方便项目各处使用。这个模块可以帮助我们避免重复造轮子，提高开发效率。

正确拆分工程代码逻辑，将相关的代码逻辑打包成一个可复用的模块，避免代码耦合带来的风险和代码分类不清晰等问题，有利于代码的维护和重用，提高开发效率和代码的质量

1. 给你10个页面，请你给出大概完成时间？如何思考这个问题？

这10个页面是否是相互独立的，还是彼此有依赖关系和公共组件？

完成时间是否指一人独立完成，还是多人协作完成？

如果这10个页面互不相关且没有公共组件，那么预估完成时间较为简单，我们可以按照页面难度和工作量估算出每个页面的时间，再求和即可。如果是一个人独立完成，通常需要考虑工作负载、个人技能水平等因素。如果是多人协作完成，可以根据团队的人数和协作效率估算出时间。

然而，实际情况下，页面之间通常是有依赖关系和公共组件的，这会影响到完成时间的预估。在这种情况下，我们需要考虑以下几个问题来思考如何预估完成时间：

页面之间有哪些公共组件和依赖关系，会影响到时间和工作量的分配；

各个页面的难度和工作量分别是多少；

团队人员的技能水平、工作效率、协作方式等因素；

项目进度和时间要求。

综合考虑以上因素，我们可以利用过去的经验和项目管理工具（如JIRA等）来制定合理的项目计划和排期，并及时根据实际情况进行调整和优化。在项目开发过程中，还需要进行不断的进度跟踪和风险管理，以确保项目按时、高质量完成。

1. webpack的性能优化？

**优化 loaders 配置**：通过 include 属性来指定要编译的文件，或通过 exclude 来排除不需要编译的文件；

**优化 resolve.alias 配置**：通过配置别名来将原导入路径映射成一个新的导入路径；

**优化 module.noParse 配置**：让 webpack 忽略对部分没采用模块化的文件的递归解析处理，这样做的好处是能提高构建性能（例如 jquery）；

**拆分 bundles：**使用 HardSourceWebpackPlugin 拆分 bundles；

**使用 HappyPack**：webpack 是单线程模型的，也就是说 webpack 需要一个一个地处理任务，不能同时处理多个任务。HappyPack 将任务分解给多个子进程去并发执行，子进程处理完后再将结果发送给主进程；

**使用 ParallelUglifyPlugin 压缩 JS**：webpack 默认使用 UglifyJS 插件来单线程压缩代码，ParallelUglifyPlugin 插件则会开启多个子进程，把对多个文件压缩的工作分别交给多个子进程去完成，但是每个子进程还是通过 UglifyJS 去压缩代码；

**优化文件监听的性能**：在开启监听模式时，默认情况下会监听配置的 Entry 文件和所有 Entry 递归依赖的文件，在这些文件中会有很多存在于 node\_modules 下，因此一个很大的优化点就是忽略 node\_modules 下的文件，不监听它们：

**使用 Tree Shaking：**配置 Tree Shaking 来删除项目中无用的代码；

**提取公共代码：**大型网站通常由多个页面组成，每个页面都是一个独立的单页应用，但由于所有页面都采用同样的技术栈及同一套样式代码，就导致这些页面之间有很多相同的代码。可以使用 splitChunks 进行分包；

**分割代码以按需加载：**按照路由的方式实现页面级的懒加载。 例如 React 中：

1. react为什么要使用hooks

React Hooks 是 React 16.8 的新特性，它可以让我们在函数组件中使用状态和其他 React 特性。使用 React Hooks 有以下几个好处：

**更简洁、明确的代码：**使用 Hooks 可以让函数组件代码更加简洁、易于理解，对于没有太多业务逻辑或状态维护的组件，可以避免使用 Class Component 时所带来的冗余代码。

**使代码结构更加清晰：**使用 Hooks 可以把逻辑相关的代码组织为一个 Hook 函数，把不同功能的代码分离出来，使代码结构更清晰。

**可以更好地复用逻辑：**使用 Hooks 可以把具有相同功能的代码封装为一个 Hook 函数，可以在多个组件中复用，避免了代码的重复编写。

**方便进行局部状态维护：**在 Class Component 中，需要在 constructor 中初始化状态，并需要绑定事件处理函数，而使用 Hooks 可以更方便地定义组件的局部状态，并可以在函数组件的生命周期中实现状态更新及其它操作。

**更好地与其它库和框架协同：**Hooks 将函数组件与 React 的生命周期和其它特性连接起来，这样就可以更好地与 Redux、MobX、React Router 等库和框架协同工作。

总之，使用 Hooks 可以让我们写出更加简洁、清晰、易于维护、可重用的代码。Hooks 使函数组件成为 React 中的完整构建块，并为开发者提供了更好地处理前端状态、响应事件和其他 React 等功能的方式。

1. 不用for循环打印500个“\*”

可以使用字符串的 **repeat** 方法来输出多次相同的字符。例如可以使用以下方法打印500个“\*”：

**console.log("\*".repeat(500));**

其中，repeat 方法接受一个整数参数，表示要重复的次数。在上面的例子中，我们将要重复的字符设置为 “\*”，将其重复 500 次后直接输出到控制台即可。

使用 **repeat** 方法可以减少代码量，避免 for 循环所带来的额外开销和冗余代码，并且可以更加简单和易读。这也是 JavaScript 语言的一大特点——有很多内置方法可以方便地实现各种需求。

1. 在一个函数中，接受外部参数，如何区分这个参数是对象还是数组

在 JavaScript 中，可以使用 **typeof** 运算符来判断一个变量的类型。当参数为对象时，**typeof** 运算符会返回 “object”，当参数为数组时，也会返回 “object”，因此无法通过 **typeof** 判断参数的具体类型。

要区分传入的参数是对象还是数组，可以采用以下方式：

使用 **Array.isArray()** 判断是否为数组。该方法会返回一个布尔值，表示传入的参数是否为数组。

使用 **Object.prototype.toString.call()** 方法，可以返回一个表示数据类型的字符串，例如 “[object Array]” 表示数组类型，“[object Object]” 表示对象类型。因此，我们可以通过判断类型字符串的方式来判断传入的参数的类型。

1. 假如有一个父组件一个子组件，如果在父组件定义一个数据，点击按钮改变它，并且将这个数据渲染到子组件中，两个组件生命周期的执行顺序

在父组件中定义一个数据，然后将这个数据传递给子组件作为 **props**，当点击按钮时，更改父组件中定义的数据，并调用 **setState**() 方法更新组件状态，从而重新渲染父组件及其子组件。

执行顺序如下：

首先，父组件的构造函数会被调用，然后子组件的构造函数会被调用；

然后，父组件的 render() 方法会被调用，接着子组件的 render() 方法也会被调用；

当父组件中的状态发生改变时，父组件的 **componentDidUpdate**() 方法会被执行，从而再次调用 render() 方法重新渲染组件；

当子组件接收到父组件传递的 props 发生变化时，子组件的 **componentDidUpdate**() 方法会被执行，从而再次调用 render() 方法重新渲染子组件。

总之，父组件和子组件的生命周期方法的执行顺序与普通情况下一样，父组件告诉子组件 Props 发生变化时，子组件的 **componentDidUpdate**() 方法会再次被调用，以实现 Props 的更新。

1. 请说一下git工作流程？

Git 是现代软件开发中最受欢迎的版本控制工具之一。以下是 Git 工作流程的基本步骤：

**初始化仓库：**使用 **git init** 命令初始化一个 git 仓库，生成.git目录。

**添加文件到仓库：**使用 **git add** 命令将修改的文件添加到 git 的暂存区。

**提交文件到仓库：**使用 **git commit** 命令将添加到暂存区的修改提交到仓库中。

**分支操作：Git** 使用分支来进行代码的开发和管理。使用 **git branch** 命令可以查看当前仓库存在的分支，使用 **git checkout** 命令可以切换分支。在新开发功能时，可以使用 **git checkout** -**b** 命令，新建一个分支进行开发。

**合并分支：**使用 **git merge** 命令将一个分支的修改合并到另一个分支上。

**提交修改：**使用 **git push** 命令将本地仓库的修改提交到远程仓库中。

**拉取修改：**使用 **git pull** 命令将远程仓库的修改拉取到本地仓库中。

**查看状态：**使用 **git status** 命令可以查看当前仓库或文件的状态，如未提交的修改以及未追踪的文件。

**查看提交历史**：使用 **git log** 命令可以查看当前分支的提交历史。

Git 工作流程是分布式版本控制的一种体现，它采用了多分支的开发模式，可以让开发者更好地组织和管理代码，避免冲突和数据丢失。在使用 Git 进行软件开发时，需要注意合理地分支管理和提交修改，保持代码的干净性和可维护性。

1. 涉及到页面的按钮级权限你是怎么做的？

当涉及到页面的按钮级权限时，通常有两种做法：

**前端根据用户角色动态渲染:** 前端代码可以根据当前用户所属的角色，动态改变页面中的按钮状态或是否渲染该按钮，使用户只有在满足一定条件时才能够点击该按钮。这种方式比较简单，但只是单纯地隐藏按钮，并不能真正保证数据安全性。

**后台控制权限:** 后台根据用户的角色控制页面组件的显示和业务逻辑。后台给不同角色分配不同的权限，在前端通过请求后台接口来获取对应的权限。根据权限返回不同渲染的内容。这种方式可保证安全性，但需要在后台进行维护控制，增加了后台的工作量。

根据不同的项目需求和安全要求，可以采用以上两种方式或进行二者的结合使用。但是，需要注意在前端隐藏按钮时，应该保证数据操作的安全性，例如在后台再次进行验证。在后台控制权限时，需要设计良好的权限体系和管理后台，确保角色权限的分配和修改可靠和安全。

1. 小程序怎么获取到用户的信息与头像？

小程序可以通过 **wx**.**getUserInfo**() 方法获取用户的基本信息，包括昵称、头像、性别、城市等，具体步骤如下：

在小程序的 **app**.**json** 文件中添加 **scope** 属性，用于获取用户信息授权，如下所示：

**{**

**"pages":[**

**"pages/index/index",**

**"pages/user/user"**

**],**

**"window":{**

**"navigationBarTitleText": "小程序"**

**},**

**"requiredBackgroundModes":["audio"]**

**"permission": {**

**"scope.userinfo": {**

**"desc": "获取用户信息"**

**}**

**}**

**}**

在需要获取用户信息的页面中引入 **wx**.**getUserInfo**() 方法：

**wx.getUserInfo({**

**success: function(res) {**

**var userInfo = res.userInfo;**

**var nickName = userInfo.nickName;**

**var avatarUrl = userInfo.avatarUrl;**

**var gender**

1. 你说一下webscoket有哪些常用额api，分别作用是什么？

**WebSocket**是一种在客户端和服务器之间进行双向通信的网络协议，在实际应用中经常使用WebSocket API进行编程。以下是WebSocket常用的API和它们的作用：

**new WebSocket(url, protocols)：** 创建一个WebSocket对象，指定连接的URL和可选的协议字符串。

**WebSocket.send(data)：** 将数据发送到服务器。

**WebSocket.close([code[, reason]])：** 关闭WebSocket连接。可选的code和reason参数可以指定关闭的原因代码和文本。

**WebSocket.readyState：** 返回当前WebSocket连接的状态码。常用的值有：0表示连接尚未建立，1表示连接已建立，2表示连接正在关闭，3表示连接已关闭。

**WebSocket.onopen：** 当WebSocket连接建立时触发的事件。

**WebSocket.onmessage：** 当从WebSocket接收到消息时触发的事件。

**WebSocket.onerror：** 当WebSocket发生错误时触发的事件。

**WebSocket.onclose：** 当WebSocket连接关闭时触发的事件。

这些API可以帮助开发人员创建WebSocket应用程序，进行双向通信，并管理连接状态。

1. 本地存储 cookie、localStorage、sessionStorage的区别。

首先，四者最大的区别在于 cookie、localStorage、sessionStorage 都是前端的本地存储，数据保存在浏览器中；而 session 是后端的本地存储，数据保存在服务器中。

**数据有效期：**

**cookie**：一般由服务器生成，可设置失效时间。如果在浏览器生成，默认是关闭浏览器后失效；

**localStorage**：除非手动删除，否则永久保存；

**sessionStorage**：仅在当前会话有效，关闭浏览器或浏览器窗口后被清除；

**session**：由服务器生成，可设置失效时间。

**作用范围：**

**cookie**：在所有同源窗口中共享；

**localStorage**：在所有同源窗口中共享；

**sessionStorage**：只在当前窗口有效；

**存储大小：**

**cookie**：4KB；

**localStorage**：一般 5MB；

**sessionStorage**：一般 5MB；

**与服务器通信**：

**cookie**：每次请求都会携带在 HTTP 请求头中，如果使用 cookie 保存过多数据会带来性能问题；

**localStorage**：不会主动把数据发送给服务器，仅在本地保存；

**sessionStorage**：不会主动把数据发送给服务器，仅在本地保存；

1. 小程序的getUserProfile用不了怎么办

1、只调用**wx.login**拿到**code**后，让用户自行补充姓名和头像**。**

2、**input**组件的**type**属性设置为**nickname**后，可以拿微信呢称

3、**button**组件的**open-type**设置为**chooseAvatar**，可以拿到微信头像

1. uniapp跳转页面的API有哪些？分别有什么特点？

**navigateTo：**保留当前页面，跳转到应用中的某个页面（不允许跳转到 **Tab** 页）。新页面可以通过 **navigateBack** 返回到原页面。该 **API** 相对快速，但最多只能跳转到五层页面。

**redirectTo：**关闭当前页面，跳转到应用中的某个页面。新页面无法通过 **navigateBack** 返回到原页面。该 API 相对来说较快，但无法返回上一页。

**reLaunch：**关闭所有页面，打开某个特定页面。该 **API** 的主要用途是在某些特殊场景下重新打开主页，但对于一般跳转场景并不适用。

**switchTab：**跳转到应用的 **Tab** 页，从而打开某个特定的页面。该 **API** 只适用于底部导航被选中的情况下，不能在非 **Tab** 页中跳转。

1. 你能说一下堆、栈、链表、树之前的一些特点吗？

当我们讨论数据结构时，堆、栈、链表、树是常见的几种基础数据结构。它们各自有一些特点，可以在不同的情况下使用。

**堆：**

是一种完全二叉树结构；

一般用于实现优先队列等数据结构；

可以分为最大堆和最小堆；

最大堆的根节点最大，最小堆的根节点最小；

插入、删除操作时间复杂度都是 O(log n)。

**栈：**

是一种简单的数据结构，它的元素按照一定的顺序排列；

实现方式可以用数组或链表；

后进先出(LIFO)的原则，即最后进入的元素最先弹出；

是函数调用中的重要数据结构；

插入、删除操作时间复杂度都是 O(1)。

**链表：**

是一种线性数据结构，它的元素不是连续存储的；

每个节点包含两个部分：数据和指向下一个节点的指针；

可以分为单向链表、双向链表和循环链表；

链表的插入、删除操作时间复杂度是 O(1)；

能够有效地动态管理内存，具有较强的灵活性。

**树：**

是一种非线性数据结构，由若干个节点组成；

每个节点包括数据和指向子节点的指针；

可以分为二叉树、平衡树、红黑树、B+树等；

二叉树每个节点最多有两个子节点；

用于存储有层级关系的数据，如目录结构、组织结构等；

插入、删除操作的时间复杂度取决于树的高度，平衡树的高度较低，插入、删除操作时间复杂度为 O(log n)。

以上是堆、栈、链表、树的一些基本特点，它们各自有一些适用场景和具体应用。了解它们的特点，可以帮助我们更好地选择合适的数据结构来解决问题。

1. 除了闭包还有哪些内存泄漏的场景？

除了闭包外，还有一些其他场景也容易导致内存泄漏。下面列出一些主要的场景：

**DOM 元素未正确清除：**如果在 JavaScript 中保留了对 DOM 元素的引用，并且这些引用未被正确清除，在页面卸载或销毁时将导致内存泄漏。

**计时器未正确清理：**如果在 JavaScript 中注册了计时器，并忘记在不需要时取消，将导致计时器一直运行，即使在页面卸载或销毁时也不会停止，导致内存泄漏。

**全局变量：**如果在 JavaScript 中声明了全局变量，并且这些变量未在不需要时移除，它们将一直存在于内存中，导致内存泄漏。

**对象引用循环：**如果对象之间形成引用循环，这将使 JavaScript 垃圾回收机制无法正确识别和回收不再使用的对象，导致内存泄漏。

**事件监听器未正确清理：**如果在 JavaScript 中添加了事件监听器，并且这些监听器未在不需要时移除，将会导致内存泄漏。

总之，内存泄漏是指不再使用的内存未被正确释放，这将导致应用程序不断消耗内存，最终可能导致性能下降或崩溃。因此，开发者需要注意以上场景，避免内存泄漏的发生。

1. 怎么判断浏览器上两个元素相交？

在前端开发中，判断两个元素是否相交是一个非常常见的需求，可以通过以下方法来实现：

**getBoundingClientRect 方法：**该方法可以返回一个 DOM 元素的大小及其相对于视口的位置。通过比较两个元素的边界框，可以判断它们是否相交。具体来说，如果两个元素的边界框相交，则它们相交。

**Intersection Observer API：**该 API 可以观察一个元素与另一个元素或视口的相交情况，并在相交状态变化时执行一些操作。通过注入一个回调函数，在相交状态改变时进行操作，根据回调参数中的交叉比率或交叉信息来判断元素是否相交。

**模拟选区：**可以使用 document.getSelection() 方法获取一个选区对象，然后通过选区对象的 getRangeAt() 方法获取选区范围。如果选区范围与判断的元素相交，则它们相交。

**CSS 判断：**如果两个元素有父元素，可以通过使用 CSS 的左、上、宽和高属性，结合一些简单的计算来判断它们是否相交。

**监听滚动事件：**可以通过监听滚动事件，并判断两个元素是否在可视区域内，来判断它们是否相交。

根据实际需求和场景不同，可以选择不同的方法来判断元素是否相交。

1. react闭包陷阱是什么？如何解决这种问题

React 闭包陷阱是指在 React 中使用闭包会导致意外的副作用，从而引起一些无法预测和难以排查的问题。这通常是由于 React 的状态和 props 会随时间变化，但在某一时刻闭包中的状态并不会随着这些变化而更新，从而导致闭包中的状态和实际状态不一致。

在 React 中，我们可以通过将状态保存在 useState 或 useReducer 钩子的返回值中来解决闭包问题。这些钩子将自动嵌入基础状态管理逻辑，并在每次更新组件状态时自动进行更新。下面是一些解决方案：

1. **useRef hook：useRef hook** 可以用于保存不需要触发重新渲染的旧数据。通过在闭包中引用 **useRef** 的返回值，可以避免在重新渲染时更新它。

2. **useCallback hook：useCallback hook** 可以用于返回一个稳定的函数引用，而不是一个新的函数引用。这可以确保闭包中的函数始终引用相同的函数，从而避免在重新渲染时更新。

3. **useState 和 useReducer hook：**可以使用 useState 和 useReducer hook 来管理组件状态，以解决闭包问题。这些 hook 将自动更新状态，并防止在闭包中使用旧状态。

4. **使用class组件：**使用 class 组件可以通过使用 this.state 和 this.props 来避免闭包问题，因为它们会自动更新。

总之，React 闭包陷阱可以通过使用 React 自带的 Hook 和避免在闭包中使用状态来解决。尽管使用这些钩子可能会稍微复杂一些，但是它们可以避免一系列与闭包相关的问题，从而降低代码维护的难度。

1. 文件批量上传，大小不一致，怎么判断上传后的顺序？

文件批量上传并且大小不一致时，可以通过多种方式来判断上传后的顺序，以下是一些常见的方式：

**1. 文件名或者文件顺序**：在文件上传前，在前端对上传的文件按照文件名或者指定文件顺序排序，然后在上传成功后，通过文件名或者顺序来判断上传后的文件顺序是否准确。

**2. 指定上传顺序**：可以事先制定每个文件的上传顺序，采用同步上传或队列上传方式，确保每个文件的上传顺序都正确。

**3. 通过计算哈希值来判断**：通过计算每个文件的哈希值作为文件的唯一标识，在上传成功后，再将文件哈希值与之前预先计算好的哈希值进行比较，如果相等，则证明上传后的顺序正确。

**4. 在上传文件时添加标识符**：可以在上传文件时，为每个文件添加一个标识符或者参数，当文件上传成功后，从后台获取该参数，根据参数值来判断各个文件的上传顺序是否正确。

综上所述，可以采取不同的方式来判断上传后的文件顺序是否正确。不同的场景可以根据实际需求选择不同的方法。

1. 说一下你对插槽以及作用域插槽理解？

**插槽（Slot）**是**Vue.js**组件模板中一种特殊的HTML标签，允许开发者在组件内部预留一个或多个“孔”，让父组件可以动态地往里面“填充内容”。简单来说，插槽就是一种将父组件传入的内容动态地渲染到子组件内部的机制。

**作用域插槽（Scoped Slot）**是插槽的一种特殊类型，允许子组件在插槽内部访问父组件的数据和方法。这个特性可以将组件的灵活性和可复用性大大提高，因为它使得父组件能够动态地向子组件传递数据和方法，并且能够在多个不同的组件之间复用。

举个例子，假设我们有一个名为 UserList 的用户列表组件，它需要接受一个用户数组作为 prop，并显示每个用户的名字和邮箱。我们可以在 UserList 中定义一个作用域插槽，让父组件可以动态地传入一段自定义的 HTML，用于显示用户的详细信息。在作用域插槽内部，我们可以通过访问 slot-scope 属性来获取父组件传入的数据和方法。

1. vue常用修饰符有哪些？

Vue.js框架提供了一些修饰符（Modifiers），用于在指令后面添加一些特殊的行为。常用的Vue修饰符如下：

**1. `.**prevent**`：**阻止默认事件的发生，例如：`<a v-on:click.prevent></a>`。

**2. `.**stop**`：**停止事件冒泡，例如：`<a v-on:click.stop></a>`。

**3. `.**capture**`：**添加事件侦听器时使用事件捕获模式，例如：`<a v-on:click.capture></a>`。

**4. `.**once**`：**指令只会触发一次，例如：`<a v-on:click.once></a>`。

**5. `.**self**`：**只当事件发生在该元素本身时触发，不包括其子元素，例如：`<div v-on:click.self></div>`。

**6. `**.passive**`：**指令不会阻止事件的默认行为，可以提高移动端的性能，例如：`<div v-on:touchmove.passive></div>`。

**7.** `.native`**：**指令绑定到组件的根元素上而不是组件内部，例如：`<my-component v-on:click.native="doSomething"></my-component>`。

这些修饰符可以应用在v-on、v-model、v-show、v-cloak和v-bind等指令上，以实现不同的功能。掌握这些修饰符可以让我们更加灵活地使用Vue.js框架，提高开发效率。

1. vuex中保存的数据页面刷新后丢失了怎么办？

在 Vuex 中保存的数据是存储在内存中的，当页面刷新时，这些数据就会丢失。为了解决这个问题，我们可以将 Vuex 中的数据持久化到浏览器的本地存储（如 localStorage 或 sessionStorage）中。这样即使页面刷新，数据也不会丢失。

1. 拖拽用过吗？如何在网页中实现拖拽功能

在 Web 前端开发中，我们可以使用以下两种方式在网页中实现拖拽功能：

**1、使用** HTML5 **拖放** API

HTML5 提供了一组用于处理拖放相关事件的 API，即拖放 API。我们可以使用这些 API 来实现在网页中的拖拽功能。HTML5 拖放 API 包括以下事件：

dragstart：拖拽开始触发的事件

dragenter：拖拽元素进入目标元素时触发的事件

dragover：拖拽元素悬停在目标元素上时触发的事件

dragleave：拖拽元素离开目标元素时触发的事件

drop：拖拽元素在目标元素上释放时触发的事件

dragend：拖拽结束触发的事件

HTML5 拖放 API 提供了一些方法，例如：

dataTransfer.setData(format, data)：将数据存放到拖拽事件的对象中

dataTransfer.getData(format)：从拖拽事件的对象中获取数据

**2、使用第三方拖拽库**

除了使用 HTML5 拖放 API，我们也可以使用第三方拖拽库来快速实现拖拽功能。常见的第三方拖拽库有 jQuery UI、React DnD 和 Vue.Draggable 等。例如，Vue.Draggable 提供了一个基于 Vue.js 的拖拽组件，你可以在 Vue 项目中使用它来实现拖拽功能。

总之，在 Web 前端开发中，拖拽是一项非常常见的功能之一，我们可以采用上面提到的方法来实现它，让用户更加轻松自如地移动、交互和管理网页上的元素。

1. 如何实现大文件分段上传？

分段上传可以将大文件切分成多个小的文件块进行上传，一次只上传一部分，最后将这些文件块合并成一个完整的文件。

**以下是实现大文件分段上传的一些步骤：**

**1、**将上传的大文件切分成小文件块，建议大小为几MB到几十MB的范围内，具体大小可以根据上传速度和实际需求进行调整。

**2、**上传文件块时，每次只上传一个文件块，可以使用HTTP协议的分块传输编码（chunked encoding）来实现，或者使用HTTP的Content-Range头来指定上传的文件块在整个文件中的位置。

**3、**上传文件块时，要使用相同的文件名和文件类型，以确保合并时这些文件块能够正确地组合在一起。

**4、**上传完成所有文件块后，进行文件的合并操作，可以将文件块按照它们在原始文件中的位置拼接起来。这个过程最好在服务器端完成，以减少客户端开销和避免网络中断导致的文件合并失败。

总之，分段上传可以提高上传完成率并减少上传失败的概率，可以提高用户上传大文件的体验。

1. 什么叫CSRF攻击？如何防止这种攻击？（换种说法，你的token被别人拿到了，利用这个token伪造你的身份去访问）

CSRF（Cross-site Request Forgery）攻击，也称为“跨站请求伪造”攻击，**是一种利用用户已经登录了某个网站的信任进行攻击的方式。**攻击者通过一些手段欺骗用户在其他网站上执行他们并不知情的操作，比如删掉邮箱里的邮件或在社交媒体账号中发布一些恶意消息等。

**攻击流程如下**：

1. 用户登录某个网站A，并在该网站上获得了一个cookie。

2. 在用户没有登出该网站A的情况下，用户访问了另一个网站B，并在该网站上执行了一些操作。

3. 网站B利用用户在网站A上的信任，发起了一次在用户不知情的情况下对网站A的请求。

**防止CSRF攻击需要采取以下一些方法：**

**1. 同源策略：**保证网站A、B必须同源（协议、域名、端口都相同），避免其他域名下的网站发起对网站A的CSRF攻击。

**2. 阻止第三方请求：**可以在请求中加入一些令牌或其他随机数，并在服务器端对这些信息进行验证，从而阻止不合法的请求。

**3.** **CSRF Token验证**：在每个表单或链接中添加一个CSRF Token，确保该操作来源于合法的网站。如果用户在其他网站访问该链接，请求中不会包含合法的Token，因此请求会被服务器拒绝。

**4.** **Verify-Referer**：在请求的HTTP头中加入Referer头，验证请求是否来自合法的参考站点。

总之，遵循同源策略，采用防护措施比如CSRF Token验证和Verify-Referer等方式，可以有效防止CSRF攻击。

1. 正式环境中如果出现了bug你怎么解决呢？

如果在前端正式环境中出现了 bug，首先应该确认 bug 的具体表现，然后分析产生 bug 的原因。

**以下是一些常用的解决方法：**

**1.检查代码：**检查前端页面的代码和相关的 JavaScript、CSS 和HTML 文件是否存在语法错误或逻辑错误。另外，可以使用调试工具，如Chrome DevTools或Firefox Web Developer等来查看网站的代码，以帮助诊断问题。

**2.查看服务器日志：**如果问题与服务器相关，比如后端接口出现错误或者服务器遇到了一些问题，可以查看服务器日志以帮助确定问题所在，并作出相应的修复。

**3.使用debugger（调试器）进行调试：**在JavaScript代码中插入断点，并使用浏览器的调试器进行调试。在调试过程中，可以查看变量的值、函数的调用堆栈等信息，以帮助找出问题所在。

**4.利用版本控制工具进行回滚：**如果问题很严重或者已经影响到了用户，可以使用版本控制工具来回滚到之前的稳定版本。通常情况下，正式环境应该与版本控制工具保持同步，因此回滚应该是一个可行的方法。

总之，解决前端正式环境中出现的问题需要有很好的分析能力和技术能力，在确定问题的原因后，要及时采取适当的措施来修复问题，确保网站的稳定性和可靠性。

1. 什么叫单点登录？

单点登录（Single Sign-On，简称SSO）是一种用户身份验证机制，它可以在多个应用程序或网站中使用单组凭据进行身份验证。在SSO中，用户仅需在首次登录时提供一次凭据，即可通过认证，之后在任何已配置为使用相同身份验证服务的其他应用程序或网站上自动登录。

通常，SSO机制由一个中央身份验证授权系统（Identity Provider，简称IdP）维护所有的用户信息和认证信息。当用户访问其他应用程序时，应用程序会将用户重定向到 IdP，以验证用户的身份。如果用户已经经过身份验证，则 IdP 将用户重定向回应用程序，并向应用程序提供身份认证信息。

SSO **的优点如下：**

**提高用户体验：**用户只需要提供一次凭据进行身份验证，即可访问多个应用程序或网站，避免了反复登录的繁琐。

**增强安全性：**SSO 可以集中管理用户信息和认证信息，避免了用户在不同应用中使用相同密码的问题，从而提高了安全性。

**提高运维效率：**SSO中心化管理了用户信息和认证信息，可以避免在多个应用程序或网站上分别管理用户名和密码，从而减少系统管理员的工作负担。

总之，SSO提供了简化用户身份验证的方式，可以提高用户体验，增强安全性，减少系统管理员的工作负担，被广泛应用于企业内部门户、电子商务网站、在线办公工具等场景。

1. uniapp云开发是什么意思？

UniApp云开发是指使用uni-app框架结合云开发技术的一种开发方式。UniApp是一个跨平台的开发框架，它可以基于 Vue.js 框架提供快捷的开发体验，同时支持多种前端开发模式，如H5、小程序、App等。而UniApp云开发，是在此框架的基础上，利用云开发技术实现了一些基本的云端功能，如开发、部署、监控、数据分析等。

与传统的基于服务器的开发模式不同，UniApp云开发借助云环境，实现了前端应用的快速开发和部署，并提供了完善的云端服务，如云函数、数据库、云存储、即时通讯等，可以显著减轻前端开发人员的后端服务工作负担，从而提高了开发效率。另外，uni-app云开发还支持云IDE，可以在线编写、修改和测试代码。

目前，UniApp云开发主要支持基于微信小程序和支付宝小程序的开发，可在其它前端开发平台中集成云开发的相关能力。UniApp云开发是一种快速开发技术，对于需要快速迭代开发、在小程序中开发移动应用等场景下具有较高的适用性。

1. 一个项目从架构层面如何实现一键换肤

一键换肤是指通过切换不同的主题风格，实现界面样式的动态更新。从架构层面来说，一个项目实现一键换肤需要具备一些技术支持和架构支持。

以下是一些关键的技术和架构要点：

**1. 颜色资源：**将应用程序中的颜色和样式资源进行抽象和封装，使得这些资源可以在多个主题之间共用。比如，在一个app主题文件中定义不同主题所需要的颜色值、图片样式、字体等。

**2. 主题管理器：**添加主题管理器模块，负责管理app主题文件，以及根据用户的选择动态加载对应的主题文件，更新应用程序的主题风格。

**3. 动态加载方式：**应采用动态加载方式，使得用户可以在使用应用程序时随时切换不同的主题。

**4. 缓存机制：**应该有缓存机制，可以减少应用程序的网络请求。对于用户选择的主题，需要进行相应的缓存，确保在下次使用应用程序时不需要重复下载相应的主题文件。

**5. MVP、MVC、MVVM等架构模式可以在相应基础上，加入额外的功能模块切换给业务部分。**比如，在MVP架构中View和Presenter适当进行改造来支持主题切换。在切换主题时，Presenter可以调用主题管理器模块从新的主题文件中加载对应的资源。

总之，要实现一键换肤，需要将应用程序资源进行抽象和管理，添加主题管理器模块、动态加载方式和缓存机制等模块支持。在架构模式的选择上可以按照对应框架有约定的方式进行整合。

1. 多维数组变一维数组？请写出代码？

将一个多维数组变成一维数组的常用方法是使用递归。递归函数可以遍历整个数组，并将每一个子数组中的元素添加到结果数组中，最终得到一个一维数组

1. 项目的i18n怎么实现的？

i18n是国际化的缩写，通常是指将应用程序进行本地化处理，以便能够在多个不同语言和地区的环境中使用，为用户提供更好的体验。

以下是一些常用的i18n实现方法：

1. 手动处理：最基本的方式是在应用程序的代码中手动进行文字和数值的翻译，并通过if/else等条件语句确定展示哪一种语言。这种方法适用于小型项目，不需要高度动态的内容。

2. 国际化框架：使用国际化框架，如React intl、jQuery i18n、Vue I18n等，将多语言处理分别配置为组件或模板，并处理需要翻译的内容，从而实现不同语言的展示。

3. 第三方平台：通过使用第三方语言翻译平台，如Google Translate、Baidu Translate等，调用API来实现翻译，从而实现国际化。

对于一个大型项目中i18n的实现，通常应该使用国际化框架，因为它提供标准化的解决方案，可拓展性强，并且可以方便地支持多个语言和管理语言资源。另外，建议将文字和数值考虑成一个外部配置文件，方便日后修改和维护。

总之，i18n是一种通用的应用程序全球化和本地化的实现方法，对于涉及多语言和地区的项目，应该使用相应的技术手段和工具来实现。

1. 说说你对垃圾回收机制的了解

垃圾回收机制是指编程语言自动管理动态分配的内存，及时回收不再使用的内存空间。在垃圾回收机制中，程序无需手动分配和释放内存，这可以避免许多常见的内存管理问题，如内存泄漏和野指针。

垃圾回收机制在不同的编程语言中实现方式各不相同，但通常都包含以下几个主要组成部分：

**1.垃圾收集器：**负责自动扫描程序中分配的内存空间，检测出不再使用的内存块，将其回收。

**2.标记-清除算法：**当一个对象的所有引用都被删除时，这个对象就成为了垃圾。垃圾收集器使用标记-清除算法来识别和回收未使用的堆内存。

**3.引用计数：**垃圾收集器会跟踪每个对象被引用的次数。当一个对象的引用计数为零时，垃圾收集器就会释放该对象的内存空间。

**4.内存压缩：**不同的垃圾回收器采用不同的算法来压缩内存碎片，以便下一次分配内存时可以更有序地使用内存。

总之，垃圾回收机制使得开发人员能够更轻松地编写高效且可靠的代码，同时确保内存使用的可控性和安全性。

1. 你们公司接口有加密吗？你是如何对参数进行加密的？

为了保障前端传输的参数安全，前端可以使用不同的加密方式将参数加密后再发送给后端。以下是一些常见的前端加密方式：

1. HTTPS：HTTPS是一种基于TLS/SSL协议的安全传输协议，可以保证前后端之间数据传输的隐私和完整性。在使用HTTPS时，浏览器与服务器之间的数据流是加密的，而且数据传输的过程中双方可以相互验证对方身份，因此可以保证数据的安全性。

2. Base64编码：Base64是一种将二进制数据转换为文本格式的编码方式，它可以将不可读的二进制数据转换为易读的文本格式，以实现数据传输安全。Base64编码虽然可以将数据加密，但由于它是一种编码而非加密方式，因此不是最安全的加密方式。

3. AES加密：AES是一种高级加密标准，可以将数据加密为只有通过密钥才能解密的密文，因此是一种比较安全的前端加密方式。AES具有高效、可靠、安全的特点，被广泛应用于前端数据加密。

4. RSA加密：RSA是一种公钥加密算法，可以生成公钥和私钥，通过私钥加密数据，公钥解密数据。虽然RSA加密是一种较为安全的加密方式，但前端的数据加密和解密需要借助服务端的公钥和私钥，因而更适用于服务端加密。

总之，前端加密需要根据应用程序的需求和场景进行选择，建议多使用已经经过充分验证的安全加密方式，避免使用简单的编码方式，同时提供密钥管理和更新等措施进行保护。

1. watch和watchEffect的区别

`watch` 和 `watchEffect` 都是 `Vue 3` 提供的响应式函数，它们可以用于监听数据的变化，但两者的用法和效果有所不同。

`watch**` 用于侦听特定数据的变化，一旦被侦听的数据发生变化，就会触发回调函数执行。**它的基本用法如下：

watch(source, callback, options?)

- `source`: 被侦听的数据源，可以是一个函数或一个响应式对象/变量。

- `callback`: 当被侦听的数据发生变化时，触发的回调函数。

- `options`: 可选配置项，包括 `deep` (是否深度侦听) 和 `immediate` (是否立即执行回调函数)。

**`watchEffect` 则可以自动捕获回调函数里面所引用的响应式数据，并在这些数据变化的时候重新运行该函数。**它的基本用法如下：

watchEffect(effect, options?)

- `effect`: 一个包含响应式状态更新相关代码的函数。当其中的响应式变量发生变化时，该函数会重新计算。

- `options`: 可选配置项，包括 `lazy` (懒执行，即只有当 `watchEffect` 函数被 `stop` 后重新启动时才会触发) 和 `scheduler` (调度器，可以控制 `effect` 的执行时机)。

总的来说，`watch` 更适合在需要在数据变化时执行特定操作或异步请求时使用，而 `watchEffect` 更适合在需要根据响应式状态进行操作时使用，它可以自动追踪依赖项。

1. 你在项目中遇到最大的问题是什么？如何解决的？
2. **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。

1. **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。

1. 如果登录过期，如何让客户没感觉的情况下继续登录

当用户在登录过期的情况下重新登录时，可以考虑以下几种方法来减少用户的不便，并确保顺畅的用户体验：

**自动刷新页面：**当用户登录过期时，自动刷新页面，避免用户需要手动刷新浏览器以继续登录。

**提示登录过期：**在页面上显示登录过期的提示信息，让用户知道他们需要重新登录。

**保留用户输入的信息：**如果可能的话，记录用户之前输入的信息，以便他们不需要重新输入。

**提供单点登录：**让用户在多个系统和应用程序之间自动登录，即使一个系统的登录过期，他们仍可以继续访问其他系统。

**在客户端执行自动登录：**在登录会话过期后，使用客户端技术（如OAuth 2.0或OpenID Connect）将用户重新登录到系统，以减少用户感受到的中断。

这些方法可以让用户不感到被打断，从而提高用户体验。

1. 后端接口等等没问题，为啥前端传数据后端拿不到，分析原因

前后端数据传输的过程中，如果后端接收不到前端传输的数据，原因可能有以下几种：

**前端未正常传输数据：**前端数据未正常传输到后端，可能是因为前端传输的数据格式不正确，或者前端代码存在错误，导致无法正常传输数据。

**网络传输问题：**网络传输过程中可能出现网络延迟、数据丢失或中断等问题，导致部分数据无法正常传输到后端。

**注：(后端可能会出现的问题)：{**

**后端接口设置问题：**后端接口可能出现设置问题，例如数据格式不正确、接口路由设置错误、接口参数配置不正确等等，导致后端无法正确接收到前端传输的数据。

**后端数据处理问题：**后端可能存在数据处理问题，例如后端无法正确解析前端传输的数据、数据转换错误等问题。

}

要解决这些问题，后端开发人员应该检查接口的设置并确保它们与前端代码匹配。此外，后端开发人员应该为接口添加适当的错误处理和日志记录，以便快速发现和解决问题。前端开发人员可以参考后端api文档，确保发送正确的请求格式、参数和权限等。

1. 前端除了JOSN外还有什么数据格式

除了JSON外，在前端开发中还有许多其他数据格式可用于数据传输。以下是一些常见的前端数据格式：

1. XML（可扩展标记语言）：XML是一种具有可扩展性的标记语言，可用于在前端和后端之间传输数据。XML与JSON类似，但需要更多的策略和技术来保护应用程序的安全性。

2. CSV（逗号分隔值）：CSV是一种标准格式，在电子表格和数据库中常用。它是一种纯文本格式，每个值都由逗号分隔，并在行末添加换行符。

3. YAML（YAML ain't Markup Language）： YAML是一种简单的数据表示语言，它比JSON更易于阅读和编写。它的主要应用是在配置文件中传输结构数据。

4. protobuf（Protocol Buffers）：PROTOBUF是Google开发的一种数据序列化格式，主要用于高效地传输和存储数据。它是一种二进制格式，相比于JSON和XML可以实现更高效的数据传输。

总之，不同的数据格式适用于不同的场景和任务，前端开发人员应该选择最适合应用程序需求的数据格式。同时需要注意，不同数据格式可能需要不同的处理方式和技术，开发人员需要了解不同数据格式的特点和用途，以便选择和使用最适合的数据格式。