# **elm**

# **1.elm**

## **1.1.项目&功能**

### **1.1.1.做什么**

#### **1.1.1.1.使用 Vue3 最新语法开发一个仿饿了么移动端 Web App**

### **1.1.2.哪些功能？**

#### **1.1.2.1.功能页面、轮播图、滚动加载、图片懒加载、购物车、滚动组件、动画效果、Token 登录**

### **1.1.3.7个接口模块**

#### **1.1.3.1.首页详情接口**

#### **1.1.3.2.首页搜索接口**

#### **1.1.3.3.商品列表接口**

#### **1.1.3.4.用户登录接口**

#### **1.1.3.5.我的详情接口**

#### **1.1.3.6.商家列表接口**

#### **1.1.3.7.商家详情接口**

### **1.1.4.6个视图组件**

#### **1.1.4.1.首页视图组件**

#### **1.1.4.2.我的视图组件**

#### **1.1.4.3.搜索视图组件**

#### **1.1.4.4.登录视图组件**

#### **1.1.4.5.商家详情视图组件**

#### **1.1.4.6.商品详情视图组件**

### **1.1.5.4+个基础组件**

#### **1.1.5.1.ScrollView 组件**

#### **1.1.5.2.LoadingView 组件**

#### **1.1.5.3.Popup 组件**

#### **1.1.5.4.Overlay 组件**

## **1.2.技术架构**

### **1.2.1.业务层**

#### **1.2.1.1.页面**

* 首页页面
* 搜索页面
* 个人页面
* 登录页面
* 商家详情页
* 商品详情页

#### **1.2.1.2.功能**

* 切换动画
* 倒计时
* 轮播图
* 购物车功能
* 滚动加载
* 图片懒加载
* 滚动组件
* Token 登录

### **1.2.2.框架层**

#### **1.2.2.1.组件**

* 路由 vue-router
* 状态管理 pinia
* 请求库 axios
* 移动端组件库 vant-ui

#### **1.2.2.2.工具**

* 脚手架&构建工具
  + vite
* 代码检查工具
  + eslint
  + prettier
  + commitlint
  + husky

#### **1.2.2.3.框架**

* MVVM 框架
  + Vue3
* 语言
  + TypeScript
* mock 服务端
  + json-server

#### **1.2.2.4.部署**

* git
* nginx
* docker
* jenkins

## **1.3.H5**

### **1.3.1.技术栈**

#### **1.3.1.1.typescript**

* 安装使用
  + https://blog.csdn.net/weixin\_41442779/article/details/135895684
  + 使用npm初始化项目
    - npm init
  + 安装typescript
    - npm install typescript -D
  + 将typescript编译成JavaScript
    - npx tsc 文件名
* 基础概念
  + TypeScript是具有类型语法的JavaScript。
  + 一种基于JavaScript的强类型、静态的编程语言，提供了类型检测的工具。
  + 特性
    - 易读
    - 开源
    - 提供代码的错误检测。
    - 和JavaScript兼容性好，兼容 es6等。
    - 提供可靠类型，规范函数入参和出参。
    - 从编程语言来说
      * 按数据类型分
        + typescript 静态类型语言： 在编译代码时能确定变量类型
        + JavaScript 动态类型语言： 在运行代码时能确定变量类型
  + Typescript 缺点
    - 引入太多的新概念，类型泛型、枚举等，渐进式，学会语法就能用。
    - 不适合小项目，会增加开发成本，需要在项目中声明很多的类型等。
  + https://blog.csdn.net/weixin\_41442779/article/details/135874487
* 函数 function
* 接口 interface
* 类 Class
* 泛型 <T>
* 类型断言

#### **1.3.1.2.git**

* 安装与配置
  + git init
* 基础使用
* 常见命令
* SourceTree 使用

#### **1.3.1.3.vite**

* 基础概念
  + 一种新型前端构建工具,能够显著提升前端开发体验
  + 主要由两部分组成
    - 一个开发服务器：它基于原生ES模块,提供了丰富的内建功能,比如速度快到惊人的模块热更新（HMR）
    - 一套构建指令：它使用 Rollup 打包你的代码，并且它是预配置的，可输出用于生产环境的高度优化过的静态资源
* Vite 是怎么优化的？
  + 两个方面优化
    - 开发服务器启动
      * Vite从依赖打包和源码打包这两个方面提升性能
        + 依赖:使用esbuild进行依赖预打包, esbuild使用Go编写,会比javascript-based的打包工具快 10-100倍
        + 源码:使用浏览器原生es module提供源码,让浏览器接管打包工具的部分工作
    - 热更新
      * 使用ESM不需要重新编译:一些打包工具的开发服务器在文件更改时,需要重新构建整个项目，来获取新的模块依赖关系
      * 使用浏览器缓存加速: Vite利用HTTP头来加速整个页面的重新加载
* vite 配置
  + 在项目的vite.config.ts文件中配置。
* 什么是跨域？
  + 同源：两个url请求的协议 protocol、端口 port、域名 host 相同就是同源
  + 跨域：就是不同源的url请求
  + 为什么限制跨域？
    - 出于安全性，浏览器会限制脚本内发起跨源 HTTP 请求 （XMLHttpRequest 和 Fetch API）。
    - web 应用请求  
        
       只能从加载应用程序的同一个请求 HTTP 资源。  
        
       除非包含 CORS 响应头。
  + 如何解决跨域问题？
    - 常用方法：  
        
       jsonp：<script> 不受同源策略限制  
        
       跨源域资源共享 CORS：允许web应用服务器进行跨源访问控制  
        
       使不同源变成同源（使用反向代理）
  + 反向代理如何解决跨域问题？
    - 通过反向代理服务器分配不同标识进行访问，不同的域服务端

#### **1.3.1.4.vue3**

* 安装
  + 使用create-vue创建vue3项目
  + npm init vue
* composition API
* 前后端分离开发、持续集成
* 生命周期
* ref
  + 接受一个内部值，返回一个响应式的、可更改的 ref 对象，此对象只有一个指向其内部值的属性 .value
* reactive
  + 返回一个对象的响应式代理。
* computed
  + 接受一个 getter 函数，返回一个只读的响应式 ref 对象。该 ref 通过 .value 暴露 getter 函数的返回值。它也可以接受一个带有 get 和 set 函数的对象来创建一个可写的 ref 对象。
* watch
  + 侦听一个或多个响应式数据源，并在数据源变化时调用所给的回调函数。
  + 默认是懒侦听的，即仅在侦听源发生变化时才执行回调函数。
* 组合式 API：setup()
  + setup() 钩子是在组件中使用组合式 API 的入口，通常只在以下情况下使用：
    - 需要在非单文件组件中使用组合式 API 时。
    - 需要在基于选项式 API 的组件中集成基于组合式 API 的代码时。
* 组合式 API：生命周期钩子
  + onMounted()
    - 注册一个回调函数，在组件挂载完成后执行。
  + onUpdated()
    - 注册一个回调函数，在组件因为响应式状态变更而更新其 DOM 树之后调用。
  + onUnmounted()
    - 注册一个回调函数，在组件实例被卸载之后调用。
  + onBeforeMount()
    - 注册一个钩子，在组件被挂载之前被调用。
  + onBeforeUpdate()
    - 注册一个钩子，在组件即将因为响应式状态变更而更新其 DOM 树之前调用。
  + onBeforeUnmount()
    - 注册一个钩子，在组件实例被卸载之前调用。
  + onErrorCaptured()
    - 注册一个钩子，在捕获了后代组件传递的错误时调用。
  + onActivated()
    - 注册一个回调函数，若组件实例是 <KeepAlive> 缓存树的一部分，当组件被插入到 DOM 中时调用。  
      这个钩子在服务器端渲染期间不会被调用。
  + onDeactivated()
    - 注册一个回调函数，若组件实例是 <KeepAlive> 缓存树的一部分，当组件从 DOM 中被移除时调用。  
        
      这个钩子在服务器端渲染期间不会被调用。
* 组件封装
* 自定义指令directive

#### **1.3.1.5.vue-router**

* 基础用法  
  动态路由匹配
* 什么是路由？
  + 单页面应用（又叫SPA）：不刷新而切换页面某部分，实际是切换页面里的部分内容。
  + 单页面应用路由：实际表示应用页面的状态
  + url中pathname部分（history api） 或 hash（url中#后面的部分）用来记录路由
  + 一个url对应一个路由
  + 路由就是表示页面的状态，一般用url的 pathname 或者 哈希#带着后面的部分 来表示
* 路由是如何配置的？
  + 第一步找到 内容组件渲染的地方-<router-view> （app.vue 渲染的地方，组件页面渲染的地方等）
  + 第二步找到，路由与组件的对应关系-router配置文件里，配置的路由（router/index.ts）
  + 第三步找到，触发路由跳转的地方 （<router-link>代码处或者代码可动态配置（代码函数调用跳转路由方法的地方） ）(需要判断是否要设置)

#### **1.3.1.6.pinia**

* 基础概念
* state
* getter
* action
* store

#### **1.3.1.7.代码规范**

* husky
* commitlint
* eslint
* prettier

#### **1.3.1.8.性能优化**

* 防抖
* 加载优化
* 图片懒加载

#### **1.3.1.9.设计模式**

* 常用设计模式
  + 工厂模式：使用Vue3的构造函数或者工厂函数来创建组件实例。
  + 单例模式：使用Vue3的Vue实例来创建全局的单例。
  + 观察者模式：使用Vue3的$watch方法来监听数据的变化。
  + 代理模式：使用Vue3的proxy对象来实现数据的代理。
  + 模块模式：使用Vue3的import和export语法来实现模块化。
  + 装饰器模式：使用Vue3的装饰器来增强类或者函数的功能。
  + 策略模式：使用Vue3的mixins来实现策略模式。
  + 适配器模式：使用Vue3的provide和inject来实现适配器模式。
  + 迭代器模式：使用Vue3的for…of循环来实现迭代器模式。
  + 责任链模式：使用Vue3的事件总线来实现责任链模式。
  + 建造者模式：使用Vue3的组件选项来实现建造者模式。
  + 原型模式：使用Vue3的原型继承来实现原型模式。
  + 模板方法模式：使用Vue3的render函数来实现模板方法模式。
  + 状态模式：使用Vue3的状态管理库（如Vuex）来实现状态模式。
  + 外观模式：使用Vue3的组件来实现外观模式。
  + 享元模式：使用Vue3的组件缓存来实现享元模式。
  + 规则模式：使用Vue3的条件渲染和v-if指令来实现规则模式。
  + 。。。。。。
* 设计模式应用

#### **1.3.1.10.项目部署**

* 持续集成 CI/CD
* webhooks
* jenkins
* docker
* nginx

### **1.3.2.项目开发知识点**

#### **1.3.2.1.git提交知识点**

* 使用husky管理githooks
* 使用commitlint规范git commit信息内容

#### **1.3.2.2.项目代码规范**

* 配置eslint
  + 确保 编写的代码 遵循 编码规范
* 配置prettier
  + 将 编写的代码 格式化为 一致的风格

#### **1.3.2.3.vite**

* 基础概念
  + 一种新型前端构建工具,能够显著提升前端开发体验
  + 主要由两部分组成
    - 一个开发服务器：它基于原生ES模块,提供了丰富的内建功能,比如速度快到惊人的模块热更新（HMR）
    - 一套构建指令：它使用 Rollup 打包你的代码，并且它是预配置的，可输出用于生产环境的高度优化过的静态资源
* Vite 是怎么优化的？
  + 两个方面优化
    - 开发服务器启动
      * Vite从依赖打包和源码打包这两个方面提升性能
        + 依赖:使用esbuild进行依赖预打包, esbuild使用Go编写,会比javascript-based的打包工具快 10-100倍
        + 源码:使用浏览器原生es module提供源码,让浏览器接管打包工具的部分工作
    - 热更新
      * 使用ESM不需要重新编译:一些打包工具的开发服务器在文件更改时,需要重新构建整个项目，来获取新的模块依赖关系
      * 使用浏览器缓存加速: Vite利用HTTP头来加速整个页面的重新加载
* 跨域
  + 什么是跨域？
    - 同源：两个url请求的协议 protocol、端口 port、域名 host 相同就是同源
    - 跨域：就是不同源的url请求
  + 为什么限制跨域？
    - 出于安全性，浏览器会限制脚本内发起跨源 HTTP 请求 （XMLHttpRequest 和 Fetch API）。  
        
      web 应用请求  
        
       只能从加载应用程序的同一个请求 HTTP 资源。  
        
       除非包含 CORS 响应头。
  + 如何解决跨域问题
    - jsonp：<script> 不受同源策略限制
    - 跨源域资源共享 CORS：允许web应用服务器进行跨源访问控制
    - 使不同源变成同源（使用反向代理）
* 代理
  + 请求转发 客服端与服务端之间通过代理服务器进行请求。
  + 代理服务器类型：
    - 正向代理
      * 客户端告诉代理服务器资源地址。例如带电话，告诉服务商打给谁。
    - 反向代理
      * 客户端只告诉代理服务器想要什么资源。不知道具体地址。
  + 反向代理如何解决跨域问题？
    - 通过反向代理服务器分配不同标识进行访问，不同的域服务端
* 配置
  + 通过 defineConfig函数，将所有的 Vite 配置集中在一个对象中，从而更清晰地管理和维护项目配置。

#### **1.3.2.4.vue-router**

* 什么是vue-router？
  + vue的官方路由
    - 什么是路由？
      * 单页面应用（又叫SPA）：不刷新而切换页面某部分，实际是切换页面里的部分内容。
      * 单页面应用路由：实际表示应用页面的状态
      * url中pathname部分（history api） 或 hash（url中#后面的部分）用来记录路由
      * 一个url对应一个路由
      * 路由就是表示页面的状态，一般用url的 pathname 或者 哈希#带着后面的部分 来表示
    - 路由是如何配置的？
      * 第一步找到 内容组件渲染的地方-<router-view> （app.vue 渲染的地方，组件页面渲染的地方等）
      * 第二步找到，路由与组件的对应关系-router配置文件里，配置的路由（router/index.ts）
      * 第三步找到，触发路由跳转的地方 （<router-link>代码处或者代码可动态配置（代码函数调用跳转路由方法的地方） ）(需要判断是否要设置)
* 功能点
  + 嵌套路由映射  
    动态路由选择  
    模块化、基于组件的路由配置  
    路由参数、查询、通配符  
    HTML 5 history 模式或 hash 模式
    - 动态路由选择
      * 参数
        + pathname
        + query
        + hash
      * 选择
        + pathname:path
        + query:pathname,query
        + hash:pathname,hash
      * 解析
        + 参数：[1](https://stackoverflow.com/questions/34344529/path-in-pathname-parameters-for-nested-router)
        + 路径映射：[2](https://blog.csdn.net/weixin\_40898859/article/details/105472312)
    - 模块化、基于组件的路由配置
      * 路由映射
        + 导入 Route
        + 基于组件进行配置
        + 将不同路由分离
      * 配置路由
        + 配置嵌套路由
        + 配置组件路由
      * 实现
        + [代码实现](https://www.runoob.com/javascript/js-router.html)
    - 路由参数、查询、通配符
      * 参数
        + 查询：[1](https://zhuanlan.zhihu.com/p/74937005)
        + 通配符：[2](https://www.ruanyifeng.com/blog/2014/11/javascript-router.html)
      * 使用
        + 查询：[3](https://github.com/routerjs/routerjs/wiki/v2.x-Guide#route-paths-and-params)
        + 通配符：[4](https://router.vuejs.org/zh/docs/api/params.html)
    - HTML 5 history 模式或 hash 模式
      * 区别
        + 模式1: 基于 HTML5 的浏览器 history 属性
        + 模式2: 基于 hash 的浏览器历史
      * 使用
        + history 模式：[5](https://developer.mozilla.org/zh-CN/docs/Learn/JavaScript/Guide/Working\_with\_history)
        + hash 模式：[6](https://www.w3schools.com/js/js\_router.asp)
* hooks方法
  + useRoute
    - 记录当前路由的状态、信息，返回当前的路由地址
  + useRouter
    - 返回路由器实例。返回当前路由对象的各种属性和方法（页面跳转等功能）
* RooterView
  + 用于显示用户当前所处路由的组件。

#### **1.3.2.5.ref()**

* 返回一个响应式的、可更改的 ref 对象，此对象只有一个指向其内部值的属性 .value。

#### **1.3.2.6.watch()**

* 用于声明在数据更改时调用的侦听回调。

#### **1.3.2.7.computed()**

* 理解
  + computed 是一个函数，用于创建计算属性。
  + 计算属性是一种派生属性，它的值依赖于其他响应式数据，并且会根据这些数据的变化而自动更新。
  + 接受一个计算函数作为参数，并返回一个响应式的计算属性
  + 该计算函数会在其中定义计算属性的值，并且可以依赖其他响应式数据或其他计算属性。
* 应用场景
  + 数据转换和格式化：可以使用计算属性对响应式数据进行转换和格式化，以便在模板中直接使用转换后的值。
  + 响应式数据的组合：可以使用计算属性将多个响应式数据组合成一个新的值，并保持其响应性。
  + 过滤和排序：可以使用计算属性对列表数据进行过滤和排序，以便在模板中显示符合特定条件的数据。
* 实现原理
  + Vue 3 的计算属性基于 Proxy 代理实现
  + 当计算属性的依赖数据发生变化时，Vue 3 会自动追踪依赖并触发计算函数的重新执行，从而更新计算属性的值。
  + 这个过程是基于底层的响应式系统实现的，其中使用了 Proxy 对象来捕获对响应式数据的访问和修改操作。

#### **1.3.2.8.axios**

* 什么是 axios？
  + 基于promise 网络请求库，在服务端，使用原生node.js的http模块，在客户端（浏览器）中，使用XMLHttpRequests。
* 特点
  + 支持Promise Api
  + 支持拦截请求和响应
  + 支持取消请求
  + 支持请求体自动转换（序列化）JSON数据
* 什么是http状态码？
  + HTTP响应状态码表明特定 HTTP 请求是否成功完成
  + 状态码
    - 信息响应（100-199）
      * 表示请求已被接收，正在处理。
    - 成功响应（200-299）
      * 表示请求已成功被服务器接收、理解、并接受处理。
      * 200 OK：请求成功。常见于 GET 和 POST 请求。
    - 重定向消息（300-399）
      * 表示需要客户端进一步的操作才能完成请求。
      * 301 Moved Permanently：永久重定向。通常用于网站域名变更等情况。
      * 302 Found：临时重定向。通常用于临时跳转到其他页面。
      * 304 Not Modified：资源未修改。客户端可以使用缓存的内容。
    - 客户端错误响应(400-499)
      * 表示客户端发出的请求有误，服务器无法处理。
      * 400 Bad Request：请求参数错误。
      * 401 Unauthorized：未授权，需要身份认证。
      * 403 Forbidden：服务器拒绝了请求，权限不足。
      * 404 Not Found：请求的资源不存在。
    - 服务端错误响应(500-599)
      * 表示服务器在处理请求时发生了错误。
      * 500 Internal Server Error：服务器内部错误。
      * 502 Bad Gateway：网关错误。通常发生在代理服务器出现问题时。
      * 503 Service Unavailable：服务不可用。通常发生在服务器暂时过载或正在维护时。
* 什么是请求拦截？
  + 请求拦截是在发送 HTTP 请求之前对请求进行拦截和处理的过程。
* 什么是业务状态码？
  + 业务状态码的设计可以根据具体的业务需求灵活定义
    - 成功状态码：表示请求处理成功，通常是一个固定的数字或字符串，比如 200 或 success。
    - 错误状态码：表示请求处理失败或出现错误，可以根据具体的错误类型定义不同的错误码，比如 400 表示请求参数错误，401 表示未授权，500 表示服务器内部错误等。
    - 业务逻辑状态码：表示业务逻辑处理的不同结果，比如用户注册成功、订单支付成功等，可以根据具体的业务场景定义不同的状态码。
    - 特定状态码：用于标识特定的业务场景或特定的处理结果，比如 404 表示资源未找到，503 表示服务不可用等。

#### **1.3.2.9.post-css实现移动端适配**

* 什么是移动端适配？
  + 针对移动设备（如智能手机和平板电脑）的不同屏幕尺寸、分辨率、像素密度等特点，对网页或应用进行调整，以确保在各种移动设备上都能够有良好的显示效果和用户体验。
  + 目的是使页面内容在不同设备上呈现一致、合适，并且能够自动适应不同屏幕尺寸和方向。
* 常用适配方案
  + 不同端使用不同代码，比如 PC 端一套代码，移动端一套代码
  + 不同端使用同一套代码，一般使用 CSS 样式来控制--@media
  + 移动端屏幕适配
    - 利用 rem 按根节点（body）的字体大小来缩放
    - 利用 vh/vw 按屏幕高度和宽度来缩放
  + 例子
    - rem适配方案
      * 基于根元素的字体大小（font-size）来设置页面中元素的尺寸。使用 rem 单位定义的元素尺寸将相对于根元素的字体大小进行计算，从而实现在不同设备上的自适应。
* 什么是post-css？
  + PostCSS 是一个用 JavaScript 编写的工具，用于转换 CSS 代码。css转换工具

#### **1.3.2.10.vite全局注册vant组件**

* 全局注册组件
  + import 需要的组件
  + import 组件样式
  + 使用 app.use 注册组件
* 什么是 tree shaking？
  + 是一种在 JavaScript 中用于移除未使用代码的技术。
  + 目标是通过静态分析代码的依赖关系。。来确定哪些代码未被引用，然后将这些未使用的代码。。从最终的构建文件中删除，以减少文件大小。。并提高应用程序的性能。

#### **1.3.2.11.如何设计一个组件**

* 需求分析
  + 布局
    - 组件的样式，风格
  + 功能
    - 组件功能的属性，事件，插槽
  + 创建.vue文件
    - 在script，template，style里进行开发
  + 使用组件
    - 使用import导入组件，在template使用

#### **1.3.2.12.首页组件**

* 使用normalize.css
  + 是一个用于在不同浏览器之间建立一致的 CSS 样式的开源项目。
  + 旨在通过为不同浏览器提供统一的样式基础，解决浏览器之间的差异，以确保在各种现代浏览器中呈现一致的样式和布局。
  + Normalize.css是一种CSS reset的替代方案。
    - Reset通过为几乎所有的元素施加默认样式，强行使得元素有相同的视觉效果。
    - Normalize.css保持了许多默认的浏览器样式。这就意味着你不用再为所有公共的排版元素重新设置样式。当一个元素在不同的浏览器中有不同的默认值时，Normalize.css会力求让这些样式保持一致并尽可能与现代标准相符合。
  + 理解
    - 标签默认样式
      * 一些HTML标签在浏览器中会有默认样式，例如：body标签会有margin:8px；ul标签会有margin:16px 0;及padding-left:40px。
      * 当我们在切图软件中进行尺寸或位置测量的时候，把测量出来的数值设置到对应的标签上时，可能会受到当前标签默认样式的影响，从而页面显示效果跟设计图效果不符。
    - 清除默认样式
      * 常在网页开发中，要去掉这些影响尺寸和位置的默认样式及其他影响布局的默认值。可以参考CSS Tools: Reset CSS方案。
      * 由于Reset CSS相对“暴力”，不管你有没有用，统统重置成一样的效果，且影响的范围很大，所以更加“平和”的一种方式Normalize CSS诞生了。
      * Normalize CSS可以看成是一种Reset CSS的替代方案。创造Normalize CSS有下面这几个目的：
        + 保护有用的浏览器默认样式而不是完全去掉它们
        + 一般化的样式：为大部分HTML元素提供
        + 修复浏览器自身的bug并保证各浏览器的一致性
        + 优化CSS可用性：用一些小技巧
        + 解释代码：用注释和详细的文档来
  + 项目中使用方法
    - 安装
      * npm i normalize.css
    - 使用
      * app.vue,在项目的根目录或者类似的顶级目录中//应用程序的入口点，包含了应用程序的整体布局和结构。进行使用。
* sass
  + css 预处理器
  + 扩展了 CSS 的功能，并引入了许多有用的特性和语法，使得样式表的编写更加灵活和可维护。
  + vue项目中使用方法
    - 组件的 <style> 标签中，使用 lang 属性指定样式语言为 scss 或 sass
* 如何使用图片
  + @使用静态资源里的图片文件时，需要在vite.config.js中resolve里配置@别名然后指定解析路径
* 封装TheTop组件
  + 新建 文件名.vue 文件
  + 在主组件中使用import TheTop from '路径' 来引用组件，在template中引用组件<TheTop />
* 使用VanSearch组件
  + 在项目的主入口文件main.ts中，进行初始化 Vue 应用程序全局配置。将VanSearch组件使用app.use方法进行全局注册。这样它们就可以在整个应用程序中被访问和使用。
* 使用defineProps定义组件的Props
  + 用于组件间通信
    - 在Vue组件中，父组件可以通过 props 向子组件传递数据。props 是组件的属性，可以包含任何类型的数据，例如字符串、数字、对象、数组等。子组件接收这些 props 并可以在其模板中使用它们。
  + defineProps 是一个用于定义组件 Props 的函数。它用于声明组件接受的 Props，并进行类型检查和默认值设置。
  + 在setup函数中/组合式api的script 标签带有setup属性里使用
* 声明Props的类型
  + 用ts中的接口进行定义类型的结构
    - 在 TypeScript 中，接口（interface）是一种用于定义对象的类型的抽象结构。
    - 用于描述类似结构的对象，以确保它们符合某种约定或规范。
    - 接口可以包含属性、方法、索引签名等，它们可以被其他类型实现或扩展。当一个对象实现了某个接口时，它必须包含该接口中定义的所有属性和方法。
    - 接口在 TypeScript 中常用于类型检查和代码约定，可以帮助开发者更好地理解代码，减少错误。

#### **1.3.2.13.搜索框组件**

* 使用 defineEmits 定义组件的事件
  + 用于子组件向父组件传递消息，在父组件中，只需要监听子组件的自定义事件，然后执行相应的逻辑即可。
  + 在typescript环境下进行定义组件的事件，使用interface接口来 描述事件参数类型的结构
  + 使用 defineEmits 返回已定义的事件
* 如何定义组件的 v-model
  + v-model 本质上不过是语法糖。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。
    - 在大部分情况下， v-model="foo" 等价于 :value="foo" 加上 @input="foo = $event"；
  + v-model 是双向绑定，官方说是。
  + 单向数据流
    - v-model 是单向数据流的典型范式。
    - .sync 修饰符是单向数据流的另一个典型范式。
    - 什么是单项数据流？
      * 子组件不能改变父组件传递给它的 prop 属性，推荐的做法是它抛出事件，通知父组件自行改变绑定的值。
      * 数据向下，事件向上
  + 在子组件的vue文件中，声明父组件要向子组件传递的prop属性
  + 在子组件的vue文件中，定义用于向上派发事件emit，在子组件中触发
* 如何使用 CSS 变量
  + 组件的模板中
    - 使用了 :style 绑定动态设置了一个 --bg-color 的 CSS 变量
  + 组件的数据
    - bgColor 来设置这个变量的值
  + 在样式中
    - 使用了 var() 函数来引用这个变量，从而实现了动态的背景颜色。
* BEM 命名规范
  + BEM是块(block) 、元素(element) 、修饰符(modifier)的简写
  + 是一种用于命名 CSS 类的方法
    - - 中划线
      * 仅作为连字符使用,表示某个块或者某个子元素的多单词之间的连接记号
    - \_\_ 双下划线
      * 双下划线用来连接块和块的子元素
    - -- 双中划线
      * 双中划线用来描述一个块或者块的子元素的一种状态
  + 旨在创建可维护、可重用的样式表。

#### **1.3.2.14.自定义hooks-useToggle实现搜索页展示切换**

* hooks介绍
  + hooks 就是将去改变一个参数值时，页面也会更新对应的值的想法、抽象，用代码实现的地方
* 如何实现一个hooks
  + 在src（source）目录下创建一个use文件夹用来存放hooks文件
  + 在hooks文件中创建一个hooks
    - 写一个函数并用export来导出给对应的组件使用。
    - 该函数是用来接收和返回相关hooks的属性值或者函数、实例的
    - hooks 函数需要返回一个对象，对象中可以包含一些响应式的数据和一些方法。
* 实现跨组件通信方式
  + 事件传递
    - 理解
      * 事件传递实现跨组件通信, 流程是先创建一个hooks 用来进行参数传递，在 父组件中 针对子组件a 和 子组件b中的 子组件 c 进行跨组件通信
    - 在组件中使用hooks实现 事件传递实现跨组件通信
      * 获取hooks 中返回的属性值或者函数、实例，用来传递变更的值
      * 事件传递实现跨组件通信, 流程是先创建一个hooks 用来进行参数传递，在 父组件中 针对子组件a 和 子组件b中的 子组件 c 进行跨组件通信
      * 在组件 a中定义cancel事件来获取 hooks 传递的值
      * 在组件 b 中定义searchClick事件来获取 hooks 传递的值

#### **1.3.2.15.searchView组件开发**

* 使用<transition>和<transition-group>实现动画效果
  + Vue 中的 <transition> 和 <transition-group> 组件是基于 CSS 过渡和动画的
  + 利用了 CSS 的 transition 和 animation 属性来实现动画效果。
  + 组件通过在元素进入或离开 DOM 结构时，自动应用 CSS 类来触发过渡效果。
    - 进入和离开过渡：
      * 当元素在 DOM 中插入或从 DOM 中移除时，Vue 会检测到这些变化，并在相应的时间点自动添加或删除特定的 CSS 类名，从而触发过渡效果。
      * 过渡过程中会根据设置的 CSS 过渡属性（例如 transition）来控制动画的持续时间和效果。
    - 过渡类名：
      * Vue 会根据以下情况在元素上添加或移除相应的类名：  
        v-enter：元素刚刚被插入时的状态。  
        v-enter-active：元素正在进入时的状态，通常设置了过渡效果。  
        v-enter-to / v-enter-from：在进入过渡的结束状态（如果 CSS 过渡效果有终止状态的话）。  
        v-leave：元素被移除时的状态。  
        v-leave-active：元素正在离开时的状态，通常设置了过渡效果。  
        v-leave-to / v-leave-from：在离开过渡的结束状态（如果 CSS 过渡效果有终止状态的话）。
    - 过渡完成事件：
      * 过渡完成后，Vue 会触发相应的事件（例如 @before-enter、@enter、@after-enter、@before-leave、@leave、@after-leave），您可以利用这些事件来执行其他操作。
    - 动态过渡：
      * Vue 还允许您动态地根据状态切换过渡效果。您可以根据条件来控制是否应用过渡效果，或者为不同的状态指定不同的过渡效果。
    - <transition-group> 中的列表过渡：
      * <transition-group> 组件专门用于处理列表渲染时的过渡效果，当列表发生变化时（如元素添加或删除），它会为每个元素应用适当的过渡类名。
  + Vue 的过渡效果是通过在 DOM 元素上动态添加 CSS 类名来实现的，而这些类名的变化触发了 CSS 中设置的过渡效果，从而实现了动画。
* 组件复用实现方法
  + Props
    - 使用 Props 机制，父组件通过 Props 将数据传递给子组件，子组件通过接收 Props 来获取数据。这样可以使得子组件可以接受不同的数据，从而实现了复用。
  + Slots
    - Slots 允许父组件向子组件插入内容，从而实现了组件的复用。
      * 具名插槽可以使父组件传递具名内容到子组件中的指定位置，
      * 作用域插槽则允许子组件将自己的数据传递给父组件进行处理。
  + Mixins
    - 理解
      * 其实Mixin不是Vue专属的，可以说它是一种思想，也可以说它就是混入的意思，在很多开发框架中都实现了Mixin(混入)，我们这里主要讲解的是Vue中的Mixin。
      * 民间解释
        + 将组件的公共逻辑或者配置提取出来，哪个组件需要用到时，直接将提取的这部分混入到组件内部即可。
        + 这样既可以减少代码冗余度，也可以让后期维护起来更加容易。
        + 这里需要注意的是：

提取的是逻辑或配置，而不是HTML代码和CSS代码。

其实大家也可以换一种想法，mixin就是组件中的组件，Vue组件化让我们的代码复用性更高，那么组件与组件之间还有重复部分，我们使用Mixin在抽离一遍。

* + - * 官方解释
        + 混入 (mixin) 提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。
        + 一个混入对象可以包含任意组件选项。
        + 当组件使用混入对象时，所有混入对象的选项将被“混合”进入该组件本身的选项。
  + Composition API
    - Vue 3的Composition API，可以灵活地组织和复用组件逻辑，将相关的代码逻辑封装成自定义函数或组合函数，然后在需要的组件中进行调用。
    - 可以更灵活地组织和复用组件逻辑，使得代码更易于维护和重用。
* computed 计算属性
  + 当一个计算属性依赖的响应式数据发生变化时，Vue会检测到这一变化并重新计算这个计算属性的值。
  + 这种依赖追踪的机制
    - 是通过底层的响应式系统实现的，
    - 它使用了JavaScript的Proxy对象来拦截对响应式数据的访问，以便在数据发生变化时触发相应的更新。
* watch 监听属性
  + 使用watch函数来监听state对象中width和height属性的变化。当这些属性的值发生变化时，相应的回调函数将被调用。
  + 基于JavaScript的Proxy对象来实现的。
    - 在Vue 3中，Proxy对象被用于拦截对响应式数据的访问，并且Vue会在数据发生变化时触发相应的回调函数。
* 使用axios实例发送业务请求
  + 创建了一个 Axios 实例，并设置了一些默认配置，例如 baseURL 和 timeout。然后我们使用这个实例发送了一个 GET 请求到 /endpoint 路径，然后处理响应数据或错误。
* mock请求
  + 是指在开发和测试过程中模拟后端服务的行为，以便在没有实际后端服务的情况下进行前端开发和测试
  + 定义模拟数据结构
  + 拦截请求
  + 返回模拟数据
  + 处理特定场景
* v-if
  + 根据表达式的真假条件来有条件地渲染一段 HTML
  + 使用方式是将该指令绑定到一个布尔值的表达式上，当该表达式为真时，对应的 HTML 元素会被渲染，而当该表达式为假时，则不会渲染。
  + 原理
    - v-if 指令的原理涉及到Vue.js的虚拟DOM (Virtual DOM) 和响应式系统。
    - 当 v-if 表达式为真时，Vue.js会创建相应的DOM节点并插入到页面中，反之则会将对应的DOM节点从页面中移除。
    - Vue.js会监听 v-if 表达式所依赖的数据，当数据发生变化时，会重新计算表达式的值，并根据新的值决定是否对应地渲染或移除DOM节点。
  + v-if 和 v-show 的区别
    - v-if 是“真正的”条件渲染：如果条件为假，Vue.js 不会渲染元素。
    - v-show 则是简单地基于 CSS 的 display 属性来控制元素的显示与隐藏，始终会渲染元素，只是在条件为假时将其隐藏。
* v-for
  + 用于循环渲染数组或对象的数据，生成对应数量的 HTML 元素。
  + 使用方式是将该指令绑定到一个数组或对象上，然后使用特定的语法来定义循环体的内容。
  + 原理
    - v-for 指令的原理涉及到 Vue.js 的虚拟 DOM 和响应式系统。
    - 当遇到带有 v-for 指令的元素时，Vue.js 会将其作为一个循环块进行处理。
    - Vue.js 监听数据源（数组或对象）的变化，当数据源发生变化时，Vue.js 会重新计算循环块，并根据新的数据源生成新的虚拟 DOM 节点，最终更新到页面中。
    - 同时，Vue.js 也会根据提供的唯一键值（key）来尽可能地复用现有的元素，以提高性能。
  + 使用 v-for 进行循环渲染时，一定要为每个循环项指定一个唯一的键值（key），以帮助 Vue.js 更高效地管理循环块的更新。

#### **1.3.2.16.性能优化-实现useDounce解决请求抖动问题**

* 理解
  + 请求抖动（Request Throttling）是指在某些情况下，请求频率过高，服务器无法及时处理所有的请求，导致系统性能下降或服务不稳定的问题。
  + 解决请求抖动问题的一种常见方法是通过请求节流（Request Throttling）或请求限流（Request Rate Limiting）来控制请求的频率。
* 请求节流的实现原理
  + 限制请求的发送频率，确保在指定的时间间隔内只发送一定数量的请求。
  + 这样可以有效地减轻服务器的压力，提高系统的稳定性。
  + 常见的请求节流方法
    - 基于时间间隔的节流
      * 在一定的时间间隔内只允许发送一次请求。可以使用计时器（setTimeout 或 setInterval）来实现，或者使用 debounce 函数库来简化操作。
    - 基于请求次数的节流
      * 在一定的请求数量内只允许发送一定数量的请求。
      * 可以使用计数器来统计已发送的请求次数，并在达到限制后暂时阻止后续的请求。
    - 基于队列的节流
      * 将所有的请求按顺序排列在一个队列中，然后按照指定的频率从队列中取出请求并发送。
    - 基于令牌桶算法的节流
      * 令牌桶算法是一种流量控制算法，它通过维护一个固定容量的令牌桶来限制请求的发送频率。
      * 每当接收到一个请求时，都会从令牌桶中取出一个令牌，如果令牌桶中没有足够的令牌，则请求将被暂时阻塞或延迟处理。
    - 基于滑动窗口的节流
      * 滑动窗口是一种动态调整窗口大小的方法，可以根据实际的请求情况来动态调整请求的发送频率，以适应不同的网络环境和服务器负载。
* 使用 watch 函数监听了一个搜索关键词的变化实现基于时间间隔的节流解决请求抖动问题
  + 当搜索关键词发生变化时，我们使用定时器在一定时间（比如500毫秒）后发送请求。
  + 如果在这个时间间隔内搜索关键词又发生了变化，我们会取消之前的定时器，并重新设置一个新的定时器。
  + 这样就实现了请求的节流，避免了请求抖动问题。

#### **1.3.2.17.自定义hooks-useAsync实现请求处理**

* Promise then 和 catch 的处理
  + 使用 ref 来创建了三个响应式变量：loading 用于表示请求是否正在进行中，error 用于表示请求是否出错，data 用于保存请求成功后的数据。
  + 定义了一个 fetchData 函数来模拟发送异步请求
  + 通过 watch 监听了 loading 和 error 的变化，以处理请求的状态。
  + 调用 fetchData 函数来发送请求
  + 返回需要暴露给组件的数据和方法
* TS 声明复杂的类型结构
* 使用 jsonserver 中间件延时返回数据
  + setTimeout实现延时
    - setTimeout 是 JavaScript 中的一个函数，用于在一定的时间间隔之后执行指定的代码。
    - 当调用 setTimeout 函数时，JavaScript 引擎会创建一个计时器（timer）
    - 并将指定的回调函数和延迟时间保存起来。
    - javaScript 引擎会在延迟时间过后将该回调函数加入到任务队列中。
    - 一旦 JavaScript 的主线程空闲，并且任务队列中没有其他任务，JavaScript 引擎会从任务队列中取出 setTimeout 中指定的回调函数，并执行该函数。
    - 需要注意的是，尽管 setTimeout 的延迟时间是指定的毫秒数，但实际上并不能保证在延迟时间之后立即执行回调函数。
      * 因为 JavaScript 是单线程的，
      * 主线程可能会受到其他任务的影响而延迟执行。
      * 另外，延迟的时间也不是绝对准确的，可能会有一定的误差。
    - setTimeout 的实现原理是通过在延迟时间之后将指定的回调函数加入到任务队列中，等待 JavaScript 主线程执行。

#### **1.3.2.18.Grid网格布局**

* 理解
  + 将页面划分为行和列，并在网格中放置元素
* 基本用法
  + 创建一个网格容器
    - .container {  
       display: grid;  
      }
  + 定义行和列的大小
    - .container {  
       display: grid;  
       grid-template-rows: 100px 200px; /\* 定义两行，高度分别为 100px 和 200px \*/  
       grid-template-columns: 1fr 2fr; /\* 定义两列，宽度比例为 1:2 \*/  
      }
      * 通过 grid-template-rows 和 grid-template-columns 属性定义了行和列的大小
  + 放置元素到网格中
    - .item {  
       grid-row: 1 / 2; /\* 元素跨越第一行 \*/  
       grid-column: 2 / 3; /\* 元素跨越第二列 \*/  
      }
      * 通过 grid-row 和 grid-column 属性将元素放置到网格中的指定位置。
* flex布局
* 兼容性好的布局方式
  + 表格布局（Table Layout）：
    - 兼容的 IE 版本范围：所有版本的 IE 都支持表格布局
  + 浮动布局（Float Layout）：
    - 兼容的 IE 版本范围：所有版本的 IE 都支持浮动布局。
  + 绝对定位布局（Absolute Positioning）：
    - 兼容的 IE 版本范围：所有版本的 IE 都支持绝对定位布局。
  + 行内块级元素布局（Inline-Block Layout）：
    - 兼容的 IE 版本范围：IE 8 及更高版本支持行内块级元素布局。

#### **1.3.2.19.滚动提示栏组件开发**

* withDefaults 设置 props 默认值
  + withDefaults
    - 是一个工厂函数，用于创建一个具有默认值的响应式对象。它可以用于创建可重用的组件选项，并且可以通过给它传递不同的默认值来创建不同的配置。
    - 实现原理是将传入的配置对象与默认配置对象进行合并，并返回一个新的配置对象。如果传入的配置对象中包含与默认配置对象相同的属性，则会覆盖默认配置对象中的相应属性。否则，将使用默认配置对象中的属性。
    - 根据需要自定义Vue组件库的配置，而不必手动创建一个完整的配置对象。这使得对Vue组件库进行定制和扩展变得更加方便。
* ref获取DOM
  + 创建一个 ref 变量，并将其初始化为 null
  + 在模板中，将 ref 变量绑定到需要获取的 DOM 元素上，使用 v-bind 或简化的 :ref 语法
  + 在组件的 JavaScript 部分，可以通过访问 value 属性来获取对应的 DOM 元素
  + 原理
    - 创建响应式引用
      * 使用 ref 函数创建一个响应式的引用对象，初始值为 null。
    - 绑定引用对象
      * 在模板中，使用 v-bind 或简化的 :ref 语法将引用对象绑定到需要获取的 DOM 元素上。
    - 注册引用对象
      * Vue 会将引用对象注册到组件实例的 $refs 属性中。
    - 访问引用对象
      * 通过访问引用对象的 value 属性，可以获取对应的 DOM 元素。
      * Vue 3 中，$refs 属性已被废弃，因此，不再通过 this.$refs 访问。相反，直接在 setup 函数中访问 ref 变量的 value 属性即可
* <style>使用<script setup>里的JS变量
  + 定义一个变量，使用v-bind指令来绑定样式，然后使用模板插值或计算属性来引用 <script setup> 中的变量。
  + 实现原理是通过编译器的处理和作用域的绑定。
    - 解析模板和脚本：Vue 3 的编译器首先解析组件的模板和 <script setup> 部分的脚本。
    - 收集变量依赖：编译器会收集模板中使用的变量和 <script setup> 部分中声明的变量的依赖关系。
    - 创建上下文：编译器会为模板和 <script setup> 部分创建一个共享的上下文环境，使它们可以共享变量和函数。
    - 注入变量：编译器会将 <script setup> 部分中声明的变量注入到模板中的作用域中，使模板可以访问这些变量。
    - 生成渲染函数：编译器根据模板和 <script setup> 部分生成最终的渲染函数。
* v-html 渲染 html 模板
  + 使用 v-html 指令来渲染包含 HTML 内容的模板。
  + v-html 指令将会将指定的 HTML 字符串作为实际的 HTML 解析并渲染到对应的 DOM 元素上。
  + Vue 使用 v-html 指令渲染 HTML 模板的原理是通过解析 HTML 字符串，并将其作为实际的 HTML 内容进行渲染。这使得你可以在 Vue 组件中动态地渲染包含 HTML 内容的模板。
    - 解析模板：Vue 的编译器解析模板，并识别到使用了 v-html 指令的元素。
    - 获取 HTML 字符串：Vue 在组件实例中查找指定的数据属性（如 htmlTemplate），该属性应包含待渲染的 HTML 字符串。
    - 创建虚拟 DOM：Vue 创建一个虚拟 DOM 对象，表示 v-html 指令所在的 DOM 元素。
    - 渲染 HTML 内容：Vue 将 HTML 字符串作为实际的 HTML 解析，并将解析后的 DOM 结构插入到虚拟 DOM 对象中。
    - 更新 DOM：Vue 将更新后的虚拟 DOM 对象与实际的 DOM 进行比较，并仅更新需要更新的部分，以确保页面的高效渲染。
  + 使用 v-html 指令时要小心潜在的安全风险。由于直接解析并渲染 HTML 字符串，存在跨站脚本攻击（XSS）的风险。
* tansform: scale 实现小于 1px 的边线
  + 创建一个包含边线的容器元素
  + 在 CSS 中设置容器元素的样式，包括边线的样式
    - 使用 ::before 伪元素来创建边线
    - 设置了边线的样式为红色的 1px 实线，并使用 transform: scale(0.5) 将边线缩小为原始大小的一半。
  + 缩小边线可能导致边线的细节变得模糊或不清晰，因为浏览器对小于 1px 的绘制可能会进行舍入或模糊处理。这取决于具体的浏览器和设备。
  + 在使用 transform: scale 缩小边线时，应仔细考虑边线的清晰度和可读性，并在实际应用中进行测试和调整。
* ScrollBar轮播组件的实现原理
  + 创建容器
  + 设置样式
  + 创建滚动条
  + 设置滚动条样式
  + 监听滚动事件
  + 更新滚动条位置
  + 处理用户交互

#### **1.3.2.20.countDown倒计时组件**

* 超出的文本内容省略样式
  + text-overflow：ellipsis
* padStart()
  + 将字符串的长度填充到指定的目标长度。
  + 如果目标长度大于字符串的当前长度，则在字符串的开头插入指定的填充字符，直到字符串达到目标长度。该方法返回填充后的新字符串。
  + const str = 'Hello';  
    const paddedStr = str.padStart(10, '0');  
      
    console.log(paddedStr); // "00000Hello"
    - 将字符串 "Hello" 使用 padStart() 方法填充到目标长度为 10，填充字符为 "0"。
    - 由于字符串 "Hello" 的长度小于目标长度，因此在字符串的开头插入了 5 个填充字符 "0"，
    - 最终返回了填充后的字符串 "00000Hello"。
  + 需要注意的是，padStart() 方法在 ECMAScript 2017（ES8）中被引入，因此在使用该方法时，要确保目标环境支持该方法或使用适当的兼容性处理。

#### **1.3.2.21.自定义hooks-useCountDown实现倒计时逻辑**

* 使用requestAnimationFrame (或setTimeout)计时
  + 用于在浏览器中执行动画的 JavaScript 方法
    - 实现原理是利用浏览器的重绘机制
    - 以更有效的方式执行动画并避免过多占用 CPU 资源。
    - 会与浏览器的绘制周期同步
    - 确保动画更新在浏览器准备好绘制新帧之前完成。
    - 会在页面不可见或最小化时暂停动画，节省资源。
  + 为什么setInterval/setTimeout不准?
    - 时间不准确
      * setInterval 和 setTimeout 是基于时间间隔执行代码的。
      * 由于 JavaScript 是单线程的，当代码执行时间超过设定的时间间隔时，它们不会等待上一次执行完成，而是立即执行下一次。
      * 可能导致动画的速度不稳定，甚至产生跳帧或卡顿。
    - 不与浏览器渲染同步
      * 不考虑浏览器的重绘周期
      * 可能会在浏览器准备好绘制新帧之前执行动画更新。
      * 可能导致动画撕裂或不流畅。
    - 不考虑页面可见性和性能
      * 不会考虑页面是否可见，它们会持续运行即使页面最小化或不可见。
      * 这会浪费资源并可能降低电池寿命。
* useCountDown 的设计
  + start 方法：开始计时
  + pause 方法：暂停计时
  + reset 方法：重置时间
  + current 变量：当前时间
* 性能优化:毫秒级/非毫秒级更新
  + 性能优化:是否需要毫秒级的计时
  + 当前剩余时间 是否与 当前时间 为同一秒

#### **1.3.2.22.swipe轮播图组件开发**

* 使用provide/inject实现跨组件间通信（爷孙传值）
  + Vue 3使用了基于代理的响应式系统和更先进的数据结构来实现provide 和 inject。
  + provide 实现原理
    - 当父组件调用 provide 时，Vue 3会在父组件实例的内部创建一个响应式的引用
    - 其中存储了提供的数据。
    - 这些数据会被存储在内部的 provide 选项中
    - 并在组件树中传播。
    - 在传播过程中，Vue 3使用了 Symbol 类型的标识符来保证数据的唯一性和可靠性。
  + inject 实现原理
    - 在子组件中调用 inject 时，Vue 3会在组件树中向上搜索父级组件的 provide 选项，并获取提供的数据。
    - 然后，Vue 3会使用代理对象将这些数据注入到子组件实例中。
    - 这样一来，子组件就可以在其作用域中访问到父组件提供的数据了
    - 并且当父组件的数据发生变化时，子组件中的数据也会相应地更新。
* getCurrentInstance获取组件实例
  + 应用场景
    - 在自定义函数式组件中、或者在生命周期钩子之外的地方访问组件实例等。
    - 函数返回一个包含当前组件实例信息的对象，你可以在其中访问组件的 props、attrs、emit 等属性，以及组件实例的生命周期钩子方法等。
  + 实现原理
    - 涉及 Vue 3 内部的组件实例管理机制。
    - Vue 3 中使用了基于代理的响应式系统，组件实例也是通过代理对象进行管理的。
    - 当组件被创建时，Vue 3 会创建一个代理对象来代表该组件实例
    - 并将其存储在内部的实例管理器中。
    - 在组件的生命周期中，这个代理对象会被用于访问和修改组件的数据、方法等内容。
    - 通过一些内部的机制来获取当前正在执行的组件实例的代理对象，并返回给调用者。
    - 就可以在任意地方通过 getCurrentInstance 来获取当前组件的实例信息，以便于进行相关操作。
* 实现touch事件（触碰滚动）封装
  + touchStart：触碰开始
  + touchMove：触碰移动
  + touchEnd: 触碰结束
  + ref(), onMounted, onUnmounted
    - 在组件挂载后，通过 ref 将 DOM 元素赋值给引用对象
    - 访问引用对象的 value 属性，即可获取 DOM 元素的引用
* 实现事件监听
  + 可以基于钩子函数（onMounted ，onBeforeUnmount ）中使用documentEvent监听事件实现
* 自定义封装生命周期
  + 用已有生命周期进行组合封装
* .tsx写法
  + 理解
    - TSX 是 TypeScript JSX 的一种写法，结合了 TypeScript 和 JSX 的语法特性，用于编写基于 TypeScript 的 JSX 语法的代码。
    - JSX
      * 理解
        + 就是将 HTML 标记嵌入到 JavaScript 代码中，以便更直观、更方便地描述 UI 结构和交互行为。
        + 通过 JSX，可以使得编写组件和构建用户界面变得更加简洁、高效和可维护。
      * 是一种在 JavaScript 中嵌入 XML（或类似 XML 的语法）的语法扩展
      * 在 JavaScript 代码中可以直接书写类似 HTML 的标记，方便地描述 UI 结构。
      * 类似 HTML 的标记
        + JSX 允许在 JavaScript 中编写类似 HTML 的标记，包括 <div>、<p>、<span> 等。这使得描述 UI 结构更加直观和易读。

const element = <div>Hello, world!</div>;

* + - * JavaScript 表达式
        + JSX 中可以嵌入 JavaScript 表达式，用花括号 {} 包裹。这样可以在 JSX 中动态地插入变量、函数调用等。

const name = 'John';  
const element = <div>Hello, {name}!</div>;

* + - * 属性绑定
        + 在 JSX 中可以通过使用属性来向组件传递数据，属性名和属性值通过等号 = 进行绑定。

const color = 'red';  
const element = <div style={{ color: color }}>Hello, world!</div>;

* + - * 事件处理
        + 在 JSX 中可以通过给标记添加事件处理函数来响应用户的操作，事件名使用驼峰命名法。

function handleClick() {  
 console.log('Button clicked!');  
}  
  
const element = <button onClick={handleClick}>Click me</button>;

* + - * JSX 元素
        + JSX 语法会被转译成对应的 JavaScript 代码

const element = React.createElement('div', null, 'Hello, world!');

* + - 使用 TSX（TypeScript JSX）的写法与在 React 中类似。
  + 适合习惯按逻辑顺序编写代码，html/css/js 交叉编写顺序
  + 可以通过js动态生成组件
* .vue写法
  + 理解
    - 基于组件的架构，让开发者可以使用声明式的语法来构建用户界面。
    - 组件化开发
      * Vue.js 的核心思想是组件化开发
      * 即将 UI 划分为多个独立、可复用的组件，每个组件包含自己的模板、逻辑和样式。
      * 这种组件化的开发模式使得代码更易于维护、重用和测试
    - 数据驱动视图
      * Vue.js 使用了响应式数据绑定的方式来实现数据和视图之间的同步。
      * 当数据发生变化时，视图会自动更新，而无需手动操作 DOM。
    - 声明式模板语法
      * Vue.js 提供了一种声明式的模板语法
      * 使得在模板中可以直接使用 JavaScript 表达式来渲染数据。
    - 指令和事件处理
      * Vue.js 提供了一系列内置指令和事件处理机制，
      * 用于实现常见的 DOM 操作和事件处理。
      * 例如，v-bind 用于绑定属性，v-on 用于监听事件。
    - 生命周期钩子
      * Vue.js 组件生命周期钩子提供了一些回调函数
      * 允许开发者在组件的不同阶段执行代码。
  + 适合习惯先定义html/css，再写逻辑js

#### **1.3.2.23.内置特殊元素**

* <compent>
  + 元素用于动态地渲染不同的组件。
  + 可以通过 :is 属性来指定要渲染的组件
  + 例如 <component :is="componentName">
    - 其中 componentName 是一个动态的组件名称。
* <slot>
  + 元素可以在组件中定义可插入的内容，并在父组件中进行替换
* <template>
  + 元素则可以用来包裹多个元素或者作为条件渲染和循环的容器。
* 特殊属性
  + is
    - 动态组件的名称可以是组件的实例或者组件的选项对象。
    - 使用动态组件时，可以通过设置keep-alive来缓存组件，以提高性能。
  + key
    - 使用v-for进行列表渲染时，key是必需的。
    - 帮助 Vue 跟踪每个元素的身份，以便在添加、删除或重新排序时高效地更新 DOM。
  + ref
    - 允许我们在模板中通过一个唯一的标识符引用一个元素或组件，并且可以在 JavaScript 中进行访问。
      * 使用ref="paragraph"为<p>元素添加了 ref 属性，并使用const paragraph = ref(null)在 JavaScript 中创建了一个响应式引用。
      * 使用ref属性需要在 Vue3 中导入ref函数，通过ref(value)将一个普通的数据转化为响应式引用。
      * 同时，在 JavaScript 中通过访问value属性可以获取到实际的 DOM 元素。
  + :属性名="动态值" 的语法
    - 在 Vue 3 的模板中，: 是一个特殊的前缀，用于绑定属性的动态值。
      * 使用 :class="className" 可以动态地绑定 CSS 类名
      * 使用 :click="handleClick" 可以动态绑定点击事件处理函数。
    - 是 Vue 3 中的属性绑定写法
    - 也称为"v-bind"指令的简写形式
    - 用于将一个表达式的值绑定到组件或元素的属性上。
      * 使用 :text="message" 将数据属性 message 的值绑定到 <p> 元素的 text 属性上。当数据属性 message 的值发生变化时，<p> 元素的文本内容也会相应地更新。

#### **1.3.2.24.sticky粘性布局**

* 理解
  + CSS 布局技术
  + 元素进行“视觉转移”，使其尽可能留在最近滚动祖先元素内。
  + 它使元素在滚动过程中可以固定在某个位置，同时在滚动到一定位置后可以解除固定。
  + 在目标区域以内，它的行为就像 position:relative;在滑动过程中，某个元素距离其父元素的距离达到sticky粘性定位的要求时(比如top：100px)；position:sticky这时的效果相当于fixed定位，固定到适当位置。
  + 可以说是相对定位relative和固定定位fixed的结合
  + 元素固定的相对偏移是相对于离它最近的具有滚动框的祖先元素，如果祖先元素都不可以滚动，那么是相对于viewport来计算元素的偏移量。
* 应用场景
  + 导航栏：将导航栏固定在页面顶部，使用户在滚动页面时可以方便地导航。
  + 工具栏：将工具栏固定在页面顶部或底部，使用户在浏览内容时可以快速访问工具。
  + 侧边栏：将侧边栏固定在页面一侧，使用户在阅读内容时可以随时查看相关信息。
* 使用条件
  + 父元素不能overflow:hidden或者overflow:auto属性。
  + 必须指定top、bottom、left、right4个值之一，否则只会处于相对定位
  + 父元素的高度不能低于sticky元素的高度
  + sticky元素仅在其父元素内生效
  + 为了确保兼容性，可以使用 CSS 属性 position: -webkit-sticky 进行降级处理。
* 实现原理
  + 通过 CSS 属性 position: sticky 实现的。
    - 粘性布局需要指定一个阈值，即 top、bottom、left 或 right 属性的值。
    - 当滚动到达指定阈值时，元素会固定在该位置，并参与正常的布局流。
    - 当滚动超过另一个阈值时，元素会恢复到正常的布局。
  + 当一个元素被设置为 position: sticky 时，它会在滚动到指定阈值时变为固定定位，一直保持在指定位置
  + 直到滚动到另一个阈值后恢复正常布局。

#### **1.3.2.25.nextTick 是一个全局方法**

* 用于在下次 DOM 更新循环结束之后执行延迟回调函数。
  + 用于在更新后操作 DOM
    - 当需要在 DOM 更新后执行一些操作时，使用 nextTick 可以确保操作发生在更新循环结束之后。这样可以保证在操作 DOM 元素时，它们已经被正确渲染到页面上。
  + 访问更新后的组件状态
    - 有时候需要在组件状态更新后执行某些操作，比如在获取某个元素的尺寸或位置之后执行其他操作。使用 nextTick 可以确保在获取组件状态之前，组件已经完成了状态的更新。
  + 触发其他依赖于 DOM 更新后的操作。
    - 有些操作依赖于 DOM 更新后的状态，比如触发一些基于 CSS 动画的效果。使用 nextTick 可以确保在触发这些操作之前，DOM 已经更新完毕，从而保证操作的准确性和可靠性。

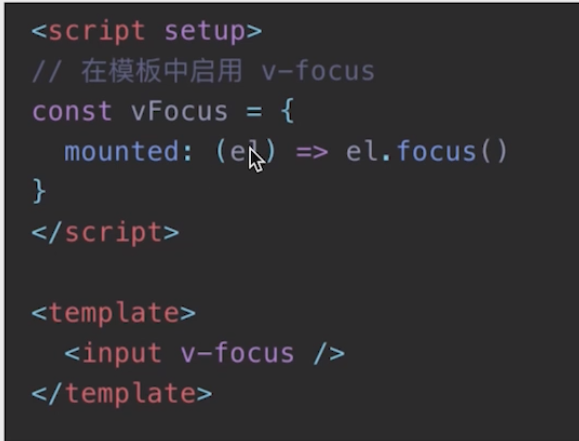
#### **1.3.2.26.unref()**

* unref实用函数
  + 如果unref()的参数是一个ref，就会返回其内部值。否则就返回参数本身。
  + const unwrappedCount = unref(count)
    - isRef(count) ? count.value : count
* 模板解包
  + 在模板上调用ref时
  + Vue会自动使用unref()进行解包。
    - 不需要在模板中使用.value进行访问
  + 只在ref是模板中的顶级属性时才生效。
    - 顶级属性
      * 在 setup 函数中使用 ref 创建了一个名为 count 的响应式对象
      * 而不是将 ref 创建的响应式对象嵌套在返回对象的其他属性中

#### **1.3.2.27..stop**

* 理解
  + 是事件修饰符之一，用于阻止事件冒泡。
    - 事件冒泡是指当一个元素上的事件被触发时，事件会从该元素开始向上冒泡至其父级元素，直到达到文档根元素。
  + 通过使用 .stop 修饰符，我们可以阻止事件继续向上冒泡，即停止事件传播。
* 应用场景
  + 阻止事件冒泡
* 实现原理
  + Vue 3 在事件处理函数中使用了 event.stopPropagation() 方法来停止事件冒泡。
  + 当事件绑定了 .stop 修饰符后，Vue 3 在生成事件处理函数时会自动添加对 event.stopPropagation() 方法的调用，从而阻止事件继续向上冒泡。

#### **1.3.2.28.性能优化--IntersectionObserver实现图片懒加载指令directive**

* 图片懒加载的介绍以及实现原理
  + 理解
    - 一般跟分页加载、整屏加载有关
    - 图片在屏幕不可见区域
    - 图片被滚动到可见区域的时候，才去加载图片资源
    - 性能优化
      * 图片懒加载可以有效地节省网络带宽和服务器资源，特别是在网页中包含大量图片时，能够显著减少页面加载所需的带宽和服务器压力。
  + 实现原理
    - 监听事件
      * 使用 JavaScript 监听页面滚动事件或者使用 IntersectionObserver API 监听图片元素进入可视区域的事件。
    - 触发加载
      * 当监听到图片进入可视区域时，通过修改图片的 src 属性或者通过 AJAX 请求来加载图片资源。
    - 懒加载库
* IntersectionObserver API 介绍
  + 理解使用
    - 是现代浏览器提供的一种用于监测目标元素与其祖先元素或顶级文档视窗（viewport）交叉状态的 API。
      * 交叉状态
        + 指的是目标元素与其祖先元素或顶级文档视窗（viewport）之间的相对位置关系，
        + 包括目标元素完全进入视窗、部分进入视窗或完全离开视窗等情况
    - 提供了一种异步观察目标元素的方法
      * Intersection Observer API 是异步执行的，它不会在每次交叉状态发生变化时立即通知观察者
      * 而是在交叉状态变化后的下一个事件循环中进行通知。
      * 这样可以减少浏览器的主线程压力，提高性能。
    - 可以在不影响主线程的情况下检测元素的可见性变化
    - 使用
      * 创建了一个 IntersectionObserver 对象 observer
      * 获取目标元素
      * 将元素注册到observer中
      * 在 observer中观察，监听元素进入视窗时机
      * 加载元素内容
  + 实现原理
    - 涉及到浏览器的渲染引擎和事件系统。
    - 注册观察器
      * 通过 new IntersectionObserver() 构造函数创建一个观察器对象，并传入一个回调函数和观察器的配置参数。
    - 观察目标元素
      * 使用观察器对象的 observe() 方法来注册要观察的目标元素。一旦目标元素的交叉状态发生变化，观察器对象会调用预先设定的回调函数。
    - 监测交叉状态
      * 浏览器的渲染引擎会在适当的时机（例如每次重新计算布局或者滚动事件结束后）检查观察器所观察的目标元素与其祖先元素或顶级文档视窗的交叉状态，并记录下来。
    - 触发回调函数
      * 如果目标元素的交叉状态发生变化，观察器对象会在下一个事件循环中异步调用预先设定的回调函数，并传递交叉状态的相关信息给回调函数。
  + 应用场景
    - 图片懒加载
      * 可以通过监听图片元素与视窗的交叉状态来延迟加载图片资源，提高页面加载性能。
    - 无限滚动
      * 可以在滚动加载更多内容时触发加载事件，从而实现无限滚动列表。
    - 监测广告曝光
      * 可以监测广告元素与视窗的交叉状态，从而统计广告曝光和点击次数等。
* Vue3 指令Direction以及插件plugin机制的介绍
  + 指令
    - 理解使用
      * 带有 v- 前缀的特殊属性
      * 用于在模板中为 HTML 元素添加特殊行为或功能。
      * 指令可以直接作用于 HTML 元素，用于对元素进行操作或绑定相应的事件处理逻辑。
      * 指令可以定义钩子函数，用于在定义的生命周期中执行特定的逻辑。
      * 使用
        + 
    - 应用场景
      * 操作 DOM
        + 可以通过指令来直接操作 DOM 元素，例如修改样式、添加类名、设置属性等。
      * 事件处理
        + 可以通过指令来绑定事件处理函数，例如 v-on 指令用于绑定事件，监听用户的交互操作。
      * 自定义行为
        + 可以通过指令来实现特定的行为，例如实现图片懒加载、拖拽排序等功能。
    - 实现原理
      * 主要涉及到 Vue 的编译器和运行时。
      * 编译器会解析模板中的指令，并将其转换成对应的指令描述对象。
      * 运行时则负责解析指令描述对象，并执行相应的逻辑，例如注册指令、执行钩子函数等。
      * 当 Vue 应用程序运行时，会通过 directive API 注册自定义指令。
      * 注册时会传入指令的名称和一个对象
        + 对象中包含了一系列的钩子函数
        + 用于在指令的生命周期中执行相应的操作。
      * 在模板中使用指令时
        + 指令的描述对象会被传递给运行
        + 运行时会根据指令描述对象执行相应的逻辑。
  + 插件机制
    - 理解使用
      * 能在 Vue 实例上注册全局功能、添加全局方法或者修改 Vue 的原型。
      * 插件是一种用于扩展 Vue 功能的机制
      * 注册插件
        + Vue 3 提供了 app.use() 方法用于注册插件
        + 通过这个方法可以将插件安装到 Vue 实例中，从而扩展 Vue 的功能。
      * 使用
        + 在 Vue 3 中，注册插件的方式是通过 app.use() 方法，传入要注册的插件。
    - 应用场景
      * 全局方法
        + 可以将一些常用的方法添加到 Vue 的原型上，从而在组件中全局可用，例如常见的 $http、$router、$store 等方法。
      * 全局组件
        + 可以将一些通用的组件注册为全局组件，从而在任意组件中都可以使用，例如常见的 Loading、Modal、Toast 等组件。
      * 插件功能
        + 可以添加一些 Vue 的全局功能，例如添加指令、混入（mixin）、过滤器等，从而增强 Vue 的功能和灵活性。
    - 实现原理
      * 主要是通过 app.use() 方法和插件对象中定义的 install 方法。
      * 当调用 app.use() 方法注册插件时，Vue 3 会自动调用插件对象中的 install 方法
        + 全局方法和属性的添加

通过 app.config.globalProperties 添加全局方法或属性，这些方法或属性将会添加到 Vue 实例上，使得它们在整个应用中可用。

* + - * + 添加全局组件

使用 app.component() 方法注册全局组件，这样该组件将在整个应用中可用。

* + - * + 添加全局指令

使用 app.directive() 方法注册全局指令，使得该指令在整个应用中可用。

* + - * + 添加混入（mixin）

使用 app.mixin() 方法添加全局混入，以便在所有组件中共享一些逻辑。

* + - * + 添加插件

使用 app.use() 方法安装其他 Vue 插件，使得这些插件在整个应用中可用。

* + - * 并将 Vue 实例作为参数传递给 install 方法。
      * install 方法可以接收 Vue 实例作为参数，从而可以在插件中访问 Vue 实例的各种属性和方法，进而实现对 Vue 的扩展功能。

#### **1.3.2.29.自定义hooks-useAuth实现登录页面逻辑**

* 介绍整体登录的逻辑
  + 前端在用户登录时useAuth的login函数给后端发送登录请求
  + 后端根据用户账号密码登信息，校验成功后,返回用户信息以及对应token
  + 前端拿到用户信息和token后,使用useUserStore和useLocalStorage分别保存到store和localStatorage中
* 登录页实现
  + 输入账号密码信息
  + 调用useAuth返回的方法来请求登录接口server
  + 后端项目里实现 token 的生成和校验逻辑
* useAuth 的实现
  + userInfo
    - useUserStore
    - useLocalStorage
  + login
  + logout

#### **1.3.2.30.Pinia**

* 理解使用
  + Vue.js 状态管理库
    - 主要用于管理 Vue 应用程序中的状态
    - 可以跨组件，页面共享包括数据、状态、以及状态的变化。
  + 于 Vue 3 Composition API(组合式 API) 的状态管理库
  + 使用
    - 定义状态：使用 defineStore() 方法选择使用对象或者函数两种方式定义状态存储。
      * 使用对象方式
        + 直接将状态和操作定义在一个对象中，并将其传递给 defineStore() 方法。这种方式比较简洁，适用于定义简单的状态存储。
      * 使用函数方式（对应响应式的方法函数）
        + 通过一个函数来定义状态和操作。
        + 这种方式更加灵活，允许你在函数内部执行任意的逻辑，并返回一个包含状态和操作的对象。
        + 这样可以更好地组织和管理状态的定义和操作。
        + 在这种方式中，你可以使用任意的 Vue 3 Composition API 方法来定义状态和操作
        + 并返回一个包含状态和操作的对象。
        + 这样可以更好地组织和管理状态的定义和操作，使得代码更加清晰和可维护。
      * 区别
        + 灵活性

函数方式

更加灵活，允许你在函数内部执行任意的逻辑

并且可以使用 Vue 3 Composition API 提供的各种功能。

对象方式

相对更加简洁，适用于定义简单的状态存储。

* + - * + 可读性

函数方式

相对更加灵活

但是可能会增加代码的复杂度和理解难度。

对象方式

更加直观

因为状态和操作都定义在一个对象中，可以一目了然

* + - * + 性能

两种方式在性能上没有明显的区别，选择哪种方式主要取决于个人偏好和项目需求。

* + - 在组件中使用状态：在组件中使用状态。
* 实现原理
  + 主要是基于 Vue 3 的响应式系统和 Composition API。
  + 当你在应用程序中创建一个 Pinia 实例时，Pinia 会创建一个全局状态存储容器，
  + 用于存储和管理所有的状态存储。
  + 当你定义一个状态存储时，Pinia 会创建一个具有响应式属性的 Vue 实例
  + 并将其存储在全局状态存储容器中。
  + 当你在组件中使用状态存储时，Pinia 会将状态存储实例注入到组件的上下文中，使得你可以在组件中通过 useStore 方法来获取状态存储实例，并访问其中的状态和方法。

#### **1.3.2.31.状态管理**

* 理解
  + 通过各种机制来管理应用程序中的状态，使得状态的管理更加清晰、灵活和可维护。
* 使用
  + 使用 Vuex、Pinia 或者其他自定义的状态管理方案来管理状态。
* 应用场景
  + 全局状态管理
    - 适用于需要在整个应用程序中共享的状态，如用户登录信息、主题设置等。
  + 局部状态管理
    - 适用于只在某个组件或组件树内部使用的状态，如某个页面的数据、组件之间的通信等。
  + 异步数据处理
    - 适用于处理异步数据请求、响应等场景，如加载数据、提交表单等。
* 实现原理
  + 主要是基于 Vue 3 的响应式系统和 Composition API。
  + 当你使用状态管理库来创建一个状态存储时，它会创建一个具有响应式属性的对象或者函数，并且会将其存储到全局状态管理容器中。
  + 当状态发生变化时，状态管理库会自动触发更新视图的操作，使得视图与状态保持同步。

#### **1.3.2.32.自定义hooks-useLocalStorage-保存用户状态信息，进行持久化**

* LocalStorage
  + 理解
    - 提供了一种持久化存储方式，可以将数据存储在浏览器中
    - 并且在页面重新加载后仍然保留这些数据
    - 直到用户手动删除或者浏览器清除。
    - 存储的数据是以键值对的形式存在的，每个键值对都包含一个键和一个对应的值。
  + 使用
    - 存储数据
      * localStorage.setItem('key', 'value');
    - 读取数据
      * const value = localStorage.getItem('key');
    - 删除数据
      * localStorage.removeItem('key');
  + 应用场景
    - 本地缓存
      * LocalStorage 可以用来缓存一些用户相关的数据，如用户偏好设置、用户配置信息等。
    - 状态持久化
      * LocalStorage 可以用来保存一些应用状态，以便在用户下次访问时恢复应用的状态。
    - 用户登录信息
      * 可以使用 LocalStorage 存储用户登录信息，以便在用户重新访问页面时保持登录状态。
  + 实现原理
    - 基于浏览器提供的 localStorage 对象
    - 浏览器的存储机制。
    - 当你调用 localStorage.setItem() 方法来存储数据时，浏览器会将这些数据保存在本地的数据库中。
    - 这些数据是以键值对的形式存储的，并且在浏览器关闭后仍然保留。
    - 当你调用 localStorage.getItem() 方法来读取数据时
    - 浏览器会从本地数据库中获取对应的值，并返回给你。
  + 注意
    - LocalStorage 存储的数据是以字符串的形式存储的
    - 如果你存储的是对象或者数组等复杂数据类型，需要先将其转换为字符串再进行存储
    - 读取时需要再将其转换回原来的数据类型。

#### **1.3.2.33.$**

* 以 $ 开头的方法通常是 Vue 实例内置的方法或属性，用于处理 Vue 实例的各种功能和生命周期。
* 命名约定
  + 以 $ 开头的方法通常是 Vue 框架内部使用的，用于区分开发者自定义的方法和属性。

## **1.4.MockAPI**

### **1.4.1.技术栈**

#### **1.4.1.1.json-server**

### **1.4.2.项目文件结构**

#### **1.4.2.1.data：存放所有数据 json 文件**

#### **1.4.2.2.public：存放静态资源，比如图片**

#### **1.4.2.3.src：项目的处理逻辑**

#### **1.4.2.4.app.js：项目入口文件，包括应用创建、中间件使用router.js：处理自定义路由**

#### **1.4.2.5.db.js:处理 json-server 的路由**

#### **1.4.2.6.controller:存放 controller**

#### **1.4.2.7.service：存放 service**