### TODO： 项目实战或练习，才会记住。

### 一.环境安装

#### 1．1使用vite创建vue项目

|  |
| --- |
| # npm 6.x  npm init vite@latest <project-name> --template vue  # npm 7+, 需要加上额外的 双短横线  npm init vite@latest <project-name> **--** --template vue  cd <project-name>  npm install # 安装依赖。如果不能翻墙，使用国内淘宝等镜像  npm run dev # 运行 |

#### 1.2 项目结构



src/main.js是入口文件，它会引用src/App.vue

#### 1.3 IDE与插件

安装vs code（ctrl + ` 打开终端）

**安装Volar插件**（语法高亮、代码补全、代码格式化）---已废弃，安装Vue-official插件

安装Chinese中文界面、**Vue 3 Snippets（快速生成基础模板）**

**Vue 3 Snippets具体使用：**

**TODO：**

### 2.安装与使用ElementPlus插件

|  |
| --- |
| **#安装**  **npm install element-plus –save**  **#main.js中引入**    **# vue文件中使用element-plus中的具体组件，比如：button**      <el-button type="primary">Primary</el-button> |

### 3.使用css库

tailwind库

### 4.使用vue-router

4.1 路由

什么是路由？在前端页面上，当点击菜单栏等元素，让浏览器地址发生了变化。



什么是哈希模式？

哈希模式是利用 URL 中的哈希值（**即 URL 中#后面的部分**）来实现页面的导航和状态管理。**哈希值不会被包含在向服务器发送的请求中**，仅用于浏览器端的页面操作和状态记录。比如，https://example.com/page#section2中，#section2就是哈希值。

|  |
| --- |
| #1.安装（vue3使用vue-router 4.0以上版本）  npm install vue-router@4  #2. src下新建router目录，并新建index.js, 内容：  // 导入 创建哈希路由模式 和 创建路由 函数  import {createWebHashHistory, createRouter} from 'vue-router'  // myLoginView名字是本文件里自定义的  import myLoginView from '../page/login.vue' // 创建自己的页面  import NotFound from '../page/404.vue'  // 定义路由表  const routes = [      {          path: "/login", // 因为是哈希模式。所以用host:port/#/login访问          component: myLoginView,      } ,      {          path: '/:pathMatch(.\*)\*',          name: 'NotFound',          component: NotFound      },  ]  // 创建路由  const router = createRouter({      history: createWebHashHistory(), // 函数调用      routes  })  // 导出路由对象/实例  export default router  // 错误写法。下面是导出一个router对象的属性  // export default {  //     router // router:router的简写  // }  # 3. main.js 引用router  import router from './router/index.js'  const app = createApp(App)  app.use(router)  #4. App.vue使用 router-view  <template>   <!-- 路由视图是用来显示 当前的路由，即route中的path对应的component -->      <router-view></router-view>  </template>  #5.浏览器访问：http://localhost:5173/#/login |

4.2对src设置别名

原因：在引文文件等时候，大部分文件是在src下。起别名后，方便编程。

|  |
| --- |
| # 参考vite官方文档：#https://cn.vite.dev/config/sharedoptions.html#resolve-alias  # vite.config.js配置如下：  import path from 'path' // node中的path库  export default defineConfig({    resolve: {      alias: {        //将xxx/xxx/src设置别名为~        "~": path.resolve(\_\_dirname, "src")      }    }  }) |

### 三、登录功能

3.1 Layout布局

el-row: 将屏幕(或者元素的width) 分别24列.

er-col: 通过:span指定占用少列

|  |
| --- |
| <**el-row**>  <**el-col** **:span="12**">  <div class="grid-content ep-bg-purple" />  </el-col>  <**el-col :span="12"**>  <div class="grid-content ep-bg-purple-light" />  </el-col>  </el-row> |

3.2. flex布局

3.3 Element的 input组件中使用图标

参考：

https://element-plus.org/zh-CN/component/input.html#%E5%B8%A6%E5%9B%BE%E6%A0%87%E7%9A%84%E8%BE%93%E5%85%A5%E6%A1%86

|  |
| --- |
| # 安装图标资源  npm install @element-plus/icons-vue  # <srcipt>中导入指定图标  import { Lock, Search } from '@element-plus/icons-vue'  # input组件内使用图标 ( 通过插槽方式 )                          <el-input v-model="form.name">                              <template #prefix>                                  <el-icon class="el-input\_\_icon">                                      <search />                                  </el-icon>                              </template>                          </el-input> |

效果展示：



3.4 setup语法糖

<script setup> 中的代码会在 每次组件实例被创建的时候执行。

3.5 登录表单验证处理

表单静态验证：

|  |
| --- |
| #1. 表单设置rules属性  <el-form :model="form" ref="formRef" **:rules="rules"** >  #2. 设置rules规则  const rules = {      // key要和表单 const form =reactive(...)定义的一样   // 比如，rules里的name 和 from里定义的name 名字一样      name: [          // trigger: 'blur'，失去焦点          { required: true, message: '用户名不能为空', trigger: 'blur' },          { min: 3, max: 5, message: '用户名长度3-5位', trigger: 'blur' },      ],      pwd: [          { min: 6, max: 10, message: '密码6到10位', trigger: 'blur' },      ]  }  #3. form-item设置 prop属性，要和 rules里定义的key一样    <el-form-item label="密码" **prop="pwd"**>      <!-- 方法二： 使用 : prefix-icon 定义图标 -->      <el-input v-model="form.pwd" :prefix-icon="Lock" />    </el-form-item> |
|  |

总之： prop属性值、rules里定义的key，都要和 定义form表单这个reactive对象里的字段保持一致。

表单提交时验证：

|  |
| --- |
| #1. 设置表单的引用  <el-form :model="form" **ref="formRef"** :rules="rules" >  #2. 在提交中判断  const formRef = ref(null)  const onSubmit = () => {      formRef.value.validate(boolResult => {          if (!boolResult) {              console.log("验证失败")              return          }          console.log("验证成功")      })      console.log('submit!')  } |

3.6使用axios库调用登录接口

|  |
| --- |
| 1. 安装: npm install axios  2. 导入： import axios from ‘axios’  3. 创建实例  const instance = axios({      baseURL: 'http://127.0.0.1:8989',      timeout: 3000  })  4. 发送请求：  instance.post(  “/admin/login”, // url  {username:”123”, pwd:”456”} // data  ) |

注：axios.post是返回一个 promise对象. 用.then(res=>{}) 处理返回值,用.catch(err=>{}) 处理错误信息。

3.7 登录成功使用router.push(“/”)跳转首页

|  |
| --- |
| 1. 导入并初始化：  import { useRouter } from 'vue-router'  const router = useRouter()  2. 跳转  router.push(“/”) |

登录成功、失败后，使用EINotification提示框提示

|  |
| --- |
| 1. import {ElNotifcation} from ‘element-plus’ 2. ElNotification({   title: '错误',  message: '登录失败',  type: 'error',  duration: 2500  }) |

3.8 使用cookie保存token

|  |
| --- |
| 1. 安装： npm install vue3-cookies  2. 导入并初始化：  import {useCookies} from 'vue3-cookies'  const {cookies} = useCookies() 3. 存cookie  cookies.set(“token”, res.data.data.token) |

3.9使用axios拦截器

**请求拦截器**： 在发送请求前做点东西，比如：每个请求都需要在header放token，那么将这个逻辑统一放到请求拦截器中

响应拦截器： 在返回真正的响应前做点什么。比如：vue本身返回的结构体嵌套太深，可以直接返回response.data, 而不是返回response； 响应报错了，统一弹提示框等等。

注： 代码都可在axios官网上找到。

|  |
| --- |
| // 添加请求拦截器  instance.interceptors.request.use(function (config) {      // 在发送请求之前做些什么      // 在发送请求之前，将cooke中的token添加到请求头中      const token = cookies.get('token')      if (token) {          config.headers.Authorization = `Bearer ${token}`      }      return config;    }, function (error) {      // 对请求错误做些什么      return Promise.reject(error);    });  // 添加响应拦截器  instance.interceptors.response.use(function (response) {      // 2xx 范围内的状态码都会触发该函数。      // 对响应数据做点什么      return response.data;  // 返回服务器的响应数据，业务代码不再用response.data.data了    }, function (error) {      // 超出 2xx 范围的状态码都会触发该函数。      // 对响应错误做点什么      // 对响应错误做点什么： 弹出错误信息      ElNotification({          message: error.response.data.msg || '请求失败',          type: 'error',          duration: 3000      })      return Promise.reject(error);    }); |

3.10 封装通用函数

|  |
| --- |
| 1.将弹窗消息封装到 common/util.js 中    2. 将token操作封装到 common/cookie.js中  import {useCookies} from 'vue3-cookies'  const {cookies} = useCookies()  const TOKEN\_KEY = 'token'  export function setToken(value) {      cookies.set(TOKEN\_KEY, value)  }  export function getToken() {      return cookies.get(TOKEN\_KEY)  }  export function removeCookie() {      cookies.remove()  } |

3.11使用vuex进行状态管理

使用时机： 多个组件共享读取数据时，需要使用vuex进行管理【相当于**全局单例模式**】

|  |
| --- |
| 1. 安装   npm install vuex@next –save   1. 新建store/index.js   import { createStore } from "vuex"  const store = createStore({  // 固定叫state，相当于全局变量  **state**: {  count: 1  },  mutations: {  // 第一个参数固定是state  increment1 (**state**) {  // 变更状态  state.count++  }  // 也可以定义一个传参进来  increment2 (**state**, value) {  // 变更状态  state.count +=  }  }  })  export default store   1. main.js使用store   import store from ‘./store’  app.use(store)   1. vue文件中script修改共享变量   import {useStore} from ‘vuex const store = useStore()  store.commit(“increment2”) store.commit(“increment2”, 20)   1. vue文件中<template>访问共享变量   {{$store.**state**.user}} |

注： 不管访问还是修改，state是枢纽，不要忘记； 这个store的整体流程和router很像

3.12 router的全局前置守卫

用途： 在跳转到某个路由时，先执行下这个beforeEach前置守卫。 那么就可以通过跳转、取消跳转、重定向。

返回值：

false： 取消导航，回到from处

undefined或true： 正常跳转到to处

{name: ‘/login’}: 重定向导航（**from不变, 会再触发执行beforeEach**）

|  |
| --- |
| router.beforeEach((to, from) => {      const token = getToken()      // 1.token不存在 并且 目标不是登录页,则跳转登录页.  一定要检查to.path,防止无限重定向      if (!token && to.path !== "/login") {          return {path: '/login'}      }      // 2.token存在，用户还想到登录页登录，则取消跳转      if (token && to.path === "/login") {          return false      }      // 3.返回undefine或者true,表示正常跳转到to上  }) |

3.13 vuex状态管理优化

状态管理2大特性：

1.响应式：状态的变化（即状态数据，即store.state中的数据）会自动触发Vue组件重新渲染。

2.刷新页面数据丢失： 数据存在内存中，访问速度快，但同时刷新页面时，数据（状态数据，即store.state中的数据）会丢失。

解决刷新时用户信息丢失问题。登录成功后，将用户信息放入到localstorage中（只能放字符串，要json.stringfy下），然后在路由前置守卫中，从 Vuex 状态管理或本地存储中获取当前用户信息后， 再更新Vuex状态管理。

|  |
| --- |
| // 第一种方式报错： Cannot read properties of undefined (reading 'state'). 原因： useStore() 是一个组合式 API 钩子函数，只能在setup中使用， 不能在模块顶层使用(因为此时vuex还没有初始化)  // import {useStore} from 'vuex'  // const store = useStore()  // 第二种方式报错：The requested module '/src/store/index.js' does not provide an export named 'store'。 用{store}这种方式是命名导出。而vue项目是用默认导出 (default export)  // import {store} from '../store'  // 第三种方式不报错  import store from '../store'  router.beforeEach((to, from) => {      // 保存状态管理的数据一直存在(状态管理数据放在内存中,因此访问速度快,同时状态管理数据也是响应式的)      let user = (store.state.user)  || JSON.parse(localStorage.getItem('user'))      if (user && !store.state.user) {          store.commit('setUserInfo', user)  }  // ….  } |

3.14 登录页按下回车就发登录请求

|  |
| --- |
| function handlerKeyDown(e) {      console.log("handlerKeyDown , e:", e)      if (e.key == "Enter") {          onSubmit()      }  }  onMounted(() => {      console.log('组件已经挂载到页面了（即DOM渲染完成了），可以操作DOM或发请求')      document.addEventListener('keydown', handlerKeyDown)  })  onUnmounted(() => {      // 页面销毁时，放移除，防止内存泄漏.      // document表示本vue组件，windows表示整个窗口。只要在登录页面监听用户按回车后，执行登录操作就行      document.removeEventListener('keydown', handlerKeyDown)  }) |

生命周期顺序： setup、 onMounted

因此： setup是在onMounted之前执行，而onMounted表示vue组件加载完成(即dom渲染完成)， 所以在setup内不能操作dom。 setup就是定义ref响应式变量、函数定义等。

3.14退出登录

|  |
| --- |
| login.vue  import { showModal, showSuccessMessage } from '../common/util';  import { logout } from '../api/api';  import { useRouter } from 'vue-router'  import { useStore } from 'vuex'  const router = useRouter()  const store = useStore()  const logoutBtn = () => {      showModal("确定要退出登录吗？")          .then(res => {              //调用退出登录接口,不管成功失败,都走finally逻辑              logout().finally(() => {                  //1. 清除cookie、清除用户状态                  // store.dispatch本身是同步的，它会触发store中action，action通常是异步的，但是也可以放同步函数.                  store.dispatch("logout")                  // 3. 跳转到首页                  router.push("/login")                  // 4. 提示退出成功                  showSuccessMessage("退出登录成功")              })          })          .catch(() => {              // 点击"取消",啥也不干          })  }  消息提示框：  // 消息提示框  export function showModal(content, title = "") {      // 因为点击"确认"、"取消"会分别执行then、catch回调，所以要将对象return出去      return ElMessageBox.confirm(          content,          title,          {              confirmButtonText: '确认',              cancelButtonText: '取消',          }      )  }  vuex状态:  const store = createStore({      state:{          user: null // 有效数据的格式是对象. 初始化为null，表示没有登录，方便判断（否则要判断为一个非空对象）      },      // mutations是同步操作,使用者调用store.commit函数触发      mutations: {          setUserInfo(state, user) {              state.user = user          }      },      // actions内是放异步操作，使用者通过store.dispatch函数触发. 但是action可以放同步的操作.      // 异步操作 指的是需要等待结果返回的非阻塞操作，例如发送 HTTP 请求      actions: {          logout({commit}) {              // 清除cookie              removeCookie()              // 清除用户状态              commit("setUserInfo", {})          }      }  }) |

3.15 nprogress: 页面顶部进度条 （<https://www.npmjs.com/> ，js各种包的开源库）

|  |
| --- |
| 1. 安装： npm install nprogress  2. 主要功能：  在页面顶部显示一个细细的进度条  支持手动控制进度开始、结束、递增  支持配置样式（颜色、高度等）   1. 纯js基本使用：   import nprogress from 'nprogress';  import 'nprogress/nprogress.css'; // 引入样式  // 开始进度条  NProgress.start();  // 模拟加载，3秒后完成  setTimeout(() => {  nprogress.done();  }, 3000);   1. vue 使用：   import nprogress from 'nprogress';  import 'nprogress/nprogress.css'; // 要导入安装的样式。可放在mian.js  router.beforeEach((to, from,) => {  nprogress.start();  });  router.afterEach(() => {  nprogress.done();  }); |

### 四.后台layout全局

4.1 后台layout主布局

|  |
| --- |
| 1.使用el-container布局容器 |
| 2.修改路由：    1.App.vue是整个应用的入口组件，其中的<router-view>是最顶层的路由视图  2.当访问/login时，不会经过layoutIndex组件，而是直接在App.vue的<router-view>中渲染myLoginView  3.流程  访问/login的渲染层次：  App.vue  └── <router-view> (渲染myLoginView)    访问/的渲染层次：  App.vue  └── <router-view> (渲染layoutIndex)  └── layoutIndex的<router-view> (渲染myIndexView) |

4.2 Header组件布局

|  |
| --- |
| 1.两边居中  display: flex;  justify-content: **space-between**;  2.flex布局下的各个子元素间距  .global-header-left {  display: flex;  align-items: center;  **gap: 10px;**  //gap: 用于设置 Flex 子项之间的间距，避免手动给每个子项加 margin-right 或类似的样式，更方便、语义更清晰。  }  3.fixed固定  position: fixed; // 脱离文档流(不占据原本的位置，对其他元素没有影响，其他元素也不会为它腾地方，所以称为fixed固定。)  top:0; // 盒子模型，盒子上方的top是0  left: 0;  right: 0;  4.鼠标悬停时显示手型，并且更改背景颜色  .g-h-l-icon {  font-size: 20px;  margin-left: 20px;  height: 50px;  width: 50px;  cursor: pointer; // 鼠标悬停时显示手型  &:hover { // 鼠标悬停时，背景颜色变为深蓝色，会将width和height区域的背景颜色改掉，所以设置了height  background-color: #3a8fc0;  }  } |

4.3刷新、全屏按钮功能

|  |
| --- |
| 1.刷新  const handleRefresh = () => {  window.location.reload()  }  2.全屏切换  2.1 安装useVue： npm i @vueuse/core  2.2 导入  import { useFullscreen } from '@vueuse/core'  const { isFullscreen, toggle } = useFullscreen()  2.3 使用  <el-icon class="g-h-r-icon" @click="handleFullscreen">  <!-- 目前是非全屏状态，则显示全屏图标，否则显示已在全屏的图标 -->  <FullScreen v-if="!isFullscreen" />  <Aim v-else />  </el-icon>  const handleFullscreen = () => {  toggle()  } |

1.isFullscreen判断是否全屏。 toggle是自动切换函数，如果处于非全屏，调用toggle就会进入全屏。如果是全屏状态，调用toggle进入非全屏.

2. useVue中有很多插件使用

|  |
| --- |
| 2.4 使用计算属性  <!-- store也是响应式变量,如果退出登录时，将store.state.user设置为null，那这里机会报错. 所以用 计算属性 优雅判断下 -->  <!-- {{ $store.state.user.avatar }} -->  {{ username }}  <script setup>  const username = **computed**(() => {  if (store.state.user) {  return store.state.user.username  }  return ""  })  </script> |

* 1. 修改密码

|  |
| --- |
| 1. 使用Drawer抽屉，展示修改密码的页面   <el-drawer v-model="showDrawer" title="修改密码哦" **:**close-on-click-modal="false">  // 这里放表单  </el-drawer>   1. 展示效果： |

知识点：

1. Drawer 拥有和 Dialog 几乎相同的 API, 在 UI 上不一样。 Drawer是 呼出一个临时的侧边栏.
2. close-on-click-modal是一个布尔属性，为false时表示 阻止点击遮罩层时关闭抽屉
3. : 是 v-bind 指令的缩写，用于将数据绑定到 HTML 属性上。是否需要加 : 取决于你传递的值是静态字符串还是动态表达式。举例：

3.1不需要加 : 的情况：当传递的值是一个固定的字符串时，可以直接写，如 title="修改密码哦"。

3.2需要加 : 的情况：当传递的值是一个 JavaScript 表达式（各种变量、计算结果、函数返回值等）时，必须使用 :，如 :close-on-click-modal="false"

4.5弹框表单组件化(即drawer侧边栏)

|  |
| --- |
| 0.组件代码组织和使用    1.布局 ： 提交按钮在底部，内容占据其它空间，如果内容长度超过一定值，则显示滑动条。  <div class="main">  <div class="content">  <slot></slot>  </div>  <div class="footer">  <el-button type="primary" @click="submit">{{ submitText }}</el-button>  </div>  </div>  // 父元素使用flex布局  .main {  height: 100%;  width: 100%;  // background-color: red;  display: flex;  flex-direction: column;  }  // 子元素1  .content {  // **flex: 1,即自动占据父容器剩余的所有空间。 前提是父容器一定要是flex布局**  // 注： 容器剩余的，也就是父容器下的其他所有子元素空间去掉后，这里就是去掉footer占据的空间  **flex: 1;**  // 元素在y方向超过容器长度，则显示滚动条  **overflow-y: auto;**  }  // 子元素2  .footer {  height: 50px;  }  2.defineProps、 defineExpose、defineEmits  defineProps：暴露属性，参数是对象（则父组件可以设置属性值了）  defineExpose：暴露方法，参数是对象（则父组件可以调用组件的方法了）  defineEmits: 定义事件，参数是list (则父组件可以监听事件并执行对应回调函数)  3.组件使用(设置prop属性，调用expose的方法、监听emit事件执行回调)  <form-drawer ref="formDrawerRef" size="45%" @submitEmit="onUpatePwdSubmit" submitText="确认修改">  <el-form :model="form" ref="formRef" :rules="rules" label-width="auto"  style="max-width: 600px; margin: 1em auto;">  <el-form-item label="旧密码1" prop="oldPwd">  <el-input v-model="form.oldPwd"></el-input>  </el-form-item>  </el-form>  </form-drawer>  const formDrawerRef = ref(null)  const handleModifyPassword = () => {  formDrawerRef.value.open()  }  const onUpatePwdSubmit = () => { //….} |

这块整体的代码，要记住。很有代表性。

4.5侧边栏菜单

|  |
| --- |
| 1.只有一级菜单(只使用el-sub-menu)  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">          <template #title>              <span>only一级啦</span>          </template>      </el-sub-menu>  </el-menu>    2.二级菜单  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">          <template #title>              <span>一级啦</span>          </template>        <el-menu-item index="1-4-1">二级啦</el-menu-item>      </el-sub-menu>  </el-menu>    3.三级菜单  <el-menu class="my-menu" @open="handleOpen" @close="handleClose">      <el-sub-menu index="1">         <template #title><span>一级</span></template>         <el-sub-menu index="1-4">              <template #title>二级</template>              <el-menu-item index="1-4-1">三级</el-menu-item>          </el-sub-menu>      </el-sub-menu>  </el-menu> |

总结：

1. 用el-sub-menu表示子菜单，其尾部有箭头标志； 用el-menu-item表示是最底层的菜单，其尾部没有箭头标志。
2. 如果需要在菜单出增加图标，在<template #title>中添加

|  |
| --- |
| <el-menu-item index="1-4-1">  <**template** #title>  **<el-icon> <location /> </el-icon>**  <span>items One</span>  </template>  </el-menu-item> |

* 1. 使用v-for、v-if构建菜单

|  |
| --- |
| 1. 使用template标签，它是一个 不会被渲染成 DOM 的占位标签，方便操作。 |
| v-if判断是否是带子项的菜单，如果是的话，则用v-for渲染子项菜单(这里默认只有二级菜单), 如果不是的话，那就走到v-else渲染不带子项的菜单 |

4.6动态渲染图标

|  |
| --- |
| <component :is="xxx">， 它是Vue 的动态组件语法  xxx 可以是组件名字符串（如 'ElHomeFilled'）  xxx或者可以是组件对象（如 import { HomeFilled } from '@element-plus/icons-vue'）  所以，<component :is="HomeFilled" />，等价于 <HomeFilled />  1.正确写法：  <el-icon>  <component :is="item.icon"></component>  </el-icon>   1. 错误写法   <el-icon>  {{ item.icon }}  </el-icon>  item.icon 是一个变量，不是一个模板标签，Vue 不会自动解析它为组件 |

4.7点击菜单进行路由跳转

|  |
| --- |
| <el-menu>开启router模式：启用该模式会在激活导航时以 index 作为 path 进行路由跳转, 使用 default-active 来设置加载时的激活项. |

4.8展开/折叠菜单栏(通过设置 菜单栏宽度实现)

|  |
| --- |
| 1.摆脱父元素宽度影响.  <!-- 设置宽度为auto，则宽度会根据global-aside子内容自适应. el-aside原先默认的宽度200px就是无效了 -->  <el-aside **width="auto"**><global-aside></global-aside></el-aside>  2.展开、折叠click事件，更改菜单栏宽度  <el-icon class="g-h-l-icon g-h-l-fold" @click="modifyMenuWidth">  const modifyMenuWidth = () => {  **store.commit**("changeAsideWidth")  }  3.1根据宽度值，设置 是否水平折叠收起菜单  <el-menu :collapse="collapse" >  const collapse = computed(() => {  **return** !(store.state.asideCurWidth == "300px")  })  3.2.动态添加样式（修改菜单栏宽度）  <div :style="appendStyle">  const appendStyle = computed(() => {  **return** {  **width: store.state.asideCurWidth**,  transition: 'all 0.2s',  // 隐藏水平滚动条（当元素内容超过父元素宽度，水平方向会出现滚动条）  // 在动态计算css时，overflow-x则需要用驼峰命名法  overflowX: 'hidden'  }  }) |

注： 摆脱父元素样式控制，还有以下方法：

1. 将样式直接写在父元素上
2. 子元素设置为fixed定位，脱离了文档流，那么也肯定脱离了父元素的影响

4.9 刷新页面时，页面当前路由和菜单关联，自动激活指定的菜单

|  |
| --- |
| <el-menu :default-active="defaultActive" >  import {useRoute} from 'vue-router'  const routes = useRoute()  const defaultActive = computed(() =>{  return routes.path  }) |

router和route的区别：

|  |  |  |
| --- | --- | --- |
| useRouter() |  | 获取**路由实例**对象，用来**编程式导航（跳转）、获取路由表** |
| useRoute() |  | 获取**当前路由信息**对象，用来**读取当前路径、参数、query 等** |

4.10动态路由

1.动态路由是指：**根据程序运行时的逻辑，动态添加、删除或修改路由规则**，而不是一开始就把所有路由都写死在 routes.js 里。

2.目的： 达到权限控制的目的，登录后动态加载对应路由（菜单 = 路由），不同用户加载不同路由。用户 A 登录 → 只能访问后台管理、商品页，用户 B 登录 → 可以访问订单、统计、系统设置。

|  |
| --- |
| //**第一步**：定义静态路由表  const routes = [      {          name: "login",          path: "/login", // 因为是哈希模式。所以用host:port/#/login访问          component: myLoginView,          meta: {              title: "登录页"          }      } ,      {          name: "layoutIndex",  // 动态的路由全部放在它的children内          path: "/",          component: layoutIndex,          children: []      },      {          path: '/:pathMatch(.\*)\*',          name: 'NotFound',          component: NotFound      },  ]  //**第二步**：根据菜单数据生成路由配置  function generateRoutesFromMenus(menus) {      const routes = []        // 递归处理菜单      function traverseMenus(menuList) {          if (!menuList || !menuList.length) return []            const result = []            menuList.forEach(menu => {              // 只处理状态为启用的菜单              if (menu.status !== 1) return                // 如果有前端路径，则创建路由              if (menu.frontpath) {                  const route = {                      path: menu.frontpath,                      name: `menu-${menu.id}`,                      meta: {                          title: menu.name,                          icon: menu.icon                      }                  }                    // 如果有前端文件路径，则动态导入组件                  if (menu.front\_file\_path) {                      // 使用动态导入（Vite的别名解析主要发生在构建时，所以用~是不行的，后端配置时里要使用绝对路径或者相对路径）                      route.component = () => import(/\* @vite-ignore \*/ menu.front\_file\_path)                  } else {                      // 默认使用空白页面                      route.component = () => import('~/page/index.vue')                  }                  console.log("push动态路由:", route)                  result.push(route)              }                // 处理子菜单              if (menu.child && menu.child.length) {                  const childRoutes = traverseMenus(menu.child)                  result.push(...childRoutes)              }          })            return result      }        return traverseMenus(menus)  }  第三步： 调用  // Vue Router的路由配置存储在内存中，页面刷新时，动态添加的路由会丢失,那么页面会出现404。解决的办法就是加个hasLoadedRoutes标志：页面刷新时，增加动态路由，设置已加载动态路由，再使用replace：true更新路由，最后出发重新导航  let hasLoadedRoutes = false;  router.beforeEach(async (to, from) => {      // 3. 如果有token且路由为空，加载动态路由      if (token && router.getRoutes().find(route => route.name === 'layoutIndex').children.length === 0 && hasLoadedRoutes === false) {          await loadDynamicRoutes()          // 标记动态路由已加载(一定要标记动态路由已加载，否则会无限重定向)          /\*当用户刷新页面时，这段代码会：              1.加载动态路由              2.返回相同的路径，触发重新导航              3.重新导航时，路由守卫再次执行              4.由于我们没有标记动态路由已加载，条件再次满足              5.再次返回相同路径，形成无限循环 \*/          hasLoadedRoutes = true          // 如果是刷新页面或直接访问URL，重新导航到当前页面          return { path: to.fullPath, replace: true } // 返回相同的路径，触发重新导航,这会让Vue Router用新的路由配置重新匹配路由（即生效新路由）      }  }) |

总结： 1. 刷新页面会导致动态路由失效，需要重新注册，结合hasLoadedRoutes标志和“{ path: to.fullPath, replace: true }“ 【难理解】

需要后端支持: 菜单接口。菜单表如下：



比较设计好的地方：分段表示： id为1-10是一级菜单， 10-19是二级菜单，20-29又是另外一段二级菜单. 【这里的rule\_id就是parent\_id】

最后：上面通过将菜单数据放在后端配置，前端路由和菜单栏都是用的同一份数据，并且后端维护数据比较灵活。但是同时代码也比较麻烦。 **简化的做法**：前端配置死所有的路由表,每个表中的meta配置权限信息，在生成菜单的时候，通过判断用户权限决定是否显示菜单。（潜在的风险：虽然未在菜单中显示，但用户仍可通过手动输入地址访问这些路由，若无额外校验，可能导致越权访问。）

|  |
| --- |
| // 所有路由的定义  const routes = [  {  name: "layoutIndex",  path: "/",  component: layoutIndex,  children: [{  name: "index",  path: "/", // 因为是哈希模式。所以用host:port/#/login访问  component: myIndexView,  meta: {  title: "首页"  **permission: [‘admin’, ‘normal’]**  }  }]  }]  // 菜单栏展示  <el-sub-menu **v-if="userRole in item.permission"** :index="item.frontpath">  <template #title>  <el-icon>  <component :is="item.icon"></component>  </el-icon>  <span>{{ item.name }}</span>  </template>  </el-sub-menu> |

4.11 tabs选项卡

|  |
| --- |
| <div class="tabList" :style="{ left: $store.state.asideCurWidth }">  <el-tabs v-model="avtiveTabsValue" type="card" @tab-change="handleTabChange" @tab-remove="removeTab">  <!--可以每个tab选项卡分别设置是否可以关闭。 后台首页不可关闭，其他都可以 -->  <el-tab-pane v-for="item in tabList" :closable="item.path != '/'" :key="item.path" :label="item.title" :name="item.path">  </el-tab-pane>  </el-tabs>  </div>  .tabList {  display: flex;  **position: fixed;**  top: 60px;  /\* right:0，表示宽度占满到最右边\*/  **right: 0;**  /\* 设置z-index和 background-color是为了在滑动主区域内容时， tabList 不会被遮挡 \*/  /\* 添加 z-index 确保在其他元素上方 \*/  z-index: 100;  /\* 添加背景白色 \*/  background-color: #fff;  } |

4.12 tabs选项卡和菜单、路由 联动绑定

|  |
| --- |
| import { **onBeforeRouteUpdate**} from '**vue-router'**  **// 路由跳转之前, 会执行onBeforeRouteUpdate函数**  **onBeforeRouteUpdate**((to, from) => {  //1. 不存在则追加  const notFind = tabList.value.findIndex(obj => obj.path == to.path)  if (notFind == -1) {  const newTab = {  title: to.meta.title,  path: to.path  }  tabList.value.push(newTab)  **setCookie(COOKIE\_KEY, JSON.stringify(tabList.value))**  }  // 2.设置tab激活选中的项  avtiveTabsValue.value = to.path  }) |

4.13 tabs选项卡，写入cookie，从cookie中加载

目的： 记录用户上次所有选中的选项卡

|  |
| --- |
| // 联动1: 刷新时/刚进入页面时， 浏览器的path路径 和 tab选项卡 绑定； 从cookie 中加载保存过的tablist  const avtiveTabsValue = ref()  const init = () => {  let savedTabListVal = getCookie(cookieKey)  console.log("savedTabList:", savedTabListVal)  if (savedTabListVal) {  tabList.value = JSON.parse(savedTabListVal)  }  const route = useRoute()  avtiveTabsValue.value = route.path  }  init() |

4.14 关闭选项卡

|  |
| --- |
| // 关闭tab选项卡  // 功能： 关闭的是最后一个，选中上一个；否则，选中下一个  const removeTab = function (tabPaneName) {  **// 0. 后台首页 不可关闭**  if (tabPaneName == "/") {  return  }  **// 1. 更新 新的激活的选项卡**  let index = tabList.value.findIndex(obj => obj.path == tabPaneName)  let nextActiveTab = ""  if (index == tabList.value.length - 1) {  nextActiveTab = tabList.value[tabList.value.length - 2].path  } else {  nextActiveTab = tabList.value[tabList.value.length - 1].path  }  avtiveTabsValue.value = nextActiveTab  **// 2. 删除关闭的选项卡**  tabList.value = tabList.value.filter(obj => obj.path != tabPaneName)  **// 3. 更新cookie**  setCookie(COOKIE\_KEY, JSON.stringify(tabList.value))  **// 4. 显示新的激活的选项卡**  router.push(avtiveTabsValue.value)  } |

4.15 keep-alive

作用：缓存被包裹的组件实例，在组件切换时不会销毁，而是保持它的状态（如输入内容、滚动位置、生命周期等）

举例： 现在在商品列表的**第二页**，此时点击其他模块后再切回到商品列表模块时，还是在**第二页**（这个过程中不会发送任何http请求）

|  |
| --- |
| <router-view></router-view>  上面是基础的写法，等价于下面的：  <router-view v-slot="{ Component }">  <component :is="Component" />  </router-view>  当前要渲染的组件是 Component  使用keep-alive  **<keep-alive>**  <router-view v-slot="{ Component }">  <component :is="Component" />  </router-view>  **</keep-alive>** |

### 五.后台主控台开发

5.0 代理配置

|  |
| --- |
| 一、axios使用  const instance = axios.create({  **baseURL**: '/whatever',  timeout: 3000  })  axiosInstance.post("**/user/profile**")  最终的请求地址是：会将 **baseURL和path**进行拼接，即'/whatever**/user/profile‘，**  **如果 baseURL 没有设置完整的地址（**比如没有写 http://xxx.xxx），那么 **axios 会默认使用当前页面所在的地址作为主机名**（例如：<http://127.0.0.1:5173），从而拼接成完整请求地址>, 即：http://127.0.0.1:5173/whatever/user/profile  二、vite.config.js 配置代理  export default defineConfig({  server: {  **proxy**: {  '/**whatever'**: {  target: 'http://ceshi13.dishait.cn', // 你后端服务的地址  changeOrigin: true, // 是否改变请求源头（伪装成目标地址）  rewrite: (path) => path.replace(/^\/whatever/, ''), // 重写路径去掉 /whatever  },  },  },  })  '/**whatever'**是代理的请求前缀路径，用于匹配前端发出的请求，决定哪些请求要被代理。  代理中会做两件事：  1.把请求转发到目标服务器<http://ceshi13.dishait.cn>  2. rewrite 会把 /whatever**/user/profile‘**→ /user/info [即：只将 /whatever去掉]  三、代理流程  浏览器控制台显示的是前端发出的原始请求，而不是代理后的请求  前端代码运行在浏览器中，它向本地开发服务器 (127.0.0.1:5173) 发送请求  然后由 Vite 开发服务器将请求代理到 ceshi13.dishait.cn 。总结如下：  1.浏览器 → 发送请求到 → 本地开发服务器 (127.0.0.1:5173)  2.本地开发服务器 → 代理请求到 → 远程服务器 (ceshi13.dishait.cn)  这样就避免浏览器直接请求ceshi13域名，导致报跨域问题了。 【参考印象笔记中“面试相关”-“前端“】 |

5.1 后台el-container布局

目的： 替换fixed布局，代码多，还复杂

|  |
| --- |
| 1. header-aside-main布局， header固定，main可滑动   <template>      <div class="common-layout">          <el-container class="containder-index">              <el-header class="header">Header</el-header>              <el-container>                  <el-aside width="200px" class="aside">Aside</el-aside>                  <el-main class="main">                      <p v-for="i in 100">test.{{ i }}</p>                  </el-main>              </el-container>          </el-container>      </div>  </template>  <style lang="scss" scoped>  .common-layout {      background-color: red;      width: 100%;      height: 100%;  }  .containder-index {      background-color: yellow;      height: 100%;      width: 100%;  }  .header {      background-color: black;      width: 100%;      height: 60px;  }  .aside {      background-color: pink;      height: 100%;  }  .main {      overflow-y: auto;      // 减号，前后要有空格      height: calc(100vh - 60px);  }  方案：使用固定高度 + 内部滚动  即：不让整个页面滚动；,而是让 <el-main> 独立滚动；  <el-header> 保持固定即可。  实现效果：  1.页面头部 <el-header> 固定；  2.内容多的时候，只有 <el-main> 会滚动；  3.滚动时不会遮挡或影响头部  关键实现：   1.common-layout、containder-index高度要有继承或者自己设置，不要依赖子元素高度。   2.el-main 的高度计算方式为：height: calc(100vh - header高度 - footer高度);   3.el-main的内容较多时，实现垂直滚动：overflow-y: auto;  这个其实跟之前的el-drawer实现是类似的，最外面是flex布局，上面子元素区域可以滑动（flex:1, overflow-y:auto），下面子元素是固定高度的“提交”按钮。 |

如果要在er-main 再加一块固定的 tab 选项卡的布局，那直接在el-main上加，然后计算el-main的高度时，也继续减掉tab选项卡的高度

|  |
| --- |
| <template>      <div class="common-layout">          <el-container class="containder-index">              <el-header class="header">Header</el-header>              <el-container>                  <el-aside width="200px" class="aside">Aside</el-aside>                  <el-container>                      <!-- 这里不一定是el-header,用个div都行 -->                      <el-header class="tab-header">                          <span>tab1</span>                          <span>tab2</span>                      </el-header>                      <el-main class="main">                          <p v-for="i in 100">hahaha.{{ i }}</p>                      </el-main>                  </el-container>              </el-container>          </el-container>      </div>  </template>  // 其他的css不变  .tab-header {      height: 50px;      background-color: orange;  }  .main {      overflow-y: auto;      // 减去header1 ，再减去tabHeader      height: calc(100vh - 60px - **50px**);  } |

5.2后台首页-统计部分展示

使用el-row和el-col布局，el-row占用24栏，给每个el-col分配6栏的宽度，每个el-rol下放 el-card元素

|  |
| --- |
|  |



|  |
| --- |
| el-card分三部分，每部分都是**可选**   1. template #header ，【头部，即第一行内容 ( slot方式)】 2. 中间局域 (这里随意填html标签) 3. template #footer 【尾部，即第三行内容 ( slot方式)】 |

5.3 el-check-tag使用



|  |
| --- |
| <div class="flex gap-2">  <el-check-tag :checked="currentTab == item.value" v-for="(item, index) in echarTabs" @**click**="**handleClickTab**(**item.value**)">  {{ item.label }}  </el-check-tag>  </div>  // 选项卡  const currentTab = ref("week")  const echarTabs = [  {  label: "近1个月",  value: "month"  }, {  label: "近1周",  value: "week"  }, {  label: "近24小时",  value: "day"  },  ]  // 选项卡切换  const handleClickTab = (selectedValue) => {  currentTab.value = selectedValue  } |

click事件，可以传自定义的值。

echarTabs定义里，label是用于展示的，value是内部实际值。currentTab用于和value匹配

5.4 echart使用

|  |
| --- |
| 1.安装 npm install echarts  2.导入：import \* as echarts from 'echarts'; //导入所有表，只用到柱状图，可按需导入  3.定义显示表的dom容器：  <div id="main" style="width: 600px;height:400px;"></div>   1. js准备数据、渲染   // 基于准备好的dom，初始化echarts实例  var myChart = echarts.init(document.getElementById('main'));  // 指定图表的配置项和数据  var option = {  title: {  text: 'ECharts 入门示例'  },  tooltip: {},  legend: {  data: ['销量']  },  xAxis: {  data: ['衬衫', '羊毛衫', '雪纺衫', '裤子', '高跟鞋', '袜子']  },  yAxis: {},  series: [  {  name: '销量',  type: 'bar',  data: [5, 20, 36, 10, 10, 20]  }  ]  };  // 使用刚指定的配置项和数据显示图表。  myChart.setOption(option); |

参考：<https://echarts.apache.org/handbook/zh/get-started/>

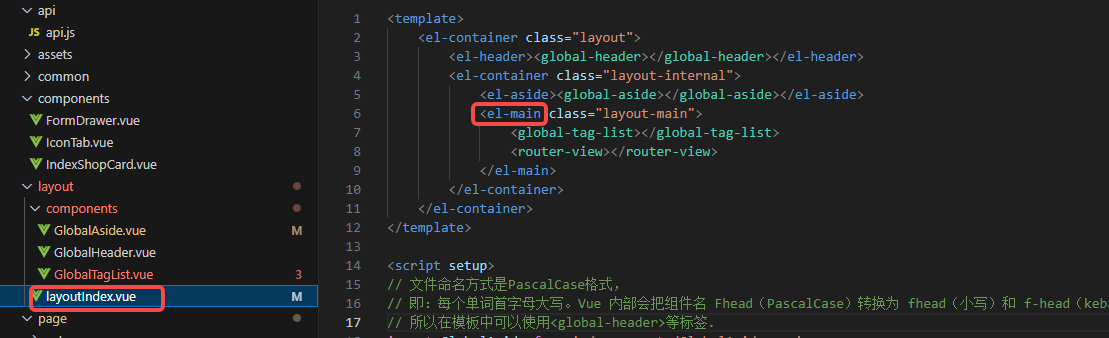
https://echarts.apache.org/examples/zh/index.html

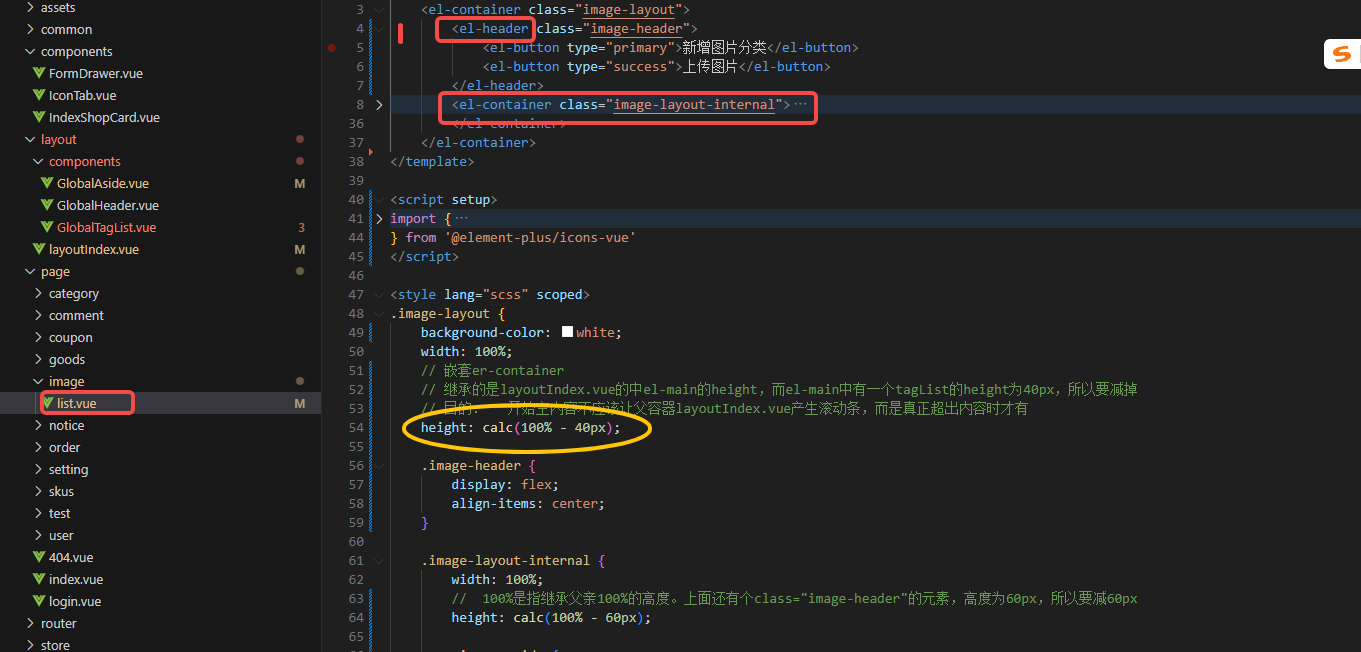
5.5 echart常用操作: 销毁实例、 加载loading、ECharts 图表大小自适应

|  |
| --- |
| // 1.页面即将销毁时，销毁echart实例。 防止页面卡顿等异常情况  onBeforeUnmount(() => {  if (myChart) {  echarts.dispose(myChart)  }  })  // 2.加载loading  myChart.showLoading()  myChart.hideLoading()  // 3. 当图表外面的盒子大小变了，自动调整图表大小，保证图表能正常显示。  import { useResizeObserver } from "@vueuse/core";  const el = ref(null)  useResizeObserver(el, (entries) => {  //const entry = entries[0]  //const { width, height } = entry.contentRect  //console.log("width: ", width, ", height:", height)  myChart.resize()  }) |

### 六.图库模块开发

6.1 继承父元素高度时





list.vue是嵌套在 layoutIndex.vue的下，更精确的是在它<route-view>下，会将<router-view>覆盖掉，而不是继承<router-view>【通过审查元素发现的】,所以list.vue的高度最高是 calc(100% - globalTagList高度) [ globalTagList高度是40px] 。 同理，.image-layout-internal的同级有一个高度为60px的el-header, 所以它的高度最高是calc(100% - 60px)

6.2 动态绑定active class、设置hover样式

|  |
| --- |
| **//1. 动态绑定 class 【当值为true，就会增加这个class】**  <div class="top-list" **:class="{ 'active': active }"**>  // ...  </div>  defineProps({  active: {  type: **Boolean**,  default: false  }  })  /\* **有空格：** .active是.top-list的**子元素** \*/  .top-list .active {  background-color: #EFF6FF;  cursor: pointer;  }  /\* **无空格：** 同时**具有两个类的元素【记忆： 黏在一起，同时具有】** \*/  .top-list.active {  background-color: #EFF6FF;  cursor: pointer;  }  // 2. scss中设置hover样式  .top-list {  border-bottom: 1px solid #e0e0e0; // 线条浅灰色  **// & 不能少 【&表示引用 父标签 】**  &:hover {  background-color: #EFF6FF;  cursor: pointer;  }  }  下面是普通css的写法 （**无空格**）  .top-list:hover {  background-color: #EFF6FF;  cursor: pointer;  } |

6.3分页页码

|  |
| --- |
| <!-- 必须使用v-model双向绑定； **：**只是单向传递，从父组件到子组件传递 -->  <el-pagination background layout="prev, pager, next" :total="totalCount" **v-model**:page-size="pageSize" **v-model**:current-page="currentPage" @current-change="handleChangeCurrentChange"/>  //参考ElementPlus官方文档   1. 必须要设置 total（总记录条数）、page-szie、current-page才能显示页码 2. 当前页码发生变动，会出发current-change事件 |

6.4 新增和编辑 共用一个FormDrawer( 或共用 表单等)

|  |
| --- |
| 1.使用editId来区分是新增或编辑，当新增记录时，id是不存在的，即0，当编辑时，数据记录存在，因此可以拿到editId。  const handleSubmit = () => {  // 根据editID决定，使用 新增函数 还是编辑函数，返回的都是一个promise对象  const fun = **editId.value > 0** ? updateImageCategory(editId.value, form) : addImageCategory(form)  fun.then((res) => {  **let showTitle = editId.value > 0 ? "编辑成功" : "新增成功"**  showSuccessMessage(showTitle)  // 编辑的话刷新当前页面，新增的话就加载首页  let newPage = editId.value > 0 ? currentPage : 1  getData(newPage)  // 关闭  formDrawerRef.value.close()  })  .finally(() => {  formDrawerRef.value.hideLoading()  })  } |

6.5 气泡框删除

|  |
| --- |
| <!-- @confirm="handleDelete, 点击 **确认** 文案，触发该回调 -->  <el-popconfirm title="是否确认删除" confirm-button-text="确认" cancel-button-text="取消" @**confirm**=" handleDelete ">  <!—显示的内容-->  **<template #reference>**  **<el-button text size="small">**  **<el-icon size="16">**  **<Close />**  **</el-icon>**  **</el-button>**  **</template>**  </el-popconfirm> |

6.6 绝对定位

|  |
| --- |
| <el-card shadow="hover">  <el-image :src="item.url" fit="fill" :lazy="true"></el-image>  <div class="name">{{ item.name }}</div>  </el-card>  .el-card {  position: relative;  }  .name {  // 使用绝对定位，是相对上一个 relative定位父元素  position: absolute;  left: 0;  right: 0;  bottom: 25px;  //等价于: background-color: #1f2937; opacity: 0.5;  background-color: rgba(31, 41, 55, 0.5);  }  实现的效果如下：    即： 将name( public/68xxx.gif) 显示在相对父元素的某个位置，这里绝对定位到父元素bottom：25px处。 正好可以“覆盖”图片的底部。 |

6.7重命名、删除、图片上传功能

### 七.公告模块

这里的功能和之前的类似，也是 增删改查功能，没有比较新的知识点。

7.1 el-table的列的2种使用

|  |
| --- |
| 1.直接展示列的内容  <el-table :data="**tableData**" stripe style="width: 100%">  <**el-table-column** **prop**="date" label="Date" width="180" **/>**  <el-table-column prop="name" label="Name" width="180" />  <el-table-column prop="address" label="Name" width="180" />  <el-table>  const tableData = [{  date: '2016-05-03',  name: 'Tom',  address: '  No. 189, Grove St, Los Angeles',  }]    总结： **el-table-column中的prop 和 tableData中对象的key 一一对应**  **2. 自定义**列的显示内容  <el-table-column prop="title" label="标题" width="180" />  <el-table-column label="操作">  **<template #default="scope">**  <el-button size="small" @click="handleEdit(**scope.$index, scope.row**)">  编辑  </el-button>  <el-button size="small" type="danger" @click="handleDelete(scope.$index, scope.row)">  删除  </el-button>  </template>  **</el-table-column>**    **总结： 使用 template #default=“scope”的插槽。 scope.$index是当前行的索引下标，scope.row是当前行的数据，对象格式.** |

### 八.管理员模块开发

这里的功能和之前的类似，也是 增删改查功能。

8.1表格自定义列展示

|  |
| --- |
| <el-table-column label="管理员" width="180">  <!-- 直接解构row，省的写scope.row.xxx -->  <template #default="{row}">  <div class="**admin**">  <div class="**avatar**">  <el-avatar :size="small" :src="row.avatar" />  </div>  <div class="**adminInfo**">  <span>{{ row.username }}</span>  <span>ID: {{ row.id }}</span>  </div>  </div>  </template>  </el-table-column>    注： css中，admin是flex布局，垂直居中就行。 adminInfo是按列排的flex布局 |

8.2 switch组件更改状态

|  |
| --- |
| <el-table-column label="状态" width="360">  <template #default="scope">  <el-switch **v-model="scope.row.status" :active-value="1"** **:inactive-value="0"**  :loading="scope.row.switchLoading"  **@change="(val) => handleStatusChange(val, scope.row)"** />  </template>  </el-table-column>  // 请求接口, 更改管理员状态  const handleStatusChange = (val, row) => {  row.switchLoading = true  changeManagerStatus(row.id, val).then(res => {  showSuccessMessage("修改状态成功")  }).finally(() => {  row.switchLoading = false  })  }    注1： el-switch有3个属性，第一个v-model，绑定的值只有0或1， 所以设置:active-value=”1”。 因为scope.row.status（后端返回的）是整形0或1，所以一定要使用 : 动态绑定，否则传的就是字符串1。  注2: 状态发生改变，会触发change事情，官方定义会有对应的回调函数，且有一个val参数【**参数名不可以改变，否则无法结构得到正确的值**】, 但是业务还需要传入 当前行的数据，所以用了一个闭包： **@change="(val) => handleStatusChange(val, scope.row)"** |

8.3自定义的选择图片组件

|  |
| --- |
| <**el-dialog** **v-model="dialogVisible"** title="选择图片" width="60vw" **top="5vh**">  // 放业务内容  **<template #footer>**  <div class="dialog-footer">  <el-button @click="dialogVisible = false">取消</el-button>  <el-button type="primary" @click="confirmChooseImage">  确认  </el-button>  </div>  </template>  </el-dialog>    注1: 用的是el-dialog弹出框。 width是弹出框宽度，top是距离浏览器顶部的距离，**dialogVisible是控制是否显示。 <template #footer>是弹出框下面放取消、确认按钮的。**  注2： 当**dialogVisible=false，**即不显示时，el-dialog内 元素 是没有被渲染成dom的，也就是通过ref是拿不到元素 |

8.4在自定义的组件上，自定义v-model实现

|  |
| --- |
| 子组件部分： 自定义的选择图片组件ChooseImage.vue:  <!-- 显示选择的照片【可选，可不看】 -->  <el-image v-if="modelValue" :src="modelValue" fit="fill" style="width: 100px; height: 100px; "></el-image>  // 1.定义modelValue  const props = defineProps({  modelValue: {  type: String  }  })  // 2. 定义emit事件  const emit = defineEmits(["update:modelValue"])  // 3. modelValue发生变化，调用emit通知父组件  const confirmChooseImage = () => {  dialogVisible.value = false  console.log("confirmChooseImage:", modelValue.value)  **emit("update:modelValue", modelValue.value)**  }  父组件部分: 父组件使用ChooseImage.vue:  <ChooseImage v-model="avatar" />  总结子组件步骤： 1.定义modelValue 属性；  2.定义update:modelValue的emit  3.当组件内modelValue改变时，调用emit通知给父组件 |

### 九.菜单权限模块开发

9.0 tree树形控件

|  |
| --- |
| <**el-tree** :**data**="dataSource" :**props**="{ label: 'name', children: 'child' }" **node-key**="id"  :default-expanded-keys="defaultExpandedKeys">  <template #default="{ node, data }">  <div>  <div class="left">  <span>{{ data.name }}</span>  </div>  <div @click.stop class="right">  <el-switch v-model="data.status" :active-value="1" :inactive-value="0" @change="(val) => handleStatusChange(val, data)" />  </el-button>  </div>  </div>  </template>  </el-tree>   1. data是总数据源，如下：   [{  **"id": 5,**  "**name**": "后台面板",  "desc": "index",  "**child**": [  {  "id": 10,  "name": "主控台",  "desc": "index",  "child": []  }  ]  }]   1. **node-key，** 用于指定每个树节点的 **唯一标识字段名**，即数据源中每个节点对象的唯一键。 **node-key**="id"，即”id”这个key对应的值是唯一的。 2. prop，在数据源这个数据结构中，哪些字段代表“标签名”（标签名就是用来展示的）、哪些字段代表“子节点”等信息      1. default-expanded-keys, 用于设置默认展开的节点, 它的值是：node-key 值组成的数组。 比如： [5], 那么只会展开id=5的节点 2. <template #default="{ node, data }"> 这个插槽是用来自定义显示，不再单纯的只展示label， data是数据源中的对象（一般就用它），node是节点      1. check-strictly默认值是false， 不断绝父子关系。 true的话是断绝父子关系，即每个节点都是独立的。   6.1默认行为（checkStrictly = false）：  当你勾选一个父节点时，所有子节点也会自动被勾选；  当你取消一个父节点时，所有子节点也会被取消。  🔁 同理，子节点勾选或取消，也会自动影响到父节点。  6.2设置 checkStrictly = true 后：  选中父节点，不会自动选中子节点。  选中子节点，也不会影响父节点的选中状态。  换句话说，每个节点的选中状态都是独立的，可以选中任意一级节点。   1. 开启**show-checkbox属性时，即展示复选框，则复选框被选中或取消的时候会触发check方法** |

9.1 阻止冒泡点击

|  |
| --- |
| 方法一：在父div上设置  <div **@click.stop** >  <el-button type="primary" link @click="append(data)" text>  </div>  方法二： 在本节点上设置  <div>  <el-button type="primary" link **@click.stop**="append(data)" text>  </div> |

9.2 单选radio组件

|  |
| --- |
| <el-form-item label="菜单/规则">  <el-radio-group **v-model="form.menu"**>  <el-radio **:value="1"** border>菜单</el-radio>  <el-radio **:value="0"** border>权限</el-radio>  </el-radio-group>  </el-form-item> |

9.3级联选择器

|  |
| --- |
| <**el-cascader** v-model="form.rule\_id" :options="dataSource"  :props="{ 'label': 'name', 'children': 'child', 'checkStrictly': true, 'value': 'id' }"  clearable />   1. options属性是数据源 2. props属性和 el-tree的一样，设置数据源中的哪个字段来展示 |

### 十.角色模块开发

|  |
| --- |
| el-tree-v2 用法和el-tree一样，它的性能更强，挂载大量数据节点，不会卡  <el-tree-v2 ref="elTreeRef" :data="allPermissionList"  :props="{ 'label': 'name', 'children': 'child' }"  **show-checkbox** :height="700" node-key="id">  node-key="id" **@check="handleChecked"**> |

|  |
| --- |
| 1. tree组件中， 设置默认选中某些节点  const handleEdit = (index, row) => {  formDrawerRef.value.open()  // 获取当前角色下的所有权限id  **ruleIds.value** = row.rules.map(o => o.id)  // 自动勾选上已有的权限  // formDrawerRef.value.open()执行后，其下的elTree还没有立即被渲染出来,即elTreeRef是null,所以设置了一个timeout延迟执行  setTimeout(() => {  **elTreeRef.**value.**setCheckedKeys**(ruleIds.value)  }, 150);  }  关键方法： **setCheckedKeys**  **2、 当复选框被点击的时候触发check事件 （只要点击就行，选中和取消都会触发）**  **/\***  **2.1 data是被点击的节点（选中状态或未选中状态）**  **2.2 info是额外的信息, checkedKeys 是所有子节点id, halfCheckedKeys 是所有半选父节点的id。 即当某个节点下所有的子节点都被选时，halfCheckedKeys是空； checkedKeys不是所有子节点时, halfCheckedKeys才会有值**  **\*/**  const selectedKeys = ref([])  const **handleChecked** = (data, info) => {  let { checkedKeys, halfCheckedKeys } = info  // 拼接 父节点和子节点的id  // selectedKeys.value = [...checkedKeys, ...halfCheckedKeys]  // 由于check-strictly默认是false，即不断绝父子关系，选父必选子，有子必有父。 所以只要提交checkedKeys即可  **selectedKeys.value = [...checkedKeys]**  **// 将selectedKeys更新到服务器数据库中。**  } |

### 十一.商品规格模块开发

10.1 自定义v-model时，子组件内在定义modelValue属性后，后续使用”props.modelValue”来获取父组件传过来的值。template或者js script都不要加.value

|  |
| --- |
| 例1： <el-tag v-for="tag in **dynamicTags**" :key="tag" closable :disable-transitions="false" @close="handleClose(tag)">  {{ tag }}  </el-tag>  const **props** = defineProps({  modelValue: {  type: String  }  })  const **dynamicTags** = **computed**(() => {  if (**props.modelValue** == "") {  return []  }  return props.modelValue.split(",")  })  例2：  <el-image **v-if="props.modelValue" :src="props.modelValue"** fit="fill"></el-image>  // 子组件定义props属性  const **props** = defineProps({  modelValue: {  type: String  }  })  // 定义emit事件  const emit = defineEmits(["update:modelValue"])  // 通知父组件  const selectedImage = ref("")  const confirmChooseImage = () => {  dialogVisible.value = false  **emit("update:modelValue", selectedImage.value)**  }    基于elementPlus这个动态tag，封装了自己的组件 |

10.2 Table表格，增加批量选中、删除

|  |
| --- |
| <el-table :data="tableData" v-loading="loading" style="width: 100%"  **@selection-change="handleSelectionChange"**>  **<el-table-column type="selection" width="55" />**  <el-table>  第一： 增加一个table-column,type=’selection’ **,** 这个就是多选框  第二： 点击多选框(选择、取消选择)会触发selection-change方法 |

10.3当table要展示的数据，在数据源中没有时，可以在获取数据源时，使用list.map,在每个对象中新增一个字段

|  |
| --- |
| const getData = (page) => {  if (typeof (page) === "number") {  currentPage.value = page  }  getCouponList(currentPage.value).then((res) => {  tableData.value = res.data.list.**map**(obj => {  // 增加自定义的'状态'字段  **obj.statusText = getStatusText(obj)**  return obj  })  totalCount.value = res.data.totalCount  })  }  <el-table-column **prop**="**statusText**" label="状态" width="180" /> |

### 十二.优惠劵模块

10.1 日期组件的使用

|  |
| --- |
| <el-date-picke  v-model="form.pickedTimeArr"  type="datetimerange"  range-separator="至" start-placeholder="开始日期" end-placeholder="结束日期" @change="handleSelectDate"  format="YYYY-MM-DD HH:mm:ss" value-format="YYYY-MM-DD HH:mm:ss" />  const form = reactive({  "order": 50,  "**pickedTimeArr**": [], // **临时变量，用于存放[start\_time, end\_time]**  "start\_time": "",  "end\_time": "",  })  // 在el-date-picker上确认选择时间后，触发的回调  const handleSelectDate = (val) => {  **if (val) {**  **form.start\_time = val[0]**  **form.end\_time = val[1]**  **}**  // 用户选择了cancel  return  }   1. v-model绑定的值是一个数组：[start\_time, end\_time]，时间格式可以是时间戳、Date对象、字符串等。 2. type="datetimerange"显示日期和时间; daterange显示日期 3. format格式是组件上肉眼看到的格式。 value-format是绑定到变量的值格式，默认是Date对象(即：console.log(new Date())， Tue Jul 22 2025 ...) 4. 组件上确认选择时间后，触发的change回调（这里将数组第一个值是选择的开始时间，第二个是结束时间） |

参考文档：

vue-router: <https://router.vuejs.org/zh/>

vuex: https://vuex.vuejs.org/zh/guide/

vue3: https://cn.vuejs.org/guide/introduction

**其他：**

1.如果store目录下有一个index.js文件，那么其他vue文件可以用下面简写的格式：

import {store} from '~/store'

而不必

import {store} from '~/store/index.js'

2. element-plus的所有内容在id=“app”中，它仅次于body下面。





1. 重点关注这3块东西（调试很有用）



第一块： 可以在页面指定看某个元素的信息。在页面选择一个元素后，可以查看它的标签名等等

第二块： styles， 可以看标签的css

第三块： 盒子模型，看宽度、高度、padding、margin等

1. 为什么全局设置box-sizing：border-box？

box-sizing: border-box 是 CSS 中一个非常重要的属性值，它决定了 **元素的宽高是如何计算的**。

一、默认行为（content-box）

box-sizing: content-box; /\* 默认值 \*/

宽高计算方式：

元素的总宽度 = content 宽度 + padding + border

例：

div {

width: 200px;

padding: 20px;

border: 10px solid black;

box-sizing: content-box;

}

实际占据宽度 = 200（内容） + 40（padding 左右） + 20（border 左右） = **260px**

二、推荐用法（border-box）

box-sizing: border-box;

宽高计算方式：

元素的总宽度 = 设置的宽度（包括 padding 和 border）

内容区宽度 = width - padding – border

例：

div {

width: 200px;

padding: 20px;

border: 10px solid black;

box-sizing: border-box;

}

实际占据宽度 = 200px，内容区宽度 = 200 - 40 - 20 = **140px**

**三、使用border-box的原因:**

* **更符合直觉**：设置宽度后，不会因 padding 和 border 撑大盒子。
* **布局更好控**：尤其在响应式布局和栅格系统中，尺寸更容易统一。
* **避免盒子“溢出”容器**的问题。

四、实践：

\* {

box-sizing: border-box;

}

1. 使用computed计算属性时，一定要写return

computed是只读属性，v-model需要一个可读可写的数据（即 ref）, 与computed的只读属性冲突。用可读可写的 computed 来适配 v-model：

|  |
| --- |
| const pickedTimeArr = computed({  **get**: () => {  if (form.start\_time && form.end\_time) {  return [form.start\_time, form.end\_time];  }  return [];  },  **set**: ([start, end]) => {  form.start\_time = start;  form.end\_time = end;  }  }); |

1. 后台管理的系统路由表定义

|  |
| --- |
| const routes = [  {  name: "login",  path: "/login",  component: myLoginView,  meta: {  title: "登录页"  }  } ,  {  name: "layoutIndex",  **path: "/",**  // 父路径  component: **layoutCompontent**,  children: [{  name: "index",  path: "/",  component: myIndexView,  meta: {  title: "首页"  }  },{  name: "goodsList",  **path: "/goods/list",**  component: GoodsList,  meta: {  title: "商品列表"  }  },{  name: "goodsList",  **path: "goods/add",**  component: GoodsList,  meta: {  title: "商品列表"  }  }  ]  }  ]  **layoutCompontent这个组件是全局布局页面，一般会包含header、侧边栏的菜单、主内容区域。path: "/"是它的路径，因为它有children, 所以它是父路径，对于chidren中最终路径的规则如下：**  路径前加 / 与不加 / 的区别：   1. **/goods/list**：是绝对路径，不会自动拼接父路径，它本身就是最终的访问路径 2. **goods/add**（没有 /）：是相对路径，会变成 /<父路径>/goods/add， 那最终路径是"//goods/add", vue-router会自动处理成合法路径，不能出现两个/,最终结果是： /goods/add   练习题：  {  path: '**/admin**', // 父路径是： /admin  children: [  {  path: 'dashboard', // 相对路径，**最终路径是: /admin/dashboard**  },  {  path: '**/**dashboard' // 绝对路径, **最终路径是: /dashboard**  }  ]  } |

1. 后端管理系统布局

header、 侧边菜单栏、 taglist要用**fixed**布局，



如果不用 fixed，滚动内容时 header/aside 会随页面一起滚动，体验很差。

如果你不为元素设置 position: fixed，它们默认是 position: static，它们会参与正常的文档流布局，这时候的 static 滚动行为是这样的：

1.页面所有元素都会随着整个 html / body 的滚动条滚动。

2.所以当你滚动 main 区域时，其实滚动的是整个 body。

3.所以header、aside 等元素也一起“跟着滚动”了（**被滚走，用户看不见**）

总结：

1.用户常操作头部的功能（如退出、头像、通知），不能被滚走。

2.用户常跳转菜单（侧栏导航），不能让菜单滚走。

3.标签页（TagList）要始终可见，不能随着主内容滚动。

所以后台系统使用 fixed 是为了实现： 头部 + 菜单栏 + tag标签栏固定，内容区可滚动的结构。 另外，fixed布局，会脱离文档流， 正常文档流里的内容可能会被挡住（当内容不显示时，排查一下是不是被fixed布局的内容挡住了。）

1. width=auto的使用

|  |
| --- |
| 1. 当global-aside是正常定位（即static）时。   <el-aside **width="auto"**><global-aside></global-aside></el-aside>  此时，设置宽度为auto，则父容器宽度会根据子内容自适应， el-aside原先默认的宽度就是无效了   1. 当global-aside是fixed定位时   <el-aside **width="auto"**><**global-aside**></global-aside></el-aside>  此时，由于 **global-aside已经脱离文档流了，那么el-side是没有子元素的，所以el-aside的宽度是0** |

1. 页面滑动时，fixed元素如何防止被覆盖

|  |
| --- |
| .tabList {  position: fixed;  top: 60px;  /\* 设置z-index和 background-color是为了在滑动主区域内容时， 标签栏不会被遮挡 \*/  z-index: 100; /\* 添加 z-index 确保在其他元素上方 \*/  background-color: #fff; /\* 添加背景白色 \*/  } |

1. 让元素靠右，或者居中 (前提： **要有明确的指定宽度，而不是由子内容决定**)

|  |
| --- |
| margin-left: auto; 【**auto 会让元素尽可能占用剩余空间**，元素会被“推”到右侧】  margin: 0 auto; 【左右都是auto，所以元素居中】 |

前提是： 父和子元素是block元素，如果是flex等布局的话，那么设置auto就无效

|  |
| --- |
| 比如： .title {  .el-pagination {  margin: 0 auto;  }  }  由于ElementPlus中的el-pagination是 flex布局，不是block，所以设置无效 |

引申： 单个元素居中，也可以用flex布局，比如：

<div class=”display: flex; justify-content: center; align-items: center”>

<div>我是要居中的元素</div>

</div>

1. 两种fixed布局

|  |
| --- |
| {  position: fixed;  top: 60px;  left: 300px;  **right: 0;**  }  设置了 left: 300px 和 right: 0，意思是：左边距离 300px，**右边对齐到浏览器右侧**（也就是**宽度由系统自动计算撑满剩余区域**，这种算是有明确的宽度，不是由子内容决定）  {  position: fixed;  top: 60px;  left: 300px;  }  没有指定右边界，默认宽度由内容决定。 此时元素宽度由：你是否手动设置 width、  元素内容宽度、、父级约束等来共同决定  注：**不会自动撑满剩余空间**  📌 示例用途：你想放一个左边浮动的小组件，比如通知框、浮窗等。 |

1. style 设置了scope时，如何修改其他文件的css？

使用 **:**deep 和 !important

1. 动画效果

<http://animate.style>

淡入： 颜色，由淡慢慢的变深， 所以叫淡入。其实就是透明度从 透明到慢慢的不透明

透明度： 本质上元素的可见程度.

不透明表示：**不能透过这个元素看到其后面的内容**，它是完全遮挡住下层的。

透明：**透过这个元素看到其后面的内容**，像玻璃一样

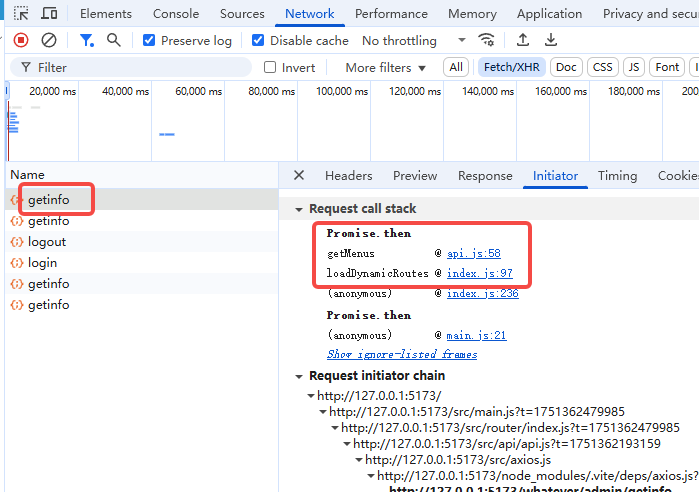
opacity 取值范围是 0 ~ 1

1 → 完全不透明（默认）

0.5 → 半透明

0 → 完全透明（看不见）

1. network选项中，看http请求的调用堆栈



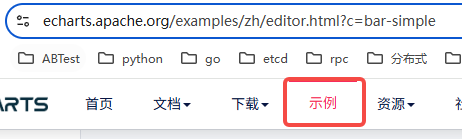
上面先后发了两次getinfo请求，可以看堆栈是否从哪里调用过来的

1. 边框： 圆角+阴影效果

background: #fff;

border-radius: 8px;

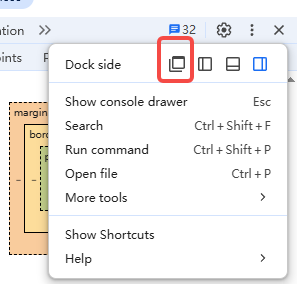
box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);



https://echarts.apache.org/handbook/zh/get-started/

https://echarts.apache.org/examples/zh/index.html

1. 浏览器的调试窗口，用 独立窗口 的模式打开，这样更方便审查元素，可以同时看“Element”、“Styles”、“Computed”. 【非常好用，五星推荐】



1. 图片高度不一样，如何调整样式?

|  |
| --- |
| <el-row :gutter="20">  <**el-col** :span="6" v-for="(item, index) in list" :key="index">  <el-card shadow="hover">  **<el-image :src="item.url" fit="fill" :lazy="true"></el-image>**  </el-card>  </el-col>  </el-row> |

分析： 图片哪里宽、高不一致就调整嘛。 **高矮不一样，那就指定大家用固定高度嘛**，比如： height:200px, 使用el-col已经平均分成4等份了，直接继承el-col的宽度就行。也就是对el-image加一个style=”width: 100%; height: 200px”

18.为什么 v-loading 前不需要加 冒号，就可以动态绑定

|  |
| --- |
| <el-table v-loading="loading" :data="tableData" style="width: 100%">  const loading = ref(true)  v-loading 是 Element Plus 的一个 自定义指令（directive），而不是组件的 prop 属性。  Vue 中，只有绑定属性时才需要用 :（语法糖，相当于 v-bind:）。  而 v-loading 是一个 指令，类似于 v-if、v-show，不需要 :。 |

19.@click=“myfun”使用

|  |
| --- |
| @click=“myfun**()**” => 无参调用  @click=“myfun**(a,b)**” => 有参调用，有且只能有两个  @check=“myfun” => 因为 TreeV2组件定义的事件，定义了会传2个参数，所以vue会自动将2个参数带过来，如下：    那定义myfunc:  const myfun = (**data, info**) => {xxxx}  如果希望在@check事件中加入自己的参数，可以使用闭包：  @check=“（data,info） => myfun3(data, info, **otherParams**)” |

20.div不一定就是block

.el-input给div加了一个 display: inline-flex，所以它是行内元素，不是block块元素

