### TODO： 项目实战或练习，才会记住。

### 一.环境安装

#### 1．1使用vite创建项目

|  |
| --- |
| # 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  ) |

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  }) |

3.7 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(“/”) |

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);    }); |