河南中医药大学信息技术学院（智能医疗行业学院）智能医学工程专业《互联网医疗服务开发》课程

第11章：Axios与框架

冯顺磊

河南中医药大学信息技术学院（智能医疗行业学院）

河南中医药大学信息技术学院智能医疗教研室

https://aitcm.hactcm.edu.cn

2025/5/9

本章概要

- Axios
- 状态管理
- 使用View UI Plus实现管理系统
- 使用Vant实现移动端应用

1. Axios 1.1 Axios简介

- 在实际开发过程中，浏览器通向需要和服务器交互，获取数据，Vue官方并未提供与服务器通信的接口，但官方推荐使用Axios来实现AJAX请求，Axios是一个基于Promise的网络请求库，用于在浏览器中以及在 Node.js 环境中发送 HTTP 请求，其本质是对原生XHR的封装，是Promise的实现版本，符合最新的ES规范。Axios能很好的与各种前端框架整合，其特点如下。
  - 可用浏览器中创建XMLHttpRequests
  - 可用从node.js创建http请求
  - 支持Promise API
  - 支持拦截请求和响应
  - 转换请求数据和响应数据
  - 取消请求
  - 自动转换JSON数据
  - 客户端支持防御XSRF

1. Axios 1.2 Axios安装

- 使用包管理工具安装

JavaScript ▾ 固定高度 复制 设置

```
1 npm install axios
```

- 直接引入CDN

JavaScript ▾ 固定高度 复制 设置

```
1 <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

JavaScript ▾ 固定高度 复制 设置

```
1 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

1. Axios 1.2 Axios安装

- 安装完成后，可在代码中引入 Axios 验证是否成功。

JavaScript ▾ 固定高度 复制 设置

```
// Node.js 或模块化环境
const axios = require('axios');

// 浏览器环境（直接引入 CDN 后，全局可用）
console.log(axios); // 输出 Axios 对象即表示安装成功
```

1. Axios 1.3 Axios API

- Axios默认暴漏的只有一个函数axios，执行axios函数可以发送AJAX请求，同时axios函数提供一些发送不同类型AJAX请求的静态方法，如get、post、put、delete等。
  - 通过axios配置创建请求
    - 使用axios(config)创建请求，示例代码如下。

JavaScript ▾ 固定高度 复制 设置

```
// 发送POST请求
axios({
    method:'post',
    url:'/uesr/12345',
    data:{
    firstName:'first',
    lastName:'last',
    }
});
```

1. Axios 1.3 Axios API

- 通过axios配置创建请求
  - 使用axios(url[,config])创建请求，示例代码如下。

JavaScript ▾ 固定高度 复制 设置

```
1 // 发送GET请求（默认方法）
2 axios('/user/12345');
```

1. Axios 1.3 Axios API

- 使用方法别名创建请求
  - axios为所有支持的请求方法提供了别名，如下所示。

JavaScript ▾ 固定高度 复制 设置

```
1 axios.request(config)
2 axios.get(url[,config])
3 axios.delete(url[,config])
4 axios.head(url[,config])
5 axios.options(url[,config])
6 axios.post(url[,data[,config]])
7 axios.put(url[,data[,config]])
8 axios.patch(url[,data[,config]])
9 axios.postForm(url[,data[,config]])
10 axios.putForm(url[,data[,config]])
11 axios.patchForm(url[,data[,config]])
```

1. Axios 1.3 Axios API

- 处理并发请求
  - axios使用all、spread函数处理并发请求，示例代码如下。

JavaScript ▾ 固定高度 复制 设置

```
1 axios.all(iterable)
2 axios.spread(callback)
```

1. Axios 1.3 Axios API

- 创建实例
  - 使用自定义配置创建一个axios实例，示例代码如下。

JavaScript ▾ 固定高度 复制 设置

```
1 const instance = axios.create({
2     baseURL:'https://some-domain.com/api/',
3     timeout:1000,
4     headers:{'X-Custom-Header': 'foobar'}
5 });
```

1. Axios 1.4 Axios的请求配置

- Axios创建请求时，需要设置配置选项，其中只有“url”是必需的，如果没有指定“method”，请求将默认使用“get”方法，常见的配置选项说明如下。

JavaScript ▾ 固定高度 复制 设置

```
{
    // url 是用于请求的服务器URL
    url:'/user',

    // method 是创建请求时使用的方法
    method:'get',

    // baseURL 将自动加在url前面，除非url是一个绝对URL
    // 它可以通过设置一个baseURL 便于为axios实例的方法传递相对URL
    baseURL:'https://some-domain.com/api/',

    // headers 是即将被发送的自定义请求头
    headers: {'x-Requested-With':'XMLHttpRequest'},

    // params 是即将与请求一起发送的URL参数
```

1. Ax

- Axios 请
  求将黑

```
15      // params 是即将与请求一起发送的URL参数
16      // 必须是一个无格式对象（plain object）或 URLSearchParams 对象
17      params:{
18          ID:12345,
19      },
20
21      // data 是作为请求主体被发送的数据，只适用于这些请求方式：PUT、POST和PATCH
22      // 在没有设置转换请求时，必须是以下类型之一
23      // string，plain object,ArrayBuffer,ArrayBufferView,URLSearchParams
24      // 浏览器专属：FormData,File,Blob
25      // Node 专属：Stream
26      data:{
27          firstName:'First',
28      },
29
30      // timeout 指定请求超时的毫秒数（0 表示无超时时间）
31      // 如果请求花费了超过 timeout 的时间，请求将被中断
32      timeout:1000,
33
34      // responseType 表示服务器响应的数据类型，可以是以下几种
35      // arraybuffer、blob、document、json、text、stream
36      responseType:'json',    // default
37
38      // proxy 定义代理服务器的主体名称和端口
39      // auth 表示 HTTP 基础验证应当用于连接代理，并提供凭据
```

1. Ax

- Axios 请求将黑

```
27          firstName: 'First',
28      },
29
30      // timeout 指定请求超时的毫秒数（0 表示无超时时间）
31      // 如果请求花费了超过 timeout 的时间，请求将被中断
32      timeout:1000,
33
34      // responseType 表示服务器响应的数据类型，可以是以下几种
35      // arraybuffer、blob、document、json、text、stream
36      responseType:'json',    // default
37
38      // proxy 定义代理服务器的主体名称和端口
39      // auth 表示 HTTP 基础验证应当用于连接代理，并提供凭据
40      // 这将会设置一个 Proxy-Authorization 头，覆写掉已有的通过使用 header 设置的自定义
41      // Proxy-Authorization 头
42      proxy: {
43          host: '127.0.0.1',
44          port: 9000,
45          auth: {
46              username: 'mikeymike',
47              password: 'rapunz31'
48          }
49      },
50  }
```

1. Axios 1.5 Axios的响应结构

- 来自某个请求的响应应包含以下信息。

JavaScript ▾ 固定高度 复制 设置

```
{
    // data 由服务器提供的响应
    data: {},

    // status 来自服务器响应的 HTTP 状态码
    status: 200,

    // statusText 来自服务器响应的 HTTP 状态信息
    statusText: 'OK',

    // headers 服务器响应的头
    headers: {},

    // config 是为请求提供的配置信息
    config: {},

    // request 是生成当前响应的请求
    // 在 node.js 中是最后一个 ClientRequest 实例（在重定向中）
    // 在浏览器中是 XMLHttpRequest 示例
    request: {},
}
```

1. Axios 1.5 Axios的响应结构

- 使用“then”时，将接收下面这样的响应。当使用 catch，或者传递一个rejection callback作为 then 的第二个参数时，响应可以通过 error 对象被使用。

JavaScript ▾ 固定高度 复制 设置

```
1 axios.get('/user/12345')
2     .then(function(response){
3         console.log(response.data);
4         console.log(response.status);
5         console.log(response.statusText);
6         console.log(response.headers);
7         console.log(response.config)
8     })
```

1. Axios 1.6 Axios的默认配置

- 全局axios默认值

JavaScript ▾ 固定高度 复制 设置

```
1 axios.defaults.baseURL = 'https://api.example.com';
2 axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
3 axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlenco
```

- 自定义示例默认值

JavaScript ▾ 固定高度 复制 设置

```
1 // 创建实例时配置默认值
2 const instance = axios.create({
3   baseURL: 'https://api.example.com'
4 });
5
6 // 创建实例后修改默认值
7 instance.defaults.headers.common['Authorization'] = AUTH_TOKEN;
```

1. Axios 1.6 Axios的默认配置

- 配置的优先级
  - 配置将会按优先级进行合并。它的顺序是：在lib/defaults.js中找到的库默认值，然后是实例的 defaults 属性，最后是请求的 config 参数。后面的优先级要高于前面的。

JavaScript ▾ 固定高度 复制 设置

```
// 使用库提供的默认配置创建实例
// 此时超时配置的默认值是 `0`
const instance = axios.create();

// 重写库的超时默认值
// 现在，所有使用此实例的请求都将等待2.5秒，然后才会超时
instance.defaults.timeout = 2500;

// 重写此请求的超时时间，因为该请求需要很长时间
instance.get('/longRequest', {
  timeout: 5000
});
```

1. Axios 1.7 Axios的拦截器

- 在请求或响应被 then 或 catch 处理前拦截它们。

JavaScript ▾ 固定高度 复制 设置

```
// 添加请求拦截器
axios.interceptors.request.use(function (config) {
    // 在发送请求之前做些什么
    return config;
  }, function (error) {
    // 对请求错误做些什么
    return Promise.reject(error);
  });

// 添加响应拦截器
axios.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;
  }, 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
    return Promise.reject(error);
  });
```

1. Axios 1.8 Axios的错误处理

JavaScript ▾ 固定高度 复制 设置

```
axios.get('/user/12345')
  .catch(function (error) {
    if (error.response) {
      // 请求成功发出且服务器也响应了状态码，但状态代码超出了 2xx 的范围
      console.log(error.response.data);
      console.log(error.response.status);
      console.log(error.response.headers);
    } else if (error.request) {
      // 请求已经成功发起，但没有收到响应
      // `error.request` 在浏览器中是 XMLHttpRequest 的实例，
      // 而在node.js中是 http.ClientRequest 的实例
      console.log(error.request);
    } else {
      // 发送请求时出了点问题
      console.log('Error', error.message);
    }
    console.log(error.config);
  });
```

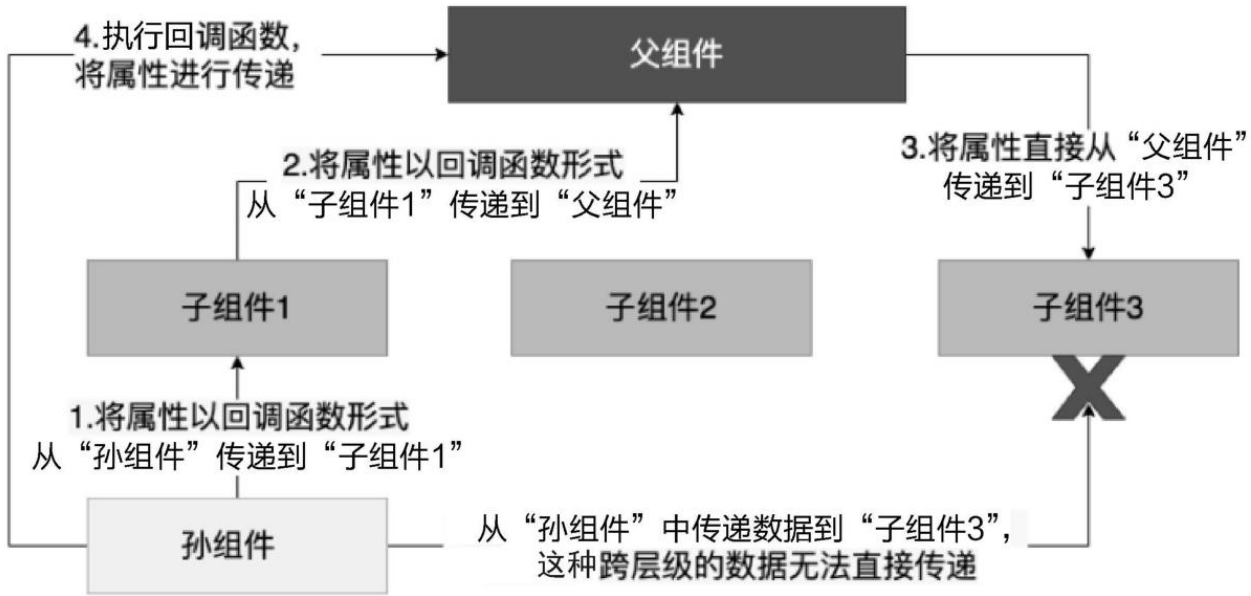
1. Axios 1.9 Axios的进一步封装

- 在项目src目录下新建api文件夹，然后在其中新建 request.js文件，主要实现axios的封装过程。
- 在项目src目录下的api文件夹中新建 http.js文件，实现几种请求方式的封装。
- 在项目src目录下新建api文件夹，然后在其中新建 api.js文件，主要实现API的封装。当一个项目中后台请求不是同一个IP地址，是多个IP地址的时候，可以在api文件夹下创建多个js文件，用来调用数据请求。

2. 状态管理 2.1 Vuex

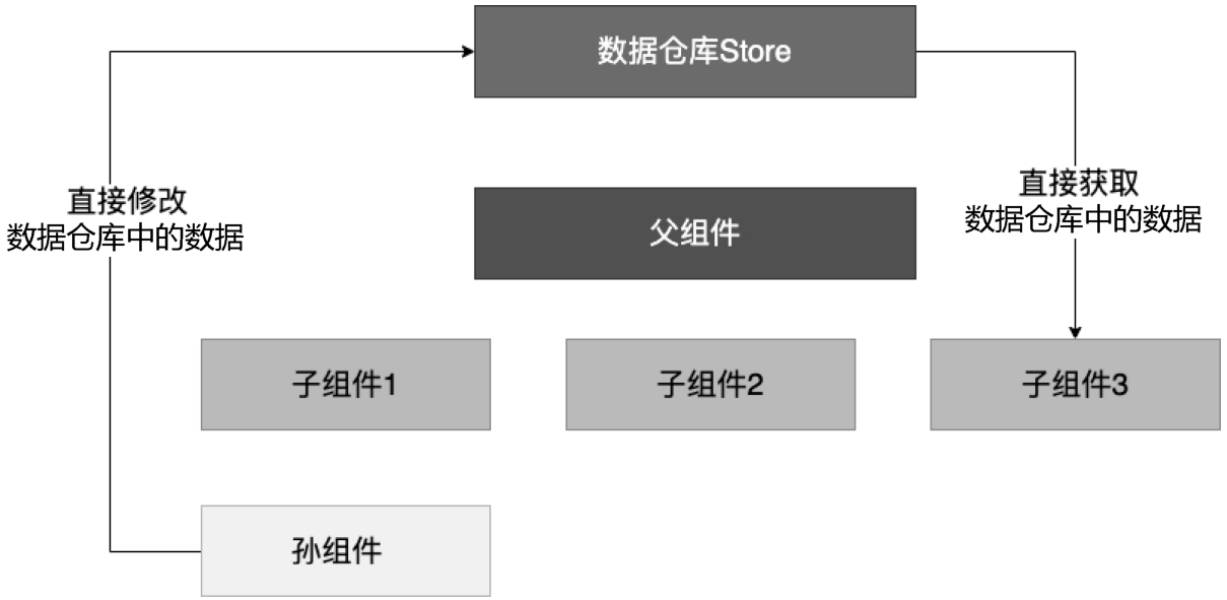
- 随着项目复杂度的增加，组件之间的层次嵌套成了一种十分常见的现象。
- 在一个项目中，嵌套层次少的可能有两三层，多的可能达到五六层。
- 在Vue的组件通信关系中，Vue组件通信尽可能使用单向流程处理，但是这样的传递过程明显很漫长，不仅流程烦琐，而且效率很低，因此，需要且必须找寻一种更合理的解决方案来优化组件之间的通信工作。

利用props属性传递实现组件通信



2. 状态管理 2.1 Vuex

- Vuex 是 Vue.js 的官方状态管理库，主要用于管理应用的共享状态（如用户登录信息、主题模式等），尤其适用于中大型单页应用。
- Vuex采用集中式存储管理应用的多个组件之间的共享状态数据，并以相应的规则保证状态以一种可预测的方式发生变化。
- 利用Vuex可以存储任意层级的组件数据内容，在其他任意层级的组件需要对应的数据时，只需直接从其数据仓库中获取与修改即可。



2. 状态管理 2.2 Vuex的核心概念

- Vuex的核心概念如下。

概念	作用
State	存储应用的核心状态（数据源），相当于组件中的 data，但全局共享。
Getter	对 State 进行计算或过滤（类似组件的 computed），避免重复逻辑。
Mutation	唯一修改 State 的方式（同步操作），需通过 commit 触发。
Action	处理异步逻辑（如 API 请求），最终通过提交 Mutation 修改 State，需通过 dispatch 触发。
Module	模块化状态管理，将复杂应用的状态按功能拆分为多个模块，避免代码臃肿。

2. 状态管理 2.3 Vuex的基础应用

- 安装Vuex

Plain Text ▾ 固定高度 复制 设置

```
1 npm install vuex@4   # Vue 3 需搭配 Vuex 4+
```

2. 状态管理 2.3 Vuex的基础应用

- 创建 Vuex 仓库（Store），在 store/index.js 中定义状态、修改逻辑和异步操作。

JavaScript ▾ 固定高度 复制 设置

```
import { createStore } from 'vuex';

export default createStore({
    // 1. 全局状态（数据源）
    state() {
        return {
            count: 0,        // 计数器数值
            isLoading: false // 加载状态（用于异步场景）
        };
    },

    // 2. Getter：派生状态（基于 State 的计算）
    getters: {
        // 判断 count 是否为偶数（返回布尔值）
        isEven: (state) => state.count % 2 === 0,
        // 带单位的数值（演示 Getter 接收参数）
```

2. 状

- 创建

```
20        // 3. Mutation：同步修改 State（唯一方式）
21        mutations: {
22            // 增加计数器（接收 payload 作为参数，默认 +1）
23            increment(state, payload = 1) {
24                state.count += payload;  // 直接修改 State（同步）
25            },
26            // 切换加载状态（接收布尔值）
27            toggleLoading(state, isLoading) {
28                state.isLoading = isLoading;
29            }
30        },
31
32        // 4. Action：异步操作（最终通过 Mutation 修改 State）
33        actions: {
34            // 模拟异步请求后增加计数器（如调用 API）
35            async asyncIncrement({ commit }) {
36                commit('toggleLoading', true);  // 开始加载（触发 Mutation）
37                await new Promise(resolve => setTimeout(resolve, 1000));  // 模拟
38                commit('increment', 2);          // 提交 Mutation（增加 2）
39                commit('toggleLoading', false); // 结束加载（触发 Mutation）
40            }
41        }
42    });
43
```

2. 状态管理 2.3 Vuex的基础应用

- 在组件中使用 Vuex 状态

JavaScript ▾ 固定高度 复制 设置

```
<template>
    <div class="counter">
        <h2>Vuex 计数器</h2>
        <p>当前数值：{{ count }}</p>
        <p>是否为偶数：{{ isEven ? '是' : '否' }}</p>
        <p>数值（带单位）：{{ countWithUnit('次') }}</p>
        <p v-if="isLoading">加载中...</p>

        <!-- 触发同步 Mutation -->
        <button @click="handleIncrement">+1（同步）</button>
        <!-- 触发异步 Action -->
        <button @click="handleAsyncIncrement">+2（异步）</button>
    </div>
</template>

<script>
import { useStore } from 'vuex';
import { computed } from 'vue';

export default {
    setup() {
```

2. 状态管理

- 在组件中使用 V

```
import { useStore } from 'vuex';
import { computed } from 'vue';

export default {
    setup() {
        const store = useStore();  // 获取 Vuex 仓库实例

        // 响应式获取 State（自动追踪变化）
        const count = computed(() => store.state.count);
        const isLoading = computed(() => store.state.isLoading);

        // 响应式获取 Getter（自动依赖 State 变化）
        const isEven = computed(() => store.getters.isEven);
        const countWithUnit = computed(() => store.getters.countWithUnit);

        // 触发同步 Mutation（修改 State）
        const handleIncrement = () => {
            store.commit('increment');  // 无参数时默认 +1
            // 也可传递参数：store.commit('increment', 3) 则 +3
        };

        // 触发异步 Action（通过 Mutation 间接修改 State）
        const handleAsyncIncrement = () => {
            store.dispatch('asyncIncrement');  // 调用异步操作
        };

        return { count, isEven, countWithUnit, isLoading, handleIncrement, handle
    }
};
</script>
```

2. 状态管理 2.3 Vuex的基础应用

- 关键细节
  - 状态的响应式
    - 必须通过 computed 包裹：直接使用 store.state.count 不会触发组件更新，必须用 computed(() => store.state.count) 包裹，确保响应式。
    - 禁止直接修改 State：直接赋值 store.state.count = 10 会导致状态不可追踪（严格模式下报错），必须通过 Mutation 修改。
  - Mutation 的同步性
    - 只能同步操作：Mutation 中不能写异步代码（如 setTimeout、API 请求），否则会导致状态修改顺序混乱。
    - 通过 commit 触发：必须用 store.commit('mutationName', payload) 调用 Mutation，确保修改可追踪（Vue DevTools 可记录）。
  - Action 的异步处理
    - 支持异步逻辑：Action 中可写 async/await 或 Promise（如调用 API），最终通过 commit 提交 Mutation。
    - 通过 dispatch 触发：用 store.dispatch('actionName', payload) 调用 Action，支持返回 Promise（可链式调用）

JavaScript ▾ 固定高度 复制 设置

```javascript
// 组件中调用异步 Action 并等待完成
const handleAsync = async () => {
    await store.dispatch('asyncIncrement');
    console.log('异步操作完成');
};
```

2. 状态管理 2.3 Vuex的基础应用

- 关键细节
  - 模块化（Module）
    - 对于大型应用，建议按功能拆分状态（如 user、cart 模块），避免根仓库代码臃肿

JavaScript ▾ 固定高度 复制 设置

```javascript
// store/modules/user.js（用户模块示例）
export default {
    namespaced: true,  // 启用命名空间（推荐，避免命名冲突）
    state() {
        return { username: '游客', isLogin: false };
    },
    mutations: {
        login(state, username) {  // 模块内的 Mutation
            state.username = username;
            state.isLogin = true;
        }
    }
};

// store/index.js（合并模块到根仓库）
import { createStore } from 'vuex';
import user from './modules/user';

export default createStore({
    modules: {
        user  // 模块会被合并到根 State（如 store.state.user.username）
    }
});
```

JavaScript ▾ 固定高度 复制 设置

```javascript
// 获取模块状态（带命名空间）
const username = computed(() => store.state.user.username);
// 提交模块 Mutation（需指定命名空间）
store.commit('user/login', '登录');  // 格式：'模块名/mutation名'
```

2. 状态管理 2.4 Vuex的持久化存储

- Vuex 的状态默认存储在内存中，页面刷新后会丢失。持久化存储的核心是将 Vuex 的状态同步到本地存储（如 localStorage、sessionStorage 或 IndexedDB），并在应用初始化时恢复状态，解决 “刷新页面后状态丢失” 的问题。
- 为什么需要持久化存储?
  - 用户登录后，刷新页面需要保留登录状态（如 userInfo）。
  - 购物车商品添加后，关闭浏览器再打开仍需保留数据。
  - 主题模式（如暗 / 亮色）需要跨会话保存。

2. 状态管理 2.4 Vuex的持久化存储

- vuex-persistedstate 是 Vuex 官方推荐的持久化插件，支持灵活配置存储引擎（默认 localStorage）、自定义存储路径、过滤不需要持久化的状态等。

Plain Text ▾ 固定高度 复制 设置

```
1 npm install vuex-persistedstate  # 安装持久化插件
```

2. 状态管理 2.4 Vuex的持久化存储

- 在 store/index.js 中引入插件并集成到 Vuex 仓库
  - 所有状态会被自动同步到 localStorage，键名为 vuex（默认），刷新页面后状态自动恢复。

- 注意事项
  - 存储大小限制：localStorage 通常有 5MB 大小限制，避免存储过大数据（如大数组）。
  - 敏感数据保护：避免直接存储 token 等敏感信息，如需存储需加密（如上示例）。
  - 服务端渲染（SSR）：若项目使用 SSR（如 Nuxt.js），需通过 paths 排除服务端特有状态，避免客户端 / 服务端状态不一致。
  - 模块化支持：若 Vuex 使用了模块化（modules），paths 需指定模块路径（如 paths: ['user.info', 'cart.items']）。

JavaScript ▾ 固定高度 复制 设置

```javascript
import { createStore } from 'vuex';
import createPersistedState from 'vuex-persistedstate';  // 导入插件

export default createStore({
    state() {
        return {
            user: { name: '游客', isLogin: false },
            cart: [{ id: 1, count: 0 }],
            temp: '临时状态（不持久化）'
        };
    },
    mutations: {
        login(state, username) {
            state.user = { name: username, isLogin: true };
        },
        addToCart(state, { id, count }) {
            const item = state.cart.find(i => i.id === id);
            if (item) item.count += count;
        }
    },
    // 集成持久化插件
    plugins: [
        createPersistedState()  // 默认将所有状态存入 localStorage
    ]
});

```

2. 状态管理 2.5 Vuex的适用场景

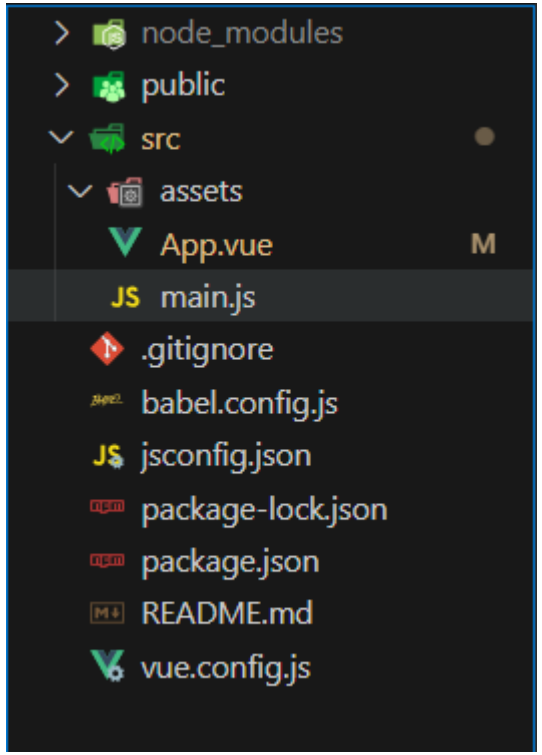
- Vuex 适用于中大型应用，尤其是以下情况：
  - 多个组件需要共享同一状态（如用户登录状态、主题模式）。
  - 状态需要跨多层级传递（如父组件 → 孙子组件）。
  - 需要追踪状态修改历史（通过 Vue DevTools 查看每次 Mutation）。
  - 对于小型应用（状态简单、组件层级浅），直接使用 props、emit 或 Vue 3 的 provide/inject 可能更轻量，避免过度设计。

3. 使用View UI Plus实现管理系统 3.1 View UI Plus简介

- View UI Plus是一套基于Vue.js 3的高质量UI组件库，为开发者提供简洁、美观且功能丰富的用户界面组件，帮助开发者更高效地构建现代化的Web应用程序和管理系统。
- View UI Plus的优点
  - 组件丰富：提供大量高质量的组件，涵盖了从基础的按钮、输入框到复杂的表格、图表等各种类型。
  - 文档详尽：提供了全面的文档和丰富的示例，涵盖了组件的各个方面，包括属性、事件、插槽等。
  - 支持新特性：支持组合式API、Teleport等。
  - 按需引入：可根据实际引入组件，避免了引入整个组件库而导致的项目体积过大，有助于提高项目的加载速度和性能。
- View UI Plus的缺点
  - 对于一些刚接触Vue.js3和该组件库的开发人员来说，可能存在一定的学习曲线。虽然文档较为完善，但由于组件库的功能较为丰富和复杂，需要花费一定的时间来熟悉和掌握各个组件的用法和特性。
  - 在进行深度自定义时，可能会有一定的难度。虽然组件库提供了一些自定义的选项，但对于一些复杂的定制需求，可能需要深入了解组件的实现原理和代码结构，才能进行有效的自定义，这对于一些开发人员来说可能是一个挑战。

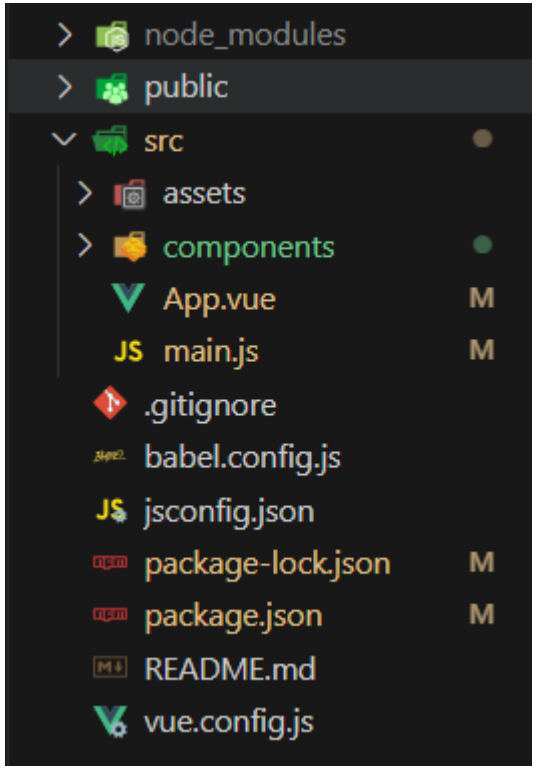
3. 使用View UI Plus实现管理系统 3.2 实现管理系统

- 步骤1：创建名为“manage”的项目
- 步骤2：打开“manage”项目，删除“src\assets”、“src\components”目录下的示例文件。
- 步骤3：打开终端输入“npm install view-ui-plus --save”安装View UI Plus。



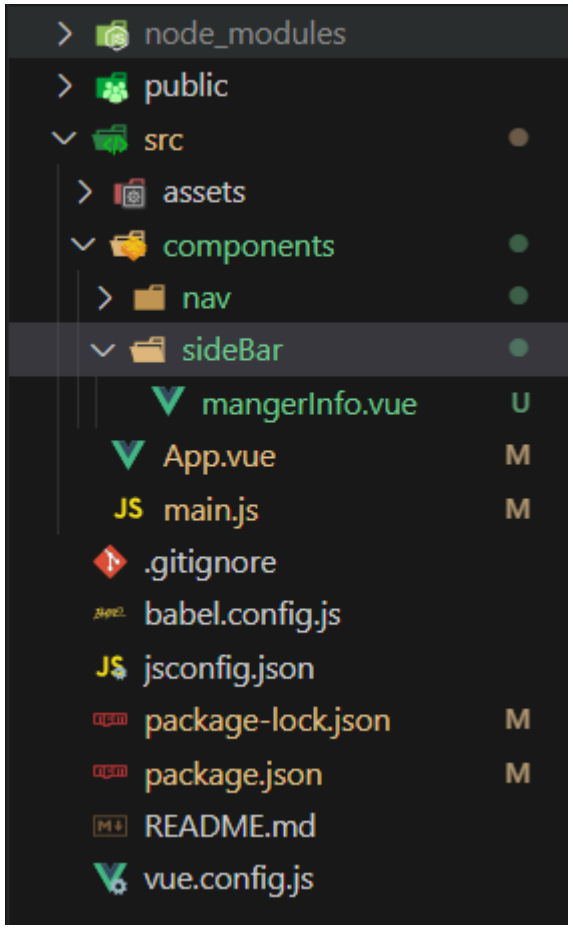
3. 使用View UI Plus实现管理系统 3.2 实现管理系统

- 步骤4：在“src\components”目录下创建管理系统导航栏文件夹“src\components\nav”，其中包含了两个文件，分别是“src\components\nav\personnalInfo”和“src\components\nav\addMangerInfo”。



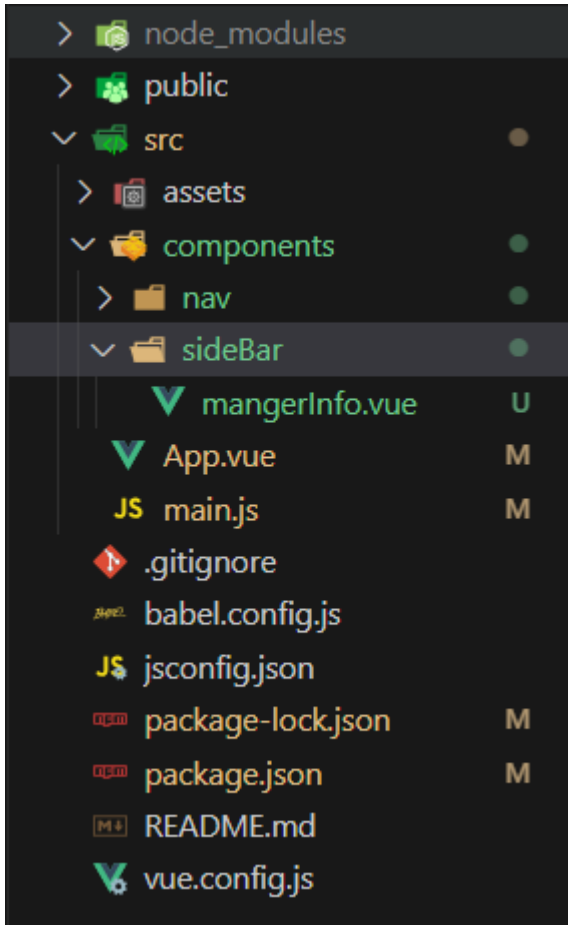
3. 使用View UI Plus实现管理系统 3.2 实现管理系统

- 步骤5：在“src\components”目录下创建管理系统侧边栏文件夹“src\components\sideBar”，目前只用到一个侧边栏所以只有一个文件“src\components\sideBar\mangerInfo”。



3. 使用View UI Plus实现管理系统 3.2 实现管理系统

- 步骤6：编辑“App.vue"文件。



4. 使用Vant实现移动端应用 4.1 Vant简介

- Vant 是一个基于 Vue.js 的移动端 UI 组件库，由有赞开源。它旨在为开发者提供高质量、易于使用的组件，帮助快速构建高性能的移动应用
- Vant的特点
  - 轻量级：Vant 的组件经过严格的压缩和优化，确保在保证性能的同时，减小应用的体积。
  - 易用性：提供详细的文档和示例，帮助开发者快速上手。组件设计简洁明了，易于理解和使用。
  - 高度可定制：支持自定义样式和属性，满足不同项目的需求。
  - 跨平台：基于 Vue.js 开发，可以运行在 iOS、Android 和 Web 等多个平台。
  - 社区支持：拥有活跃的社区，开发者可以在社区中寻求帮助和分享经验。

4. 使用Vant实现移动端应用 4.1 实现移动端应用

- 步骤1：创建名为“app”的项目
- 步骤2：打开“app”项目，删除“src\assets”、“src\components”目录下的示例文件。
- 步骤3：打开终端输入“npm i vant --save”安装Vant。
- 步骤4：开发Vant应用。

信创智能医疗系统研发课程体系

河南中医药大学信息技术学院（智能医疗行业学院）

河南中医药大学信息技术学院（智能医疗行业学院）智能医疗教研室

河南中医药大学医疗健康信息工程技术研究所