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10章：组件与路由

冯顺磊

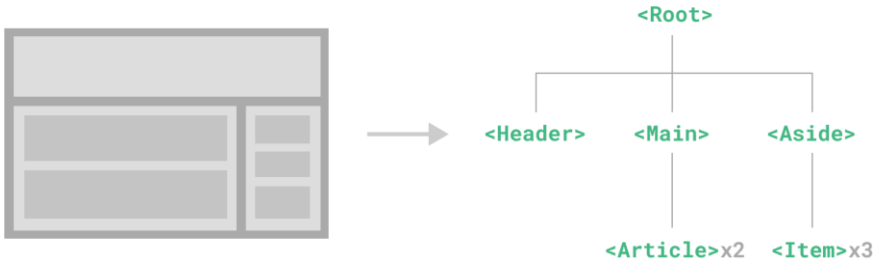
河南中医药大学信息技术学院（智能医疗行业学院）
河南中医药大学信息技术学院智能医疗教研室
https://aitcm.hactcm.edu.cn
2025/4/18

本章概要

- 组件
  - 创建项目
  - 组件简介
  - 组件注册
  - 组件通信
- 路由
  - 创建项目
  - 路由简介
  - Vue Router
  - 路由使用

1. 组件 1.1 组件简介

- 组件是将用户界面（UI）拆分并封装成独立、可复用、带有各自的逻辑的模块。
- 每个组件都是一个独立的 Vue 实例，具有独立的模板、数据、方法和生命周期钩子，因此组件可以定义和管理自身的功能和样式。



1. 组件 1.1 组件简介

- 组件的特性如下：
  - 模块化设计：组件开发是将HTML、CSS和JavaScript代码封装成组件内部的状态管理和逻辑处理，成为相互隔离的独立模块。
  - 可复用性：组件被注册或引用后，即可在应用的多个场景中重复利用。通过参数化配置，同一个组件可以展现出不同的形态和行为。
  - 嵌套结构：Vue 支持组件的嵌套使用，构建出层次分明的组件树，这种结构使得用户界面可以被拆分成多个小型的、易于管理的后代组件。
  - 生命周期钩子函数：生命周期钩子（Lifecycle Hooks）覆盖组件的整个生命周期，从组件实例化前（beforeCreate），组件创建并初始化结束（created），组件模板被编译并挂载到DOM前（beforeMount）和挂载到DOM后（mounted），响应式数据更新周期（beforeUpdate、updated），直至组件被销毁前（beforeUnmount）和销毁后（unmounted）。
  - 事件通信机制：Vue 组件支持自定义事件的发送与监听，实现组件间的有效通信。

1. 组件 1.1 组件简介

- Vue 的组件系统允许开发者使用独立可复用的组件来构建复杂的页面。编写组件时应注意以下组件定义规范。
  - 单一职责原则：一个组件只负责一件事情，保证组件的简洁性和可维护性。
  - 清晰的层次结构：对于较大的组件，可以拆分为更小的子组件，以提高复用性和可读性。
  - 使用props进行数据传递：父组件可以通过props向子组件传递数据。
  - 使用事件回调进行通信：子组件可以通过触发事件“emit”来与父组件进行通信。

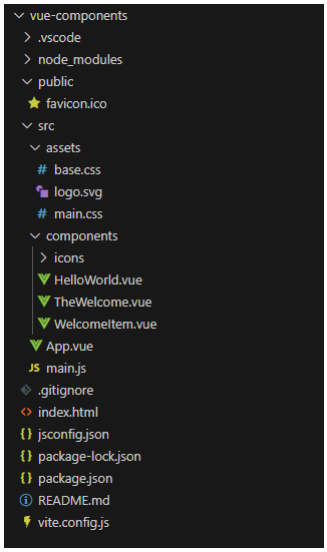
1. 组件 1.1 组件简介

- Vue 组件的主要结构由三部分组成。
  - 模板（Template）：模板用来编辑组件的HTML结构
  - 逻辑（Script）：逻辑是负责组件的JavaScript逻辑和数据等
  - 样式（Styles）：样式是组件的CSS样式。
- 构建个人信息展示组件示例来说明组件基本结构，示例代码如右所示。

```
<template>
<!-- HTML结构 -->
    <div class="info">
      <h3>{{ infoData.name }}</h3>
      <div class="num">
        <p>
          <span>文章</span><span>{{ infoData.essayNum }}</span>
        </p>
        <p>
          <span>标签</span><span>{{ infoData.labelNum }}</span>
        </p>
        <p>
          <span>分类</span><span>{{ infoData.classifyNum }}</span>
        </p>
      </div>
    </div>
</template>
<script setup>
//JavaScript逻辑
console.log('个人信息展示组件');
//数据声明
const infoData = {
  name: "MyGlog",
  essayNum: 10,
  labelNum: 10,
  classifyNum: 10,
};
</script>
<style scoped>
/* 局部样式 */
</style>
```

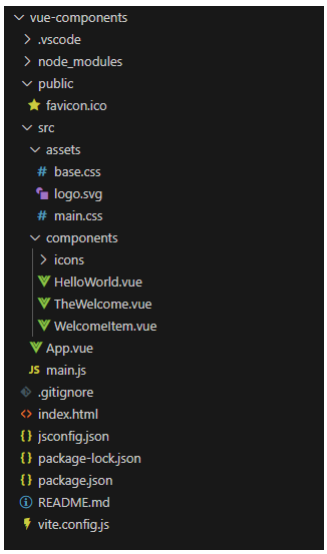
1. 组件 1.2 创建项目

- 创建项目
  - 创建示例项目“vue-components”
- 结构介绍
  - public/ 目录
    - 作用：存放无需构建的静态资源（如 favicon.ico、图片等），这些文件会直接复制到构建输出目录。
    - 注意：如果引用此目录下的文件，需使用绝对路径（如 /favicon.ico）。
  - src/ 目录
    - 核心代码存放位置，包含组件、入口逻辑和资源。
      - assets/
        - 存放需要经过构建工具处理的静态资源，如：
        - 全局 CSS 文件（如 base.css）
        - 图片、字体等（构建时会优化文件名或体积）。
      - components/
        - 存放可复用的 Vue 组件（如 Button.vue、Header.vue）。



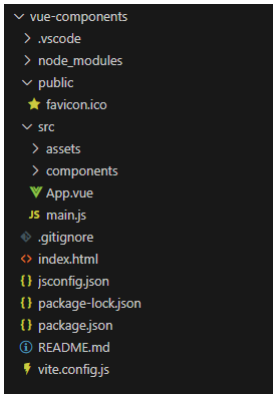
1. 组件 1.2 创建项目

- 结构介绍
  - src/ 目录
    - 核心代码存放位置，包含组件、入口逻辑和资源。
      - App.vue
        - 根组件，所有其他组件会在此组件内渲染。
      - main.js
        - 应用入口文件，创建 Vue 应用实例并挂载到 DOM。
  - index.html
    - 主 HTML 文件，Vite 会以它为入口自动注入 JavaScript/CSS 资源。
    - 包含一个 id="app" 的 ，Vue 应用会挂载到此元素。
  - vite.config.js
    - Vite 的配置文件，可自定义构建行为（如代理、插件等）。
  - package.json
    - 包含项目依赖和脚本命令。



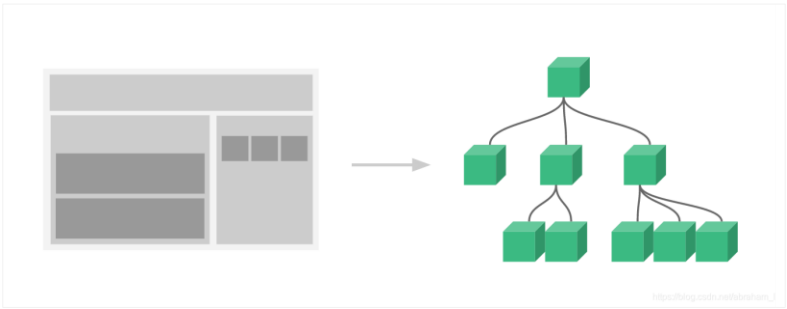
1. 组件 1.2 创建项目

- 项目准备
  - 将src目录下的assets子目录及其文件全部删除。
  - 将components目录及其所有子目录和文件全部删除。
  - 修改main.js入口文件，将assets目录下的样式文件引入代码删除。
  - 重写App.vue程序文件代码，分为script、template和style 3个部分。



1. 组件 1.3 组件注册

- 在 Vue 中组件注册是为了确保程序构建和渲染DOM时能够正确识别并应用组件。
- 注册组件时应遵循 W3C 规范中的PascalCase（首字母大写）自定义组件名，例如“GlobalComponent”，确保组件名具有描述性并且能够反映组件的功能或内容。
- 前端组件注册主要分为两种模式：全局注册和局部注册。



1. 组件 1.3 组件注册

- 全局注册
  - 全局注册可以确保在任何模板中都可以使用这个全局组件，在 Vue 中，全局注册一个组件通过“component()”方法实现。
  - 实现步骤：
    - 创建组件：同样先定义一个组件。
    - 全局注册组件：在项目入口文件（通常是 main.js）中，使用 app.component 方法进行全局注册。
    - 使用组件：在任意组件模板中直接使用已注册的组件。
  - 注意事项：
    - 命名规范：组件名称要遵循一定的命名规范，推荐使用短横线分隔命名法（如 my-component），这样可以避免与 HTML 原生标签冲突。
    - 性能影响：过多的全局注册组件可能会增加应用的初始加载时间和内存占用，因为所有全局注册的组件都会在应用启动时被加载。所以，对于一些不常用的组件，建议使用局部注册。
    - 可维护性：全局注册会让组件的使用范围变得模糊，不利于代码的维护和管理。在大型项目中，要谨慎使用全局注册，尽量遵循 “按需引入” 的原则。

1. 组件 1.3 组件注册

- 局部注册
  - 局部注册组件需要在使用它的父组件中引入或显式导入，注册后仅在当前组件可用。
  - 实现步骤：
    - 创建组件：同样先定义一个组件。
    - 局部注册组件：在使用组合式 API 时，在 <script setup> 中导入组件；使用选项式 API 时，在 components 选项中注册组件。
    - 使用组件：在组件的模板里使用已注册的组件。
  - 注意事项：
    - 命名规范：组件名称要遵循一定的命名规范，推荐使用短横线分隔命名法（如 child-component）或 PascalCase 命名法（如 ChildComponent），且要与注册时的名称保持一致。
    - 作用域：局部注册的组件只能在注册它的组件内部使用，在其他组件中无法直接使用。
    - 性能优势：局部注册有助于减少应用的初始加载时间和内存占用，因为只有在使用该组件时才会加载相关代码。所以，对于不常用的组件，建议使用局部注册。

1. 组件 1.4 组件样式

- 全局样式控制
  - 在组件中定义的样式，默认是全局有效的。
  - 也就是说，其可以作用于当前组件中的标签、子组件的根标签及外部的标签。
- 局部样式控制
  - 在style标签中添加scoped属性即可设置局部作用域样式。
    - 一旦style声明为scoped，当前组件的所有标签和子组件的根标签就都会自动添加名为data-v-xxx的唯一标识属性。
    - 在项目打包运行的页面中，style中的样式选择器的最右侧添加了名为data-v-xxx的属性选择器。这就让局部作用域样式只能作用于带data-v-xxx属性的标签，而此时只有当前组件的标签和子组件的根标签带有此属性，子组件的子标签和外部标签都没有此属性，因此局部作用域样式就只能影响当前组件的标签和子组件的根标签。

1. 组件 1.4 组件样式

- 深度样式控制
  - 将需要进行深度选择的标签用“：deep（）”来包含，它就能匹配并影响子组件的子标签。

```
<style scoped>
...
...
div h3 {
  font-size: 40px;
}
</style>
```

```
<style scoped>
...
...
div :deep(h3) {
  font-size: 40px;
}
</style>
```

```
div[data-v-7a7a37b1] h3 {
  font-size: 30px;
}
```

1. 组件 1.5 组件通信

- Vue 中的组件通信允许组件之间传递数据和事件，进而实现组件之间的状态同步。
- 在 Vue 中组件通信一般分为三类：父子组件通信、兄弟组件通信、跨层级组件通信。

1. 组件 1.5 组件通信

- 父子组件通信
  - 父组件向子组件传值
    - 父组件向子组件传递数据通过“props”属性实现，父组件在模板中使用“v-bind”将数据绑定到子组件的“props”上，子组件在“<script setup>”语法糖中通过“defineProps”来接收数据。

```vue
<template>
  <ChildComponent
    :title="parentTitle"
    :message="parentMessage"
    @childReceive="childReceive"
  />
  <div>{{ childData }}</div>
</template>

<script setup>
const parentTitle = "子组件标题";
const parentMessage = "你好子组件";
</script>
```

父组件

```vue
<template>
  <div>
    <h1>{{ title }}</h1>
    <p>{{ message }}</p>
  </div>
</template>

<script setup>
// 使用defineProps定义接收的props
const props = defineProps({
  title: String, // 假设title是一个字符串
  message: {
    // message可以是一个对象，更详细地定义prop
    type: String,
    required: true, // 标记为必需
    default: "Hello from default", // 默认值
  },
});
</script>
```

子组件

1. 组件 1.4 组件通信

- 父子组件通信
  - 子组件向父组件传值
    - 子组件向父组件传递数据通过自定义事件实现，父组件在模板中使用“v-on”（或简写为@）添加监听事件，子组件使用“defineEmits”方法触发事件，将值传递数据给父组件。

```vue
<template>
  <div>
    <h2>父组件</h2>
    <ChildComponent @send-data="handleData" />
    <p>接收数据: {{ receivedData }}</p>
  </div>
</template>

<script setup>
import { ref } from "vue";
import ChildComponent from "./ChildComponent.vue";

// 接收子组件传递的数据
const receivedData = ref(null);

// 处理子组件发送的数据
const handleData = (data) => {
  receivedData.value = data;
};
</script>
```

父组件

```vue
<template>
  <button @click="sendData">发送数据</button>
</template>

<script setup>
import { ref, defineEmits } from "vue";

// 定义可发射的事件
const emit = defineEmits(["send-data"]);

// 发送数据的方法
const sendData = () => {
  emit("send-data", "Hello from child!");
};
</script>
```

子组件

1. 组件 1.5 组件通信

- 兄弟组件通信
  - 兄弟组件之间不能直接通信，需要将数据通过“emit”发送给父组件，父组件再将数据发送给另一个兄弟组件，通过“props”接收。

```vue
<template>
  <div>
    <h2>父组件</h2>
    <ChildA @send-data="handleDataFromChildA" />
    <ChildB :dataFromChildA="receivedDataFromChildA" />
  </div>
</template>

<script setup>
import { ref } from "vue";
import ChildA from "./ChildComponent.vue";
import ChildB from "./ChildComponent2.vue";

// 接收来自 ChildA 的数据
const receivedDataFromChildA = ref(null);

// 处理 ChildA 发送的数据
const handleDataFromChildA = (data) => {
  receivedDataFromChildA.value = data;
};
</script>
```

父组件

```vue
<template>
  <button @click="sendData">发送数据</button>
</template>

<script setup>
import { ref, defineEmits } from "vue";

// 定义可发射的事件
const emit = defineEmits(["sen-data"]);

// 发送数据的方法
const sendData = () => {
  emit("send-data", "你好子组件A!");
};
</script>
```

子组件A

```vue
<template>
  <div>
    <h2>子组件B</h2>
    <p>来自子组件A: {{ dataFromChildA }}</p>
  </div>
</template>

<script setup>
// 定义 props
defineProps({
  dataFromChildA: {
    type: String,
    default: "",
  },
});
</script>
```

子组件B

1. 组件 1.5 组件通信

- 跨层级组件通信
  - Vue 提供了“provide”和“inject”选项来实现跨层级的组件通信。祖先组件可以使用“provide”选项来提供数据，后代组件则可以通过“inject”选项来接收数据。

```vue
<template>
  <div>
    <h2>父组件</h2>
    <Parent />
  </div>
</template>

<script setup>
import { ref, provide } from 'vue';
import Parent from './Parent.vue';

// 定义要传递的数据
const providedData = ref('来自父组件!');

// 使用 provide 提供数据
provide('sharedData', providedData);
</script>
```

父组件

```vue
<template>
  <div>
    <h2>子组件</h2>
    <Child />
  </div>
</template>

<script setup>
import Child from './Child.vue';
import { inject } from 'vue';

// 使用 inject 接收数据
const sharedData = inject('sharedData');
</script>
```

子组件

```vue
<template>
  <div>
    <h2>孙组件</h2>
    <p>来自父组件的数据: {{ sharedData }}</p>
  </div>
</template>

<script setup>
import { inject } from 'vue';

// 使用 inject 接收数据
const sharedData = inject('sharedData');
</script>
```

孙组件

2. 路由 2.1 路由简介

- 路由的基本思想都是关于数据的定向和转发。
- 前端路由是一种在单页应用（SPA）中管理页面导航的技术，是路径和对应的组件构成的一组映射关系。
  - 传统地址跳转是点击链接或在地址栏中输入 URL ，浏览器会向服务器发送请求，服务器会返回一个新的页面，这种方式会导致每次页面切换都需要重新加载整个页面。
  - 在 SPA 中，所有的页面内容都在一个 HTML 文件中加载，页面的切换是通过前端路由来实现的，点击链接或在地址栏中输入 URL ，前端路由会拦截这个请求，并根据 URL 地址通过配置的规则加载不同的页面内容
- 在Vue 应用启动时，会创建一个全局的路由实例（new VueRouter()），这个实例负责监听URL的变化，当URL发生变化时，路由实例会根据当前的URL去匹配之前定义的路由规则完成页面渲染。

2. 路由 2.1 路由简介

- 前端路由的实现方式主要有两种：Hash模式和History模式。
- Hash模式
  - Hash模式的前端路由通过在URL的hash部分（即#号后面的部分）来设置路由路径。当hash值改变时，浏览器不会向服务器发送请求，而是会触发“hashchange”事件，可以通过监听这个事件来捕获URL的变化，并更新页面的内容。
- History模式
  - History模式的前端路由则利用HTML5 History API来实现。这个API提供了“pushState”和“replaceState”两个方法，允许向浏览器的历史记录中添加或替换状态，同时不会触发页面的刷新。此外，当浏览器的历史记录发生变化时（如点击了前进或后退按钮），也会触发“popstate”事件。

2. 路由 2.2 Vue Router

- Vue Router是 Vue 的官方路由管理器，功能包括嵌套路由映射、动态路由选择、模块化、基于组件的路由配置、路由传参、路由查询、使用通配符、过渡效果、导航控制、自动激活 CSS 类的链接、HTML5 history 模式或 hash 模式、可定制的滚动行为、URL 的正确编码等。
- Vue Router 的特点
  - 声明式导航：通过 <RouterLink> 组件，创建 a 标签来定义导航链接。
  - 编程式导航：Vue Router 提供编程式导航的 API（如 router.push(), router.replace(), router.go() 等），允许在 JavaScript 代码中控制路由的跳转。
  - 嵌套路由：Vue Router 支持嵌套路由，允许将路由映射到嵌套的组件结构中。
  - 路由守卫：路由守卫允许在路由切换前、切换后或切换失败时执行特定的逻辑。
  - 路由参数：Vue Router 支持在 URL 地址中添加参数，并在组件中获取。
  - 路由懒加载：Vue Router 支持路由懒加载，即按需加载路由对应的组件。

2. 路由 2.3 创建项目

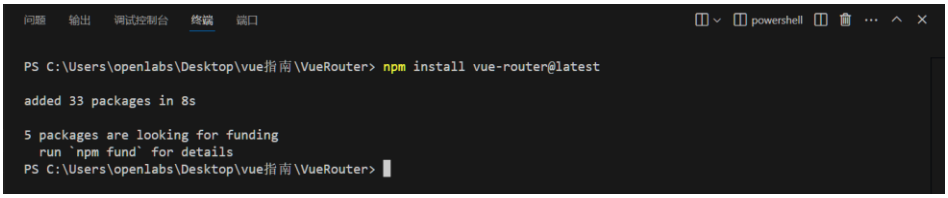
- 创建项目
  - 创建示例项目“vue-router”

2. 路由 2.4 路由应用

- 动态组件加载
  - 新建页面目录views，在该目录下新建Home.vue与Users.vue两个页面文件。
  - 在App.vue根组件文件中尝试引入Home.vue与Users.vue这两个页面文件，并在根组件文件中放置两个按钮。希望在点击按钮时，实现首页与用户页的切换操作。

2. 路由 2.4 路由应用

- 配置简单路由
  - 使用“npm install vue-router@latest”命令安装Vue Router。



2. 路由 2.4 路由应用

- 配置简单路由
  - 创建路由文件

```
// 导入 Vue Router 的 createRouter 和 createWebHashHistory 函数
import { createRouter, createWebHashHistory } from 'vue-router';

// 导入首页组件
import HomeView from '../views/HomeView.vue';

// 创建一个 Vue Router 实例
const router = createRouter({
    // 设置 history 模式为 hash 模式
    history: createWebHashHistory(import.meta.env.BASE_URL),

    // 定义路由规则
    routes: [
        // 首页路由配置
        {
            // 路由路径
            path: '/',
            // 路由名称
            name: 'home',
            // 对应的组件
            component: HomeView
        },

        // 关于页面路由配置
        {
            // 路由路径
            path: '/about',
            // 路由名称
            name: 'about',
            // 动态导入组件，实现懒加载
            component: () => import('../views/AboutView.vue')
        }
    ]
});

// 导出默认的 router 实例供其他模块使用
export default router;
```

2. 路由 2.4 路由应用

- 配置简单路由
  - 修改“src\main.js”文件，引入并全局注册路由组件

```
import './assets/main.css'

import { createApp } from 'vue'
import App from './App.vue'

// 导入路由实例，用于管理应用的路由
import router from './router';

const app = createApp(App)

// 使用 app.use() 方法注册路由到 Vue 应用中，在整个应用中使用路由功能
app.use(router);

app.mount('#app')
```


2. 路由 2.4 路由应用

- 配置简单路由
  - 创建导航组件文件，在“src”文件夹下创建“views”文件夹，在“views”文件夹下创建“HomeView.vue”文件和“AboutView.vue”文件。

```
<template>
  <div class="home">
    <h1>这是主页</h1>
  </div>
</template>

<style scoped>
.home {
  min-height: 100vh;
  display: flex;
  align-items: center;
}
</style>
```

```
<template>
  <div class="about">
    <h1>这是关于页</h1>
  </div>
</template>

<style scoped>
.about {
  min-height: 100vh;
  display: flex;
  align-items: center;
}
</style>
```

2. 路由 2.4 路由应用

- 配置简单路由
  - 修改“src\App.vue”文件，在“src\App.vue”文件中导入“RouterLink”和“RouterView”，文件内容如下。

```vue
<script setup>
// 从 'vue-router' 库导入 RouterLink 和 RouterView 组件
import { RouterLink, RouterView } from "vue-router";
</script>

<template>
  <!-- 头部区域 -->
  <header>
    <div class="wrapper">
      <!-- 导航栏 -->
      <nav>
        <!-- RouterLink 组件用于创建链接到不同路由的链接 -->
        <!-- 当前页面被激活时，会自动添加 active 类名 -->
        <RouterLink to="/">Home</RouterLink>
        <RouterLink to="/about">About</RouterLink>
      </nav>
    </div>
  </header>

  <!-- RouterView 组件用于渲染当前活动的路由视图 -->
  <RouterView />
</template>

<style scoped>
nav {
  width: 100vh;
  height: 50%;
  font-size: 24px;
  text-align: center;
  margin-top: 20rem;
}
nav a.router-link-exact-active {
  color: var(--color-text);
}

nav a.router-link-exact-active:hover {
  background-color: transparent;
}

nav a {
  display: inline-block;
  padding: 0 1rem;
  border-left: 1px solid var(--color-border);}

nav a:first-of-type {
  border: 0;
}
</style>
```


2. 路由 2.4 路由应用

- 嵌套路由：在 Vue 中，嵌套路由用于实现页面的分层级导航，适用于需要在父组件内部展示多个子组件的场景（如后台管理系统的多级菜单、带有侧边栏的主内容区域等）。
  - 嵌套路由可在一个父级路由组件中，通过 <router-view> 嵌入子路由组件的能力。例如：
    - 父路由路径为 /home，对应组件为 Home
    - 子路由路径为 /home/news 和 /home/message，对应组件为 HomeNews 和 HomeMessage
    - 此时，访问 /home/news 时，Home 组件和 HomeNews 组件会同时渲染，形成嵌套结构。

2. 路由 2.4 路由应用

- 命名路由
  - 在 Vue 中，命名路由为路由提供了一个名称，这使得路由的使用更加灵活和方便，尤其是在导航、传递参数和维护代码时。
  - 在路由配置时，通过 name 属性为每个路由规则赋予一个唯一的名称。

```javascript
const routes = [
  {
    path: '/user/:id',
    component: User,
    name: 'user'
  },
  {
    path: '/about',
    component: About,
    name: 'about'
  }
];
```


2. 路由 2.4 路由应用

- 命名路由
  - 使用场景
    - 简化路由引用：在代码里使用名称而非具体路径，可降低硬编码，提高代码的可维护性。若路由路径发生变更，只需在路由配置里修改，无需在所有使用该路由的地方修改。
    - 参数传递更清晰：在进行编程式导航时，使用命名路由传递参数更加直观，代码可读性更高。
    - 嵌套路由管理：在复杂的嵌套路由结构中，命名路由能更清晰地标识各个子路由，方便管理和导航。
  - 注意事项
    - 名称的唯一性：在路由配置中，每个路由的名称必须唯一，否则会引发路由冲突。
    - 参数传递：如果命名路由包含动态参数（如 :id），在导航时必须提供相应的参数，否则可能会导致路由匹配失败。
    - 路径变更：虽然使用命名路由可以减少硬编码，但如果路由的路径结构发生了重大变化，仍需要检查和更新相关的参数传递逻辑。

2. 路由 2.4 路由应用

- 命名视图
  - 在 Vue 中，命名视图是一种强大的路由特性，它允许在一个路由中同时渲染多个组件到不同的视图容器中，这为构建复杂的布局结构提供了便利。
  - 通常情况下，一个路由对应一个 <router-view> 组件进行渲染。而命名视图则允许为 <router-view> 组件添加 name 属性，给它们赋予不同的名称，从而可以在一个路由中同时渲染多个组件到不同名称的视图容器中。
  - 使用场景
    - 复杂布局页面：当页面需要同时展示多个不同功能区域，且这些区域的内容可以独立更新时，命名视图非常有用。例如，常见的布局有左侧侧边栏、中间主内容区和右侧辅助信息区，每个区域的内容可以由不同的组件来渲染。
    - 多面板展示：在一些管理系统或富媒体应用中，可能需要同时展示多个面板，如一个页面同时展示文章列表、文章详情和评论区，使用命名视图可以方便地实现这种布局。

2. 路由 2.4 路由应用

- 命名视图
  - 实现方式
    - 路由配置：使用 components（注意是复数形式）属性来定义多个组件，每个组件对应一个命名视图。

```javascript
const routes = [
  {
    path: '/',
    components: {
      default: MainContent, // 默认视图，即没有指定 name 的 <router-view>
      sidebar: Sidebar,
      aside: AsideInfo
    }
  }
];
```

    - 在模板中使用命名视图：在模板中，通过给 <router-view> 组件添加 name 属性来指定使用哪个命名视图。

```vue
<template>
  <div>
    <!-- 默认视图 -->
    <router-view></router-view>
    <!-- 侧边栏视图 -->
    <router-view name="sidebar"></router-view>
    <!-- 辅助信息视图 -->
    <router-view name="aside"></router-view>
  </div>
</template>
```

2. 路由 2.4 路由应用

- 命名视图
  - 注意事项
    - 名称唯一性：每个 <router-view> 的 name 属性必须唯一，否则会导致组件渲染混乱。
    - 组件加载顺序：命名视图的组件加载顺序与 <router-view> 在模板中的顺序一致，需要注意组件之间的依赖关系。
    - 路由守卫：命名视图中的每个组件都可以有自己的路由守卫，需要分别处理不同组件的导航逻辑。

2. 路由 2.4 路由应用

- 路由组件传参
  - 用于在路由路径中传递动态参数（如资源 ID、分类标识等），实现不同数据的页面展示（如 /user/1、product/2 等）
  - params参数的应用：在路由配置中，通过 参数占位符（以 : 开头）定义 params 参数，格式为 :参数名

```javascript
// 单个参数
{ path: '/user/:userId', component: UserDetail }
// 多个参数
{ path: '/blog/:category/:postId', component: BlogPost }
```

  - 占位符规则：参数名只能包含字母、数字、连字符（-）、下划线（_），且不能以 / 开头。
  - 路径匹配：参数值会被解析为字符串，包含在路由路径中（如 /user/123 中的 123 是 userId 的值）。

2. 路由 2.4 路由应用

- 路由组件传参
  - query参数的应用：query 参数无需在路由配置中预先声明，直接附加在 URL 中，格式为 ?key=value&key2=value2。
    - 访问路径：/search?keyword=vue&category=front-end
    - 对应的 query 参数：{ keyword: 'vue', category: 'front-end' }

2. 路由 2.4 路由应用

特性	query（查询参数）	params（动态参数）
路由配置	无需声明，直接附加在 URL 中	需在路由路径中定义占位符（:参数名）
必需性	可选（参数可存在或缺失）	必需（除非定义为可选参数）
参数位置	URL 中 ? 后的查询字符串（如 ?page=2）	路由路径的一部分（如 /user/123 中的 123）
应用场景	过滤条件、分页、临时参数（非唯一资源标识）	唯一资源标识（如用户 ID、商品 ID）
组件更新	参数变化时组件可能不重新加载（需手动监听）	参数变化时组件可能因缓存不重新加载（需手动处理）
参数格式	支持多个相同键（如 ?tag=vue&tag=js）	单个值（路径中参数唯一）

2. 路由 2.4 路由应用

- 编程式导航
  - 在 Vue 中，编程式导航是指通过 JavaScript 代码来控制路由的跳转，而不是使用 <router-link> 组件。这种方式在需要根据特定条件或用户交互动态改变路由时非常有用。
  - 在 Vue 里，编程式导航借助 $router 对象来达成。在选项式 API 里，this.$router 可直接运用；在组合式 API 中，得先引入 useRouter 函数来获取 router 实例。
  - 使用场景
    - 事件驱动导航：当用户点击按钮、提交表单或者完成某项操作后，需要根据操作结果进行路由跳转，例如登录成功后跳转到用户主页。
    - 条件判断导航：根据特定的条件（如用户权限、数据状态等）决定是否进行路由跳转以及跳转到哪个页面。
    - 异步操作后导航：在进行异步操作（如数据请求）完成后，根据返回结果进行路由跳转。

2. 路由 2.4 路由应用

- 编程式导航
  - 常见方法
    - router.push：此方法用于向历史记录栈添加一个新的记录，用户点击浏览器的后退按钮时，会回到之前的页面。

vue

```vue
<template>
  <button @click="goToHome">前往主页</button>
</template>

<script>
export default {
  methods: {
    goToHome() {
      this.$router.push('/home');
    }
  }
};
</script>
```

vue

```vue
<template>
  <button @click="goToHome">前往主页</button>
</template>

<script setup>
import { useRouter } from 'vue-router';

const router = useRouter();

const goToHome = () => {
  router.push('/home');
};
</script>
```

2. 路由 2.4 路由应用

- 编程式导航
  - 常见方法
    - router.go：router.go 方法用于在历史记录中向前或向后移动指定的步数，类似于浏览器的前进和后退按钮。参数为正数表示向前移动，负数表示向后移动。

```javascript
// 后退一步
router.go(-1);
// 前进一步
router.go(1);
```


2. 路由 2.4 路由应用

- 重定向
  - 重定向指的是当用户访问某个路径时，自动将其导向另一个路径。这在很多场景下都非常实用，比如网站重构后旧的 URL 需要指向新的 URL，或者根据用户的权限和状态引导到合适的页面。
  - 重定向可以在路由配置中通过 redirect 字段来实现，有多种配置方式。

```javascript
const routes = [
  {
    path: '/old-path',
    redirect: '/new-path'
  }
];
```

2. 路由 2.4 路由应用

- 重定向

```javascript
const routes = [
  {
    path: '/old-user',
    redirect: { name: 'new-user' }
  }
];
```

```javascript
const routes = [
  {
    path: '/user/:id',
    redirect: (to) => {
      // 根据条件进行重定向
      if (to.params.id === '1') {
        return '/admin';
      }
      return '/guest';
    }
  }
];
```


2. 路由 2.4 路由应用

- 别名
  - 别名是指为一个路由提供一个或多个其他路径，用户访问别名路径时，实际上访问的是原路由。别名可以提高路由的灵活性和可读性。

```javascript
const routes = [
  {
    path: '/main',
    component: MainComponent,
    alias: '/home'
  }
];
```

```javascript
const routes = [
  {
    path: '/main',
    component: MainComponent,
    alias: ['/home', '/index']
  }
];
```

2. 路由 2.4 路由应用

- 匹配当前路由的链接
  - 使用 <router-link> 的 active-class 和 exact-active-class 属性
    - <router-link> 是 Vue Router 提供的用于创建路由链接的组件，它有两个特殊属性 active-class 和 exact-active-class，分别用于设置当链接匹配当前路由时的类名以及精确匹配时的类名。

```vue
<template>
  <div>
    <!-- 设置普通匹配时的类名 -->
    <router-link to="/home" active-class="active-link">首页</router-link>
    <!-- 设置精确匹配时的类名 -->
    <router-link to="/about" exact-active-class="exact-active-link">关于</router-link>
  </div>
</template>

<style scoped>
.active-link {
  color: red;
}

.exact-active-link {
  text-decoration: underline;
}
</style>
```


2. 路由 2.4 路由应用

- 匹配当前路由的链接
  - 使用全局配置
    - 除了在单个 <router-link> 上设置类名，还可以在创建路由实例时进行全局配置。

```javascript
import { createRouter, createWebHistory } from 'vue-router';
import Home from './views/Home.vue';
import About from './views/About.vue';

const routes = [
  { path: '/home', component: Home },
  { path: '/about', component: About }
];

const router = createRouter({
  history: createWebHistory(),
  routes,
  // 全局配置匹配时的类名
  linkActiveClass: 'global-active-link',
  linkExactActiveClass: 'global-exact-active-link'
});

export default router;
```

```vue
<template>
  <div>
    <router-link to="/home">首页</router-link>
    <router-link to="/about">关于</router-link>
  </div>
</template>

<style scoped>
.global-active-link {
  font-weight: bold;
}

.global-exact-active-link {
  background-color: lightgray;
}
</style>
```

2. 路由 2.4 路由应用

- 路由守卫
  - 在 Vue Router 里，路由守卫是一项关键功能，它能让你在路由切换的各个阶段执行特定逻辑。这有助于实现权限控制、数据预加载、导航拦截等操作。
  - 全局守卫可作用于所有路由导航，有全局前置守卫、全局解析守卫和全局后置钩子三种。
    - 全局前置守卫：全局前置守卫在每次路由导航开始前触发，能用来做权限验证、数据预加载等。
    - 全局解析守卫：全局解析守卫在导航被确认之前，所有组件内守卫和异步路由组件被解析之后触发。
    - 全局后置钩子：全局后置钩子在导航完成后触发，它没有 next 函数，不能改变导航。

2. 路由 2.4 路由应用

- 路由守卫
  - 全局守卫
    - 全局前置守卫（beforeEach）：在每次路由导航开始前触发，可用于权限验证、数据预加载等操作。它接收三个参数：to（目标路由信息）、from（当前路由信息）和 next（用于控制导航流程的函数）。

```javascript
import { createRouter, createWebHistory } from 'vue-router';
import Home from './views/Home.vue';
import Dashboard from './views/Dashboard.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/dashboard', component: Dashboard }
];

const router = createRouter({
  history: createWebHistory(),
  routes
});

// 全局前置守卫
router.beforeEach((to, from, next) => {
  // to 表示要去的路由信息
  // from 表示当前所在的路由信息
  // next 是一个函数，用于控制导航流程
  const isAuthenticated = localStorage.getItem('isAuthenticated');
  if (to.path === '/dashboard' && !isAuthenticated) {
    next('/'); // 未登录，重定向到首页
  } else {
    next(); // 允许导航
  }
});

export default router;
```

2. 路由 2.4 路由应用

- 路由守卫
  - 全局守卫
    - 全局解析守卫（beforeResolve）：在导航被确认之前，所有组件内守卫和异步路由组件被解析之后触发。通常用于确保所有异步操作完成后再进行导航。

```javascript
router.beforeResolve((to, from, next) => {
  // 可以在这里进行一些最后的验证或数据处理
  next();
});
```


2. 路由 2.4 路由应用

- 路由守卫
  - 全局守卫
    - 全局后置钩子（afterEach）：在导航完成后触发，它没有 next 函数，不能改变导航行为，一般用于记录日志等操作。

```javascript
router.afterEach((to, from) => {
  // 可以在这里进行一些日志记录等操作
  console.log(`从 ${from.path} 导航到 ${to.path}`);
});
```

2. 路由 2.4 路由应用

- 路由守卫
  - 路由独享守卫：路由独享守卫是针对某个特定路由设置的守卫，在进入该路由时触发。通过在路由配置中添加 beforeEnter 选项来定义。

```javascript
const routes = [
  {
    path: '/admin',
    component: Admin,
    beforeEnter: (to, from, next) => {
      const isAdmin = localStorage.getItem('isAdmin');
      if (!isAdmin) {
        next('/'); // 不是管理员，重定向到首页
      } else {
        next(); // 允许访问
      }
    }
  }
];
```


2. 路由 2.4 路由应用

- 路由守卫
  - 组件内守卫：组件内守卫是在组件内部定义的守卫，有 beforeRouteEnter、beforeRouteUpdate 和 beforeRouteLeave 三种。
  - beforeRouteEnter：在进入路由前触发，此时组件实例还未创建，不能直接访问 this。可以通过 next 函数的回调来访问组件实例。

```vue
<template>
  <div>
    <h1>用户详情</h1>
  </div>
</template>

<script>
export default {
  beforeRouteEnter(to, from, next) {
    // 可以在进入路由前获取数据
    next(vm => {
      // 通过回调函数访问组件实例 vm
      // 这里可以进行一些初始化操作
    });
  }
};
</script>
```

2. 路由 2.4 路由应用

- 路由守卫
  - beforeRouteUpdate：在当前路由改变，但是该组件被复用时调用，比如动态路由参数变化时。可以访问 this。

```vue
<template>
  <div>
    <h1>用户详情</h1>
  </div>
</template>

<script>
export default {
  beforeRouteUpdate(to, from, next) {
    // 可以根据新的路由参数更新数据
    next();
  }
};
</script>
```


2. 路由 2.4 路由应用

- 路由守卫
  - beforeRouteLeave：在离开当前路由时触发，可用于提示用户保存未保存的数据等。可以访问 this。

```vue
<template>
  <div>
    <h1>编辑页面</h1>
  </div>
</template>

<script>
export default {
  beforeRouteLeave(to, from, next) {
    const hasUnsavedChanges = this.hasUnsavedChanges;
    if (hasUnsavedChanges) {
      if (confirm('有未保存的更改，确定要离开吗？')) {
        next(); // 允许离开
      } else {
        next(false); // 阻止离开
      }
    } else {
      next(); // 无未保存更改，允许离开
    }
  }
};
</script>
```

2. 路由 2.4 路由应用

- 路由守卫的执行顺序
  1. 导航被触发。
  2. 在失活的组件里调用 beforeRouteLeave 守卫。
  3. 调用全局的 beforeEach 守卫。
  4. 在重用的组件里调用 beforeRouteUpdate 守卫。
  5. 在路由配置里调用 beforeEnter。
  6. 解析异步路由组件。
  7. 在被激活的组件里调用 beforeRouteEnter。
  8. 调用全局的 beforeResolve 守卫。
  9. 导航被确认。
  10. 调用全局的 afterEach 钩子。
  11. 触发 DOM 更新。
  12. 用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数。

2. 路由 2.4 路由应用

- 应用场景
  - 权限管理：通过全局前置守卫验证用户的登录状态和权限，控制用户对不同页面的访问。
  - 数据预加载：在进入路由前，使用路由守卫获取所需的数据，避免页面加载后出现空白。
  - 表单验证：在用户离开表单页面时，使用 beforeRouteLeave 守卫检查表单是否有未保存的数据，提示用户保存。
  - SEO 优化：在路由切换时，使用全局后置钩子更新页面的标题和元信息，提高搜索引擎收录。

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