

vue-router详解

王红元

微博: coderwhy

微信: 372623326



实力IT教育 www.520it.com



内容概述

- 认识路由
- vue-router基本使用
- vue-router嵌套路由
- vue-router参数传递
- vue-router导航守卫
- keep-alive

什么是路由?

■ 说起路由你想起了什么?

□ 路由是一个网络工程里面的术语。

□ **路由 (routing)** 就是通过互联的网络把信息从源地址传输到目的地址的活动. --- 维基百科

■ 额, 啥玩意? 没听懂

□ 在生活中, 我们有没有听说过路由的概念呢? 当然了, 路由器嘛.

□ 路由器是做什么的? 你有想过吗?

□ 路由器提供了两种机制: 路由和转送.

✓ 路由是决定数据包从**来源到目的地**的路径.

✓ 转送将**输入端**的数据转移到合适的**输出端**.

□ 路由中有一个非常重要的概念叫路由表.

✓ 路由表本质上就是一个映射表, 决定了数据包的指向.





后端路由阶段

■ 早期的网站开发整个HTML页面是由服务器来渲染的.

- 服务器直接生产渲染好对应的HTML页面, 返回给客户端进行展示.

■ 但是, 一个网站, 这么多页面服务器如何处理呢?

- 一个页面有自己对应的网址, 也就是URL.
- URL会发送到服务器, 服务器会通过正则对该URL进行匹配, 并且最后交给一个Controller进行处理.
- Controller进行各种处理, 最终生成HTML或者数据, 返回给前端.
- 这就完成了一个IO操作.

■ 上面的这种操作, 就是后端路由.

- 当我们页面中需要请求不同的**路径**内容时, 交给服务器来进行处理, 服务器渲染好整个页面, 并且将页面返回给客户端.
- 这种情况下渲染好的页面, 不需要单独加载任何的js和css, 可以直接交给浏览器展示, 这样也有利于SEO的优化.

■ 后端路由的缺点:

- 一种情况是整个页面的模块由后端人员来编写和维护的.
- 另一种情况是前端开发人员如果要开发页面, 需要通过PHP和Java等语言来编写页面代码.
- 而且通常情况下HTML代码和数据以及对应的逻辑会混在一起, 编写和维护都是非常糟糕的事情.

前端路由阶段

■ 前后端分离阶段：

- 随着Ajax的出现, 有了前后端分离的开发模式.
- 后端只提供API来返回数据, 前端通过Ajax获取数据, 并且可以通过JavaScript将数据渲染到页面中.
- 这样做最大的优点就是前后端责任的清晰, 后端专注于数据上, 前端专注于交互和可视化上.
- 并且当移动端(iOS/Android)出现后, 后端不需要进行任何处理, 依然使用之前的一套API即可.
- 目前很多的网站依然采用这种模式开发.

■ 单页面富应用阶段：

- 其实SPA最主要的特点就是在前后端分离的基础上加了一层前端路由.
- 也就是前端来维护一套路由规则.

■ 前端路由的核心是什么呢?

- 改变URL, 但是页面不进行整体的刷新。
- 如何实现呢?



URL的hash

■ URL的hash

- URL的hash也就是锚点(#), 本质上是改变window.location的href属性.
- 我们可以通过直接赋值location.hash来改变href, 但是页面不发生刷新

```
> location.href
< "http://192.168.1.101:8000/examples/urlChange/"
> location.hash = '/'
< "/"
> location.href
< "http://192.168.1.101:8000/examples/urlChange/#/"
> location.hash = '/foo'                                made by coderwhy
< "/foo"                                              微博: coderwhy
> location.href
< "http://192.168.1.101:8000/examples/urlChange/#/foo"
> |
```



HTML5的history模式: pushState

- history接口是HTML5新增的, 它有五种模式改变URL而不刷新页面.
- history.pushState()

```
> location.href  
< "http://192.168.1.101:8000/examples/urlChange/"  
|  
> history.pushState({}, "", "/foo")  
< undefined  
|  
|  
> location.href  
< "http://192.168.1.101:8000/foo"  
|  
> history.pushState({}, "", "/")  
< undefined  
|  
|  
> location.href  
< "http://192.168.1.101:8000/"  
|  
|  
>
```

made by coderwhy
微博: coderwhy



HTML5的history模式：replaceState

■ history.replaceState()

```
> location.href
< "http://192.168.1.101:8000/"
> history.replaceState({}, '', '/foo')
< undefined
> location.href
< "http://192.168.1.101:8000/foo"
> history.replaceState({}, '', '/foo/bar')
< undefined
> location.href
< "http://192.168.1.101:8000/foo/bar"
>
```

made by coderwhy
微博: coderwhy



HTML5的history模式：go

■ history.go()

```
> location.href  
< "http://192.168.1.101:8000/"  
  
> history.go(-1)  
< undefined  
  
> location.href  
< "http://192.168.1.101:8000/foo"  
  
> history.go(-1)  
< undefined  
  
> location.href  
< "http://192.168.1.101:8000/examples/urlChange/"  
  
> history.go(1)          made by coderwhy  
< undefined            微博: coderwhy  
  
> location.href  
< "http://192.168.1.101:8000/foo"  
  
> |
```

■ 补充说明：

- 上面只演示了三个方法
- 因为 history.back() 等价于 history.go(-1)
- history.forward() 则等价于 history.go(1)
- 这三个接口等同于浏览器界面的前进后退。



认识vue-router

■ 目前前端流行的三大框架, 都有自己的路由实现:

- Angular的ngRouter
- React的ReactRouter
- Vue的vue-router

■ 当然, 我们的重点是vue-router

- vue-router是Vue.js官方的路由插件, 它和Vue.js是深度集成的, 适合用于构建单页面应用。
- 我们可以访问其官方网站对其进行学习: <https://router.vuejs.org/zh/>

■ vue-router是基于路由和组件的

- 路由用于设定访问路径, 将路径和组件映射起来。
- 在vue-router的单页面应用中, 页面的路径的改变就是组件的切换。



安装和使用vue-router

■ 因为我们已经学习了webpack, 后续开发中我们主要是通过工程化的方式进行开发的.

□ 所以在后续, 我们直接使用npm来安装路由即可.

□ 步骤一: 安装vue-router

➤ npm install vue-router --save

□ 步骤二: 在模块化工程中使用它(因为是一个插件, 所以可以通过Vue.use()来安装路由功能)

➤ 第一步: 导入路由对象, 并且调用 Vue.use(VueRouter)

➤ 第二步: 创建路由实例, 并且传入路由映射配置

➤ 第三步: 在Vue实例中挂载创建的路由实例

```
import Vue from 'vue'  
import VueRouter from 'vue-router'
```

```
Vue.use(VueRouter)
```

■ 使用vue-router的步骤:

□ 第一步: 创建路由组件

□ 第二步: 配置路由映射: 组件和路径映射关系

□ 第三步: 使用路由: 通过<router-link>和<router-view>



创建router实例

The screenshot shows a code editor interface with a dark theme. On the left is a project structure sidebar. A red box highlights the 'router' folder under 'src'. A red arrow points from this box to the 'index.js' file in the main editor area. The 'index.js' file contains the following code:

```
import Vue from 'vue'
import VueRouter from 'vue-router'

// 1.注入插件
Vue.use(VueRouter)

// 2.定义路由
const routes = []

// 3.创建router实例
const router = new VueRouter({
  routes
})

// 4.导出router实例
export default router
```

Below the code, there is a watermark: **made by coderwhy** and **微博: coderwhy**.

File Edit View Navigate Code Refactor Run Tools VCS Window Help

VueJS课程 09-vue-router基本使用 src router index.js

Project

1: Project

- 06-表单绑定
- 07-组件化基础
- 08-组件化高级
- 09-vue-router基本使用
- build
- config
- node_modules library root
- src
 - assets
 - components
 - router
 - index.js
 - App.vue
 - main.js
- static
 - .babelrc
 - .editorconfig
 - .gitignore
 - .postcssrcjs
 - index.html
 - package.json
 - package-lock.json
 - README.md
- js
- node_modules
- template
 - package-lock.json
 - template.zip
- External Libraries
- Scratches and Consoles

Favorites

Terminal

Event Log

19:1 LF: UTF-8

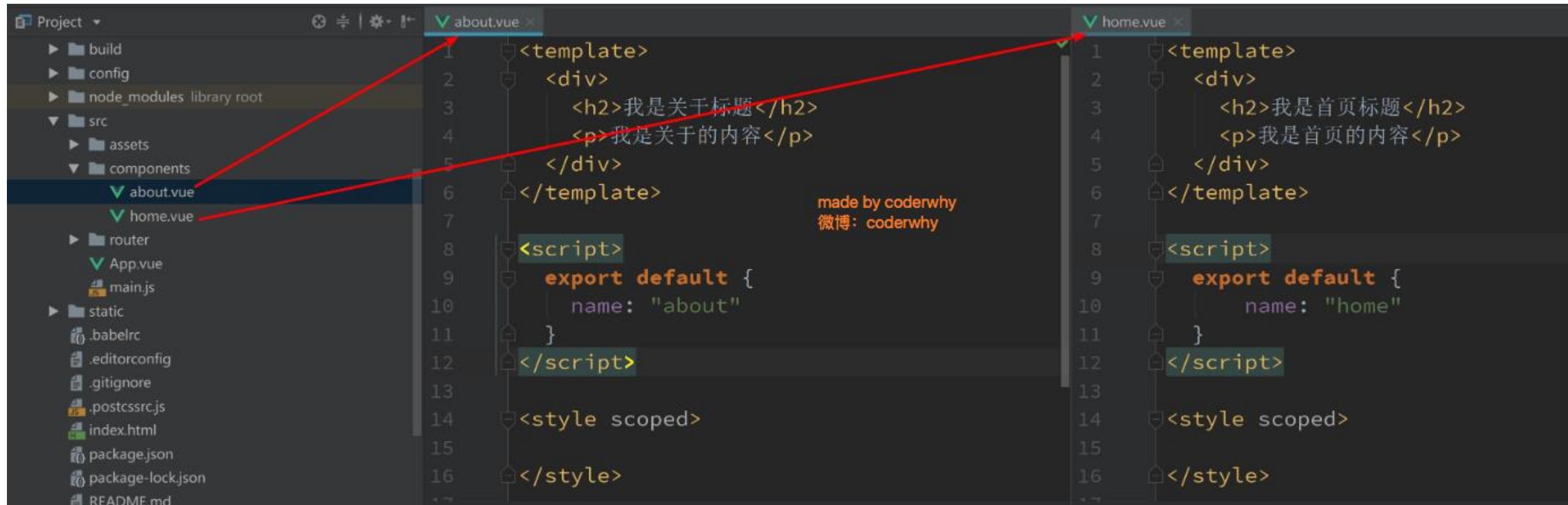


挂载到Vue实例中

```
Project index.js x main.js x
1 import Vue from 'vue'
2 import App from './App'
3 import router from './router'
4
5 new Vue({
6   el: '#app',
7   router,
8   render: h => h(App)
9 })
10
```

made by coderwhy
微博: coderwhy

步骤一：创建路由组件



The screenshot shows a Vue.js project structure in a code editor. The project tree on the left includes build, config, node_modules, library root, src (with assets and components), router (App.vue, main.js), static, .babelrc, .editorconfig, .gitignore, .postcssrc.js, index.html, package.json, package-lock.json, and README.md. Two components are open in tabs: 'about.vue' and 'home.vue'. A red arrow points from the 'components' folder in the project tree to the 'about.vue' component. Another red arrow points from the 'components' folder to the 'home.vue' component.

```
Project
  build
  config
  node_modules library root
  src
    assets
    components
      about.vue
      home.vue
    router
      App.vue
      main.js
  static
    .babelrc
    .editorconfig
    .gitignore
    .postcssrc.js
    index.html
  package.json
  package-lock.json
  README.md
```

about.vue

```
<template>
  <div>
    <h2>我是关于标题</h2>
    <p>我是关于的内容</p>
  </div>
</template>

<script>
  export default {
    name: "about"
  }
</script>

<style scoped>
</style>
```

**made by coderwhy
微博: coderwhy**

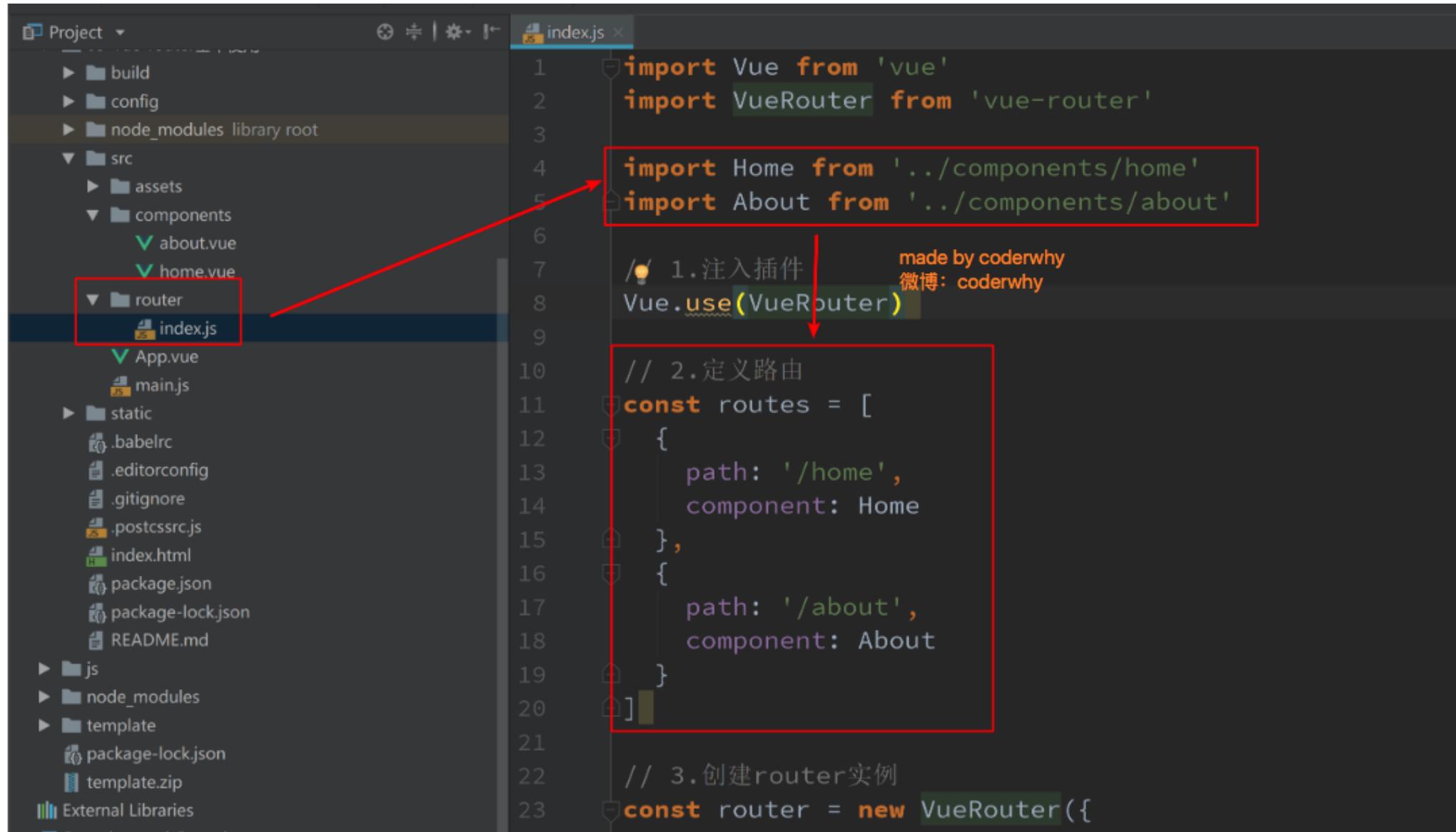
home.vue

```
<template>
  <div>
    <h2>我是首页标题</h2>
    <p>我是首页的内容</p>
  </div>
</template>

<script>
  export default {
    name: "home"
  }
</script>

<style scoped>
</style>
```

步骤二：配置组件和路径的映射关系



The screenshot shows a code editor with a file tree on the left and a code editor window on the right.

File Tree:

- Project
- build
- config
- node_modules library root
- src
 - assets
 - components
 - about.vue
 - home.vue
 - router
 - index.js
 - App.vue
 - main.js
- static
 - .babelrc
 - .editorconfig
 - .gitignore
 - .postcssrc.js
 - index.html
 - package.json
 - package-lock.json
 - README.md
- js
- node_modules
- template
 - package-lock.json
 - template.zip
- External Libraries

Code Editor (index.js):

```
import Vue from 'vue'
import VueRouter from 'vue-router'

import Home from '../components/home'
import About from '../components/about'

Vue.use(VueRouter)

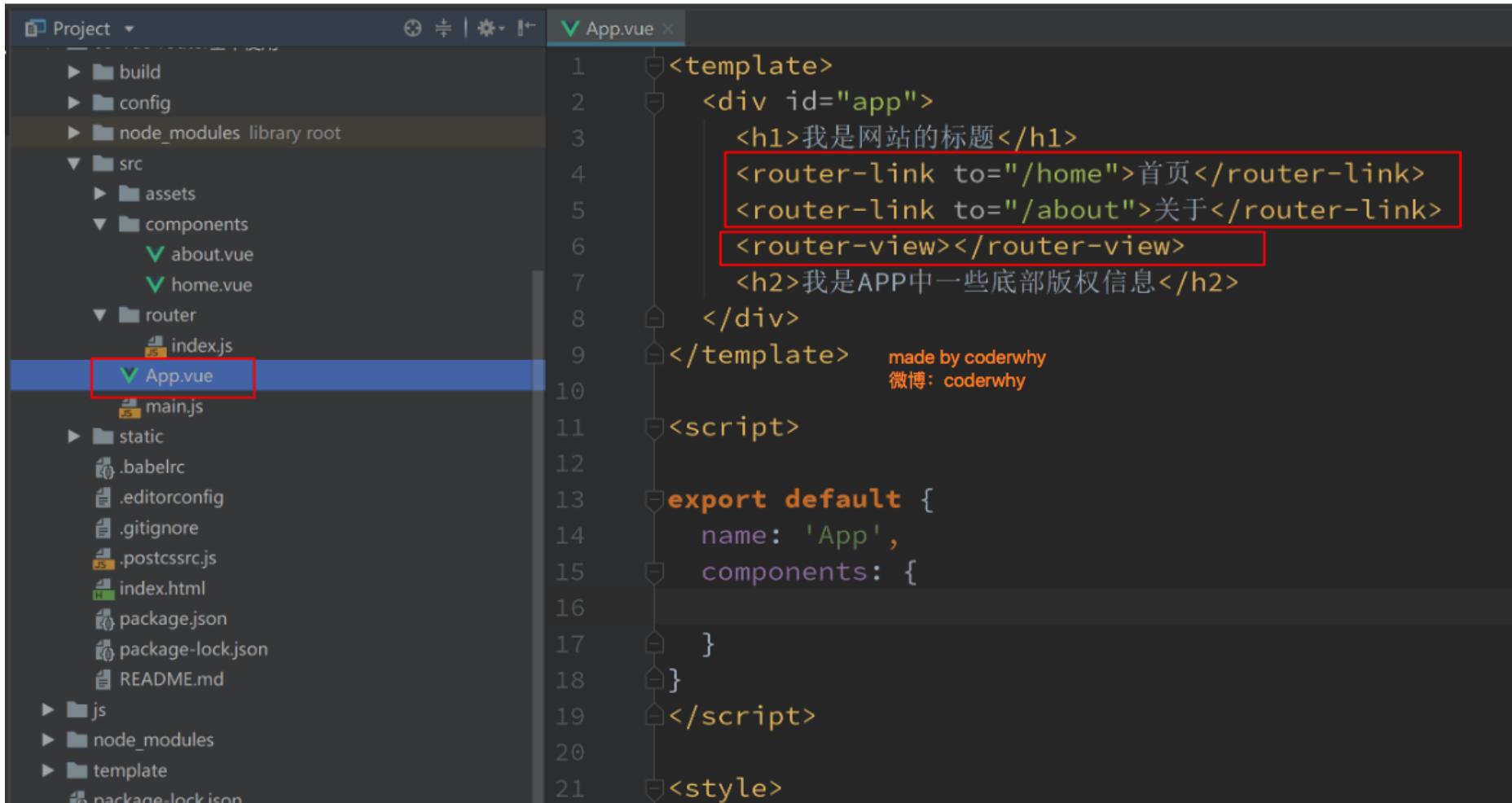
const routes = [
  {
    path: '/home',
    component: Home
  },
  {
    path: '/about',
    component: About
  }
]

const router = new VueRouter({
```

The code is annotated with red boxes and arrows:

- A red box highlights the imports of Home and About components.
- A red box highlights the `Vue.use(VueRouter)` call, with a callout "1.注入插件 made by coderwhy 微博: coderwhy". An arrow points from this callout to the `Vue.use` line.
- A red box highlights the definition of the `routes` array, with a callout "2. 定义路由". An arrow points from this callout to the first route definition.
- A red box highlights the creation of the `router` instance at the bottom of the file.

步骤三：使用路由.



The screenshot shows a code editor with a Vue.js project structure. The left sidebar lists files and folders: build, config, node_modules (library root), src (assets, components, router), and static. The router folder contains index.js and App.vue. The main area shows the content of App.vue:

```
<template>
  <div id="app">
    <h1>我是网站的标题</h1>
    <router-link to="/home">首页</router-link>
    <router-link to="/about">关于</router-link>
    <router-view></router-view>
    <h2>我是APP中一些底部版权信息</h2>
  </div>
</template> made by coderwhy
微博: coderwhy

<script>

  export default {
    name: 'App',
    components: {

    }
  }
</script>

<style>
```

The code editor highlights the `<router-link>` and `<router-view>` components with red boxes.

- `<router-link>`: 该标签是一个 vue-router 中已经内置的组件, 它会被渲染成一个`<a>`标签.
- `<router-view>`: 该标签会根据当前的路径, 动态渲染出不同的组件.
 - 网页的其他内容, 比如顶部的标题/导航, 或者底部的一些版权信息等会和`<router-view>`处于同一个等级.
 - 在路由切换时, 切换的是`<router-view>`挂载的组件, 其他内容不会发生改变.

最终效果如下





路由的默认路径

■ 我们这里还有一个不太好的实现:

- 默认情况下, 进入网站的首页, 我们希望<router-view>渲染首页的内容.
- 但是我们的实现中, 默认没有显示首页组件, 必须让用户点击才可以.

■ 如何可以让路径默认跳到到首页, 并且<router-view>渲染首页组件呢?

- 非常简单, 我们只需要配置多配置一个映射就可以了.

```
const routes = [  
  {  
    path: '/',  
    redirect: '/home'  
  },
```

■ 配置解析:

- 我们在routes中又配置了一个映射.
- path配置的是根路径: /
- redirect是重定向, 也就是我们将根路径重定向到/home的路径下, 这样就可以得到我们想要的结果了.



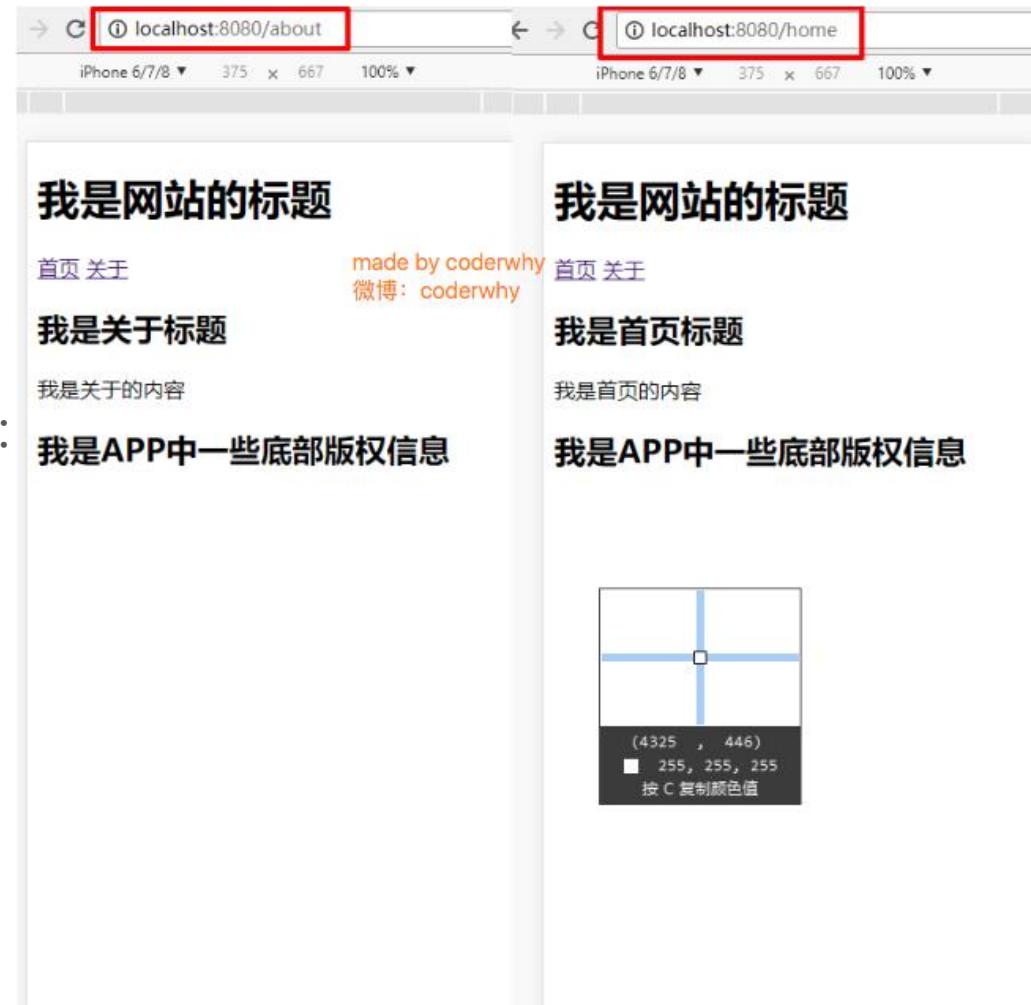
HTML5的History模式

■ 我们前面说过改变路径的方式有两种:

- URL的hash
- HTML5的history
- 默认情况下, 路径的改变使用的URL的hash.

■ 如果希望使用HTML5的history模式, 非常简单, 进行如下配置即可:

```
// 3. 创建router实例
const router = new VueRouter({
  routes,
  mode: 'history'
})
```





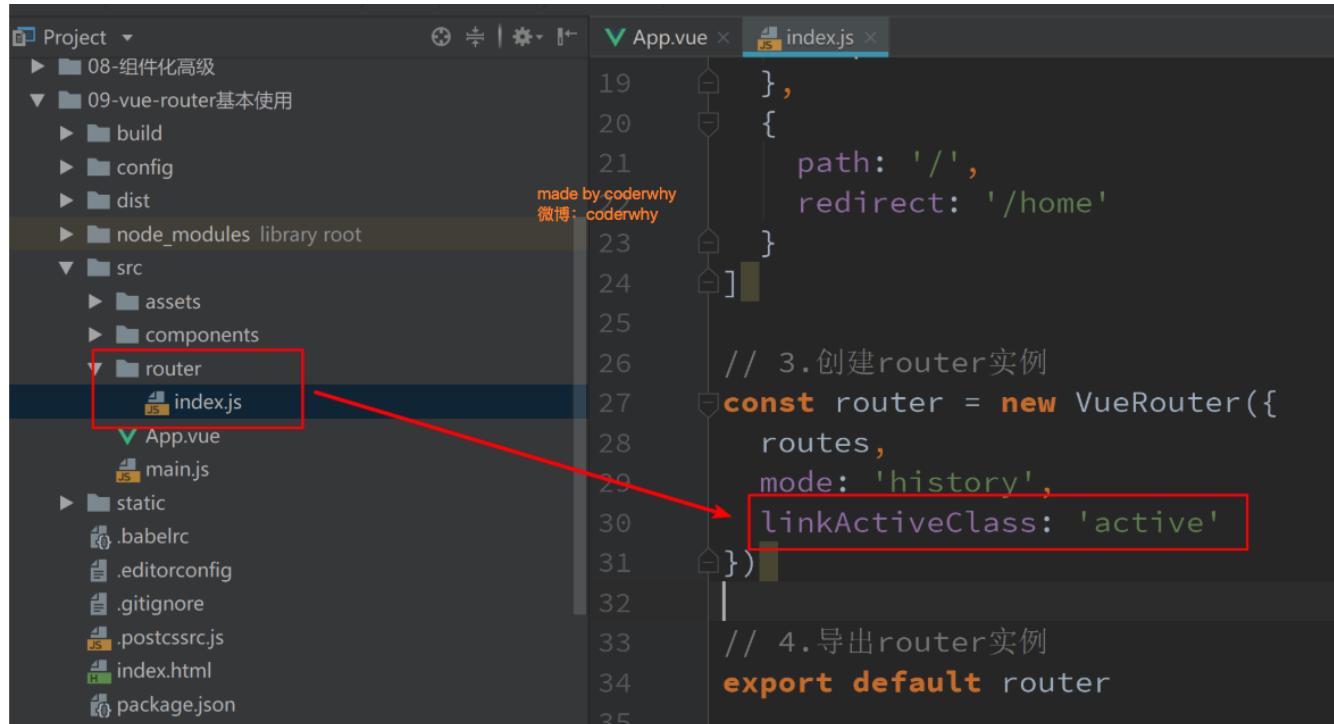
router-link补充

- 在前面的<router-link>中, 我们只是使用了一个**属性: to**, 用于指定跳转的路径.
- <router-link>还有一些**其他属性**:
 - **tag**: tag可以指定<router-link>之后渲染成什么组件, 比如上面的代码会被渲染成一个元素, 而不是<a>
 - **replace**: replace不会留下history记录, 所以指定replace的情况下, 后退键返回不能返回到上一个页面中
 - **active-class**: 当<router-link>对应的路由匹配成功时, 会自动给当前元素设置一个router-link-active的class, 设置active-class可以修改默认的名称.
 - 在进行高亮显示的导航菜单或者底部tabbar时, 会使用到该类.
 - 但是通常不会修改类的属性, 会直接使用默认的router-link-active即可.

```
~~~~~  
<div id="app">  
  <h1>我是网站的标题</h1>  
  <a href="/home" class="router-link-exact-active router-link-active">首页</a>  
  <a href="/about" class>关于</a> == $0  
  > <div data-v-8dc7cce2>...</div>
```

修改linkActiveClass

- 该class具体的名称也可以通过router实例的属性进行修改



The screenshot shows a code editor with a sidebar containing project files. A red box highlights the 'router' folder, and another red box highlights the 'linkActiveClass: 'active'' line in the 'index.js' file under 'router'. The code in 'index.js' is as follows:

```
19 },
20 {
21   path: '/',
22   redirect: '/home'
23 }
24 ]
25
26 // 3. 创建router实例
27 const router = new VueRouter({
28   routes,
29   mode: 'history',
30   linkActiveClass: 'active'
31 })
32
33 // 4. 导出router实例
34 export default router
```

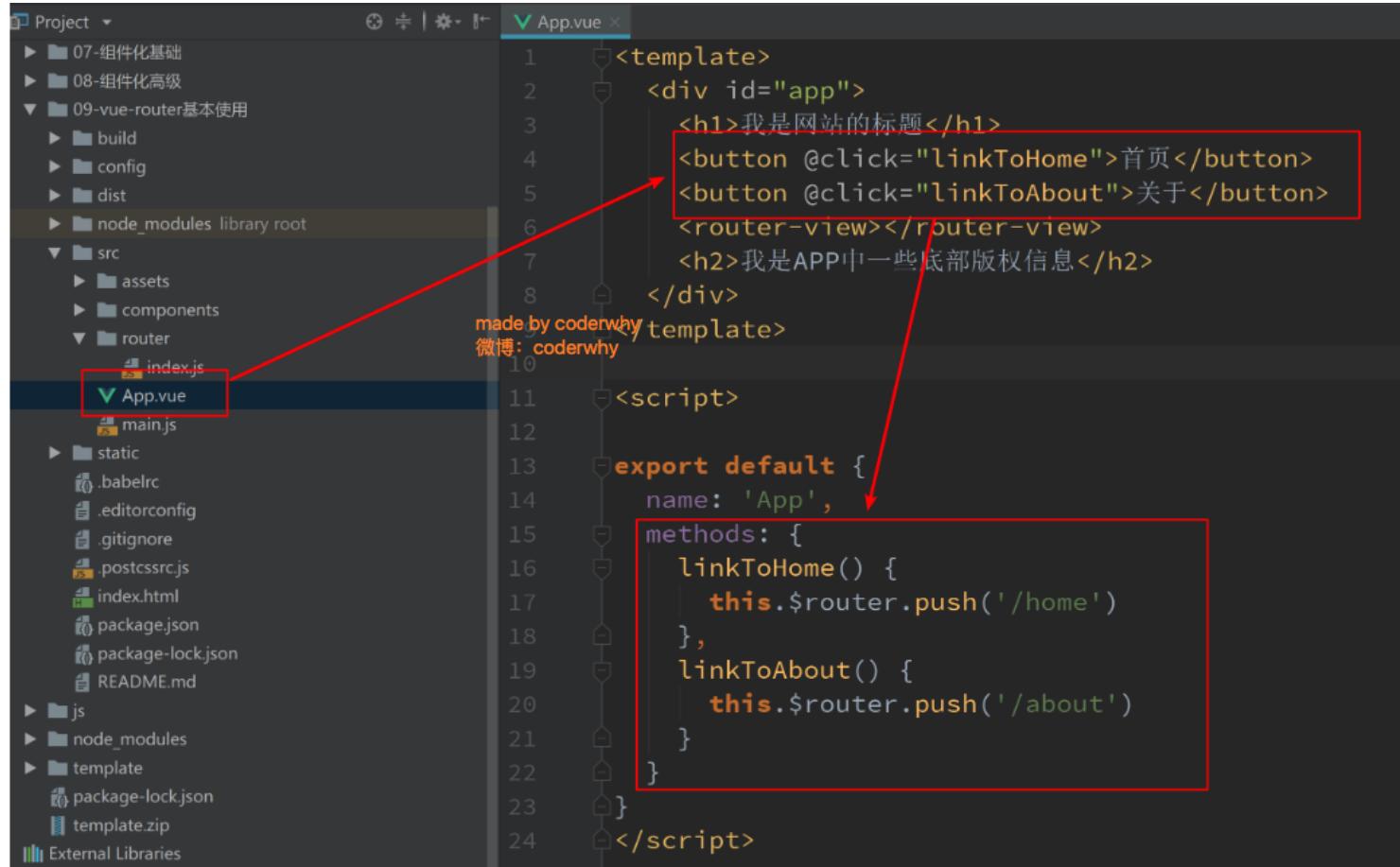
```
<div id="app">
  <h1>我是网站的标题</h1>
  <a href="/home" class="router-link-exact-active active">首页</a> == $0
  <a href="/about" class>关于</a>
  <div data-v-8dc7cce2>...</div>
  <h2>我是APP中一些底部版权信息</h2>
</div>
```

■ exact-active-class

- 类似于active-class, 只是在精准匹配下才会出现的class.
- 后面看到嵌套路由时, 我们再看下这个属性.

路由代码跳转

- 有时候, 页面的跳转可能需要执行对应的JavaScript代码, 这个时候, 就可以使用第二种跳转方式了
- 比如, 我们将代码修改如下:



The screenshot shows a Vue.js project structure in a code editor. The project root contains several folders like '07-组件化基础', '08-组件化高级', and '09-vue-router基本使用'. In the 'src' folder, there's an 'index.js' file and a 'router' folder containing 'index.js' and 'App.vue'. The 'App.vue' file is currently selected. The code in 'App.vue' is as follows:

```
<template>
  <div id="app">
    <h1>我是网站的标题</h1>
    <button @click="linkToHome">首页</button>
    <button @click="linkToAbout">关于</button>
    <router-view></router-view>
    <h2>我是APP中一些底部版权信息</h2>
  </div>
<script>
  export default {
    name: 'App',
    methods: {
      linkToHome() {
        this.$router.push('/home')
      },
      linkToAbout() {
        this.$router.push('/about')
      }
    }
  }
</script>
```

Red boxes highlight the button click events and the corresponding router push logic in the 'methods' section of the script.

动态路由

■ 在某些情况下，一个页面的path路径可能是不确定的，比如我们进入用户界面时，希望是如下的路径：

- /user/aaaa或/user/bbbb
- 除了有前面的/user之外，后面还跟上了用户的ID
- 这种path和Component的匹配关系，我们称之为动态路由(也是路由传递数据的一种方式)。

```
{  
  path: '/user/:id',  
  component: User  
}
```

```
<div>  
  <h2>{{$route.params.id}}</h2>  
</div>
```

```
<router-link to="/user/123">用户</router-link>
```

← → ⌂ ⓘ localhost:8080/user/123

我是APP页面

[首页](#)[关于用户](#)

123



认识路由的懒加载

■ 官方给出了解释:

- 当打包构建应用时，Javascript 包会变得非常大，影响页面加载。
- 如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了

■ 官方在说什么呢？

- 首先，我们知道路由中通常会定义很多不同的页面。
- 这个页面最后被打包在哪里呢？一般情况下，是放在一个js文件中。
- 但是，页面这么多放在一个js文件中，必然会造成这个页面非常的大。
- 如果我们一次性从服务器请求下来这个页面，可能需要花费一定的时间，甚至用户的电脑上还出现了短暂空白的情况。
- 如何避免这种情况呢？使用路由懒加载就可以了。

■ 路由懒加载做了什么？

- 路由懒加载的主要作用就是将路由对应的组件打包成一个个的js代码块。
- 只有在这个路由被访问到的时候，才加载对应的组件

路由懒加载的效果

```
import Home from './components/Home'
import About from './components/About'

Vue.use(VueRouter);

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

```

引用后直接使用



- static
- css
- js
 - app.801c2823389fbf98a530.js
 - manifest.2ae2e69a05c33dfc65f8.js
 - vendor.1748317793fd05195ff8.js

index.html

```
const routes = [
  {
    path: '/home',
    component: () => import('./components/Home')
  },
  {
    path: '/about',
    component: () => import('./components/About')
  },
];

▼ static
  ▶ css
  ▶ js
    ▶ 0.09675c5e37c95c6e65ff.js
    ▶ 1.266ad04847546fd5bdb2.js
    ▶ app.513c0ad5da30a20ee757.js
    ▶ manifest.f2307a2fbea088ed5ed4.js
    ▶ vendor.426ef21560bb1458790e.js
  index.html
```



懒加载的方式

- 方式一: 结合Vue的异步组件和Webpack的代码分析.

```
const Home = resolve => { require.ensure(['../components/Home.vue'], () =>
{ resolve(require('../components/Home.vue')) })};
```

- 方式二: AMD写法

```
const About = resolve => require(['../components/About.vue'], resolve);
```

- 方式三: 在ES6中, 我们可以有更加简单的写法来组织Vue异步组件和Webpack的代码分割.

```
const Home = () => import('../components/Home.vue')
```

认识嵌套路由

■ 嵌套路由是一个很常见的功能

- 比如在home页面中, 我们希望通过/home/news和/home/message访问一些内容.
- 一个路径映射一个组件, 访问这两个路径也会分别渲染两个组件.

■ 路径和组件的关系如下:

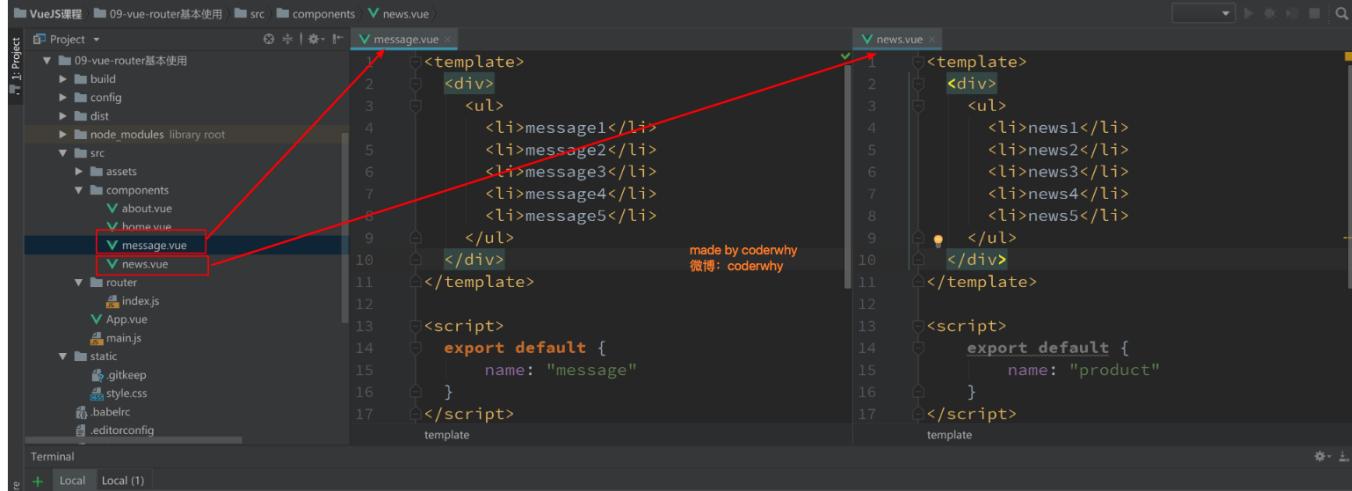


■ 实现嵌套路由有两个步骤:

- 创建对应的子组件, 并且在路由映射中配置对应的子路由.
- 在组件内部使用<router-view>标签.

嵌套路由实现

定义两个组件:



```

<template>
  <div>
    <ul>
      <li>message1</li>
      <li>message2</li>
      <li>message3</li>
      <li>message4</li>
      <li>message5</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: "message"
}
</script>

```

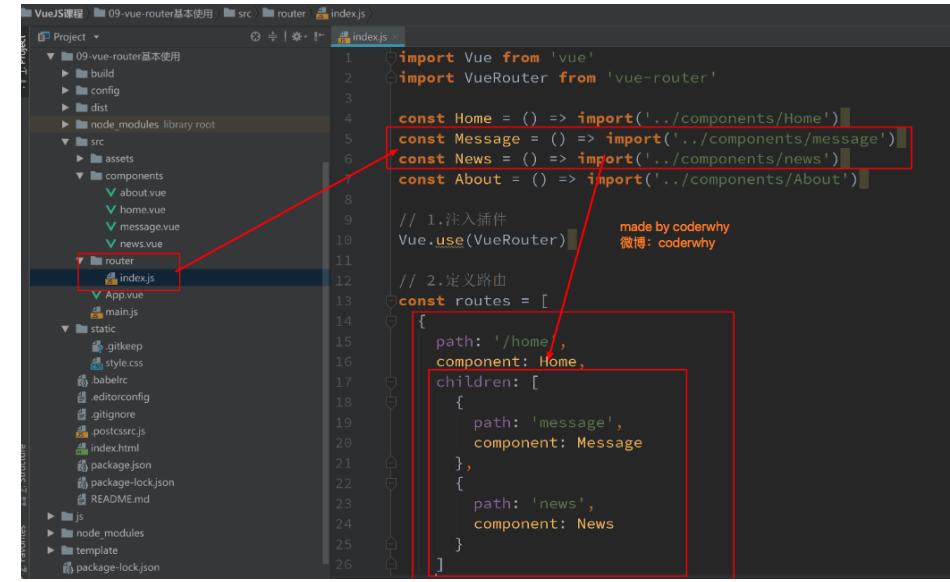


```

<template>
  <div>
    <ul>
      <li>news1</li>
      <li>news2</li>
      <li>news3</li>
      <li>news4</li>
      <li>news5</li>
    </ul>
  </div>
</template>

<script>
export default {
  name: "news"
}
</script>

```



```

import Vue from 'vue'
import VueRouter from 'vue-router'

const Home = () => import('../components/Home')
const Message = () => import('../components/message')
const News = () => import('../components/news')
const About = () => import('../components/About')

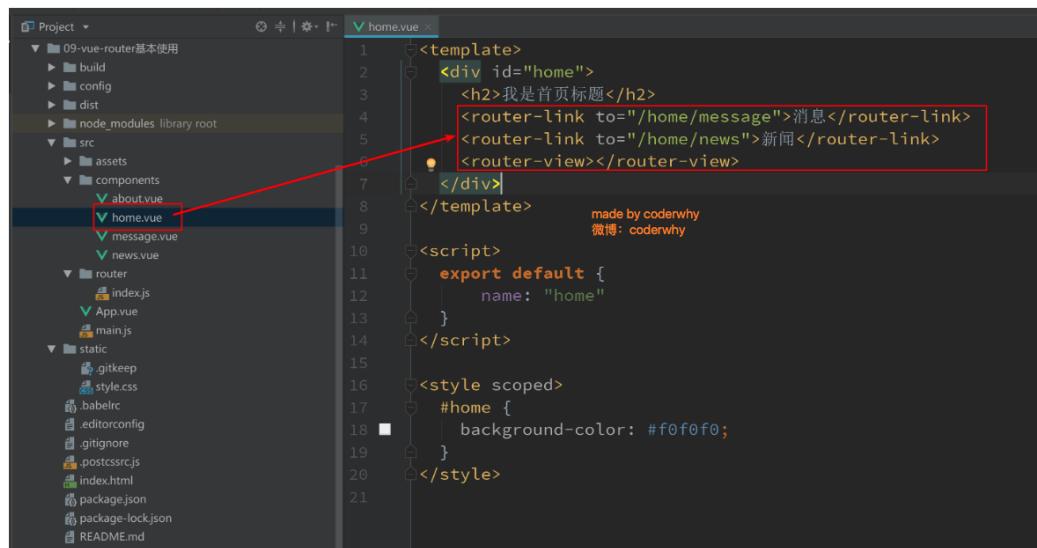
// 1.注入插件
Vue.use(VueRouter)

// 2.定义路由
const routes = [
  {
    path: '/home',
    component: Home,
    children: [
      {
        path: 'message',
        component: Message
      },
      {
        path: 'news',
        component: News
      }
    ]
  }
]

const router = new VueRouter({
  routes
})

// 使用全局前置守卫
router.beforeEach((to, from, next) => {
  console.log(`从${from.path}跳转到${to.path}`)
  next()
})

```



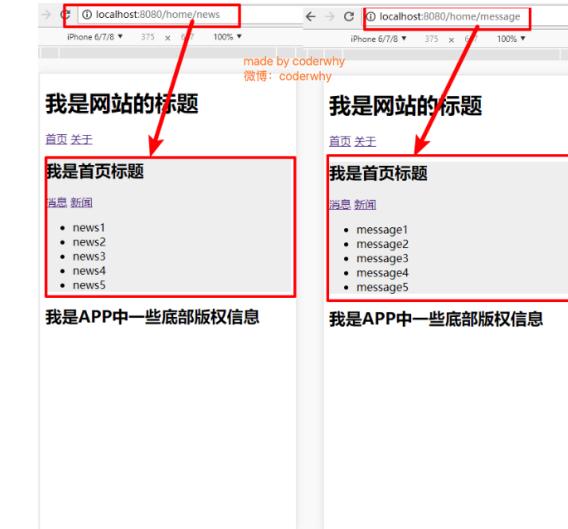
```

<template>
  <div id="home">
    <h2>我是首页标题</h2>
    <router-link to="/home/message">消息</router-link>
    <router-link to="/home/news">新闻</router-link>
    <router-view></router-view>
  </div>
</template>

<script>
export default {
  name: "home"
}
</script>

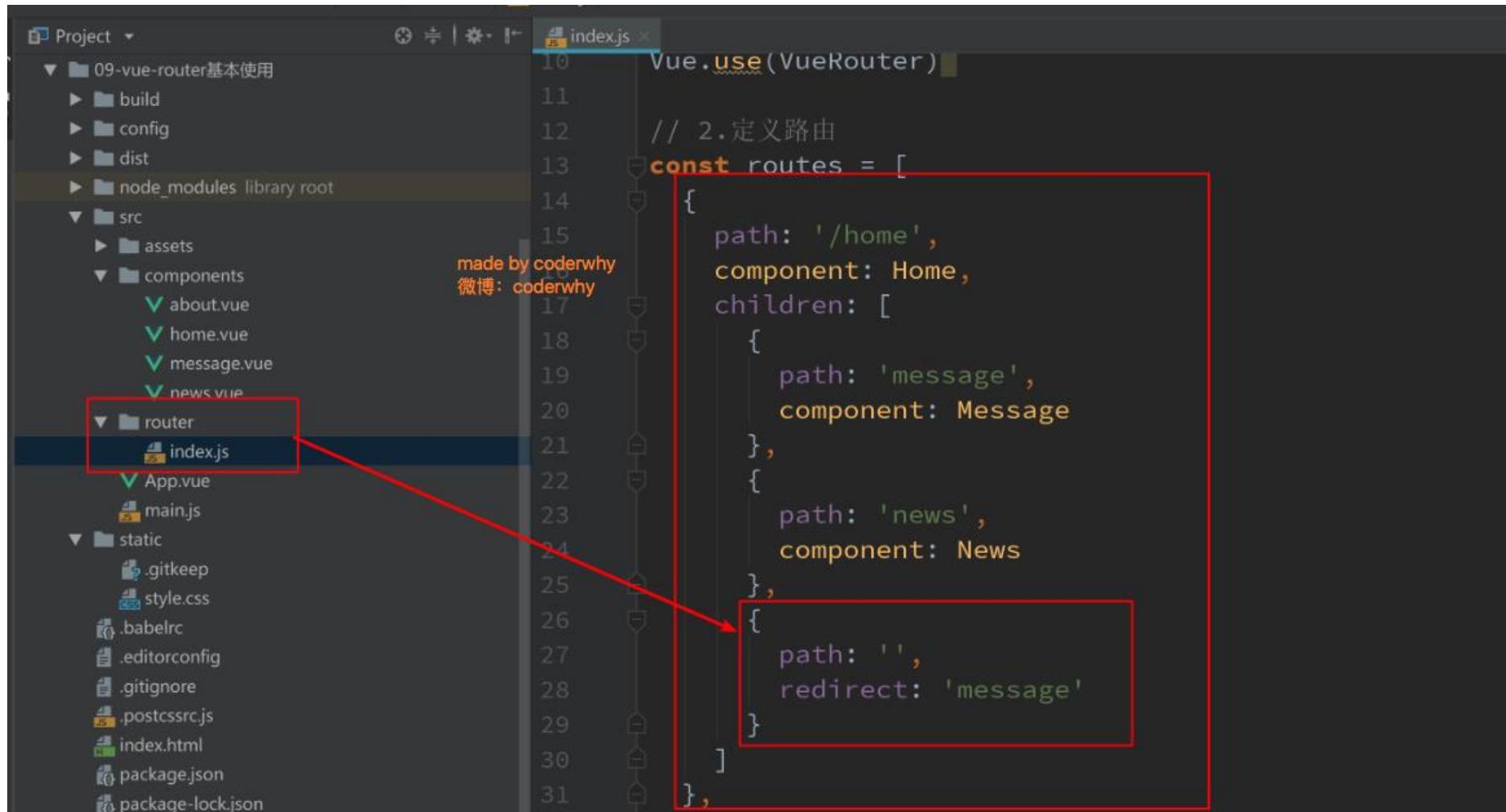
<style scoped>
#home {
  background-color: #f0f0f0;
}

```



嵌套默认路径

■ 嵌套路由也可以配置默认的路径, 配置方式如下:



The screenshot shows a code editor with a dark theme. On the left is a file tree for a project named "09-vue-router基本使用". The "src/router/index.js" file is selected and shown in the editor. The code defines a routes array with two main entries. The first entry is a nested route under "/home" that points to a "Message" component. The second entry is a default route that redirects to the "message" path. A red box highlights the entire routes array, and another red box highlights the default redirect entry.

```
10  Vue.use(VueRouter)
11
12  // 2. 定义路由
13  const routes = [
14    {
15      path: '/home',
16      component: Home,
17      children: [
18        {
19          path: 'message',
20          component: Message
21        },
22        {
23          path: 'news',
24          component: News
25        }
26      ],
27      path: '',
28      redirect: 'message'
29    }
30  ],
31  ]
32 },
```

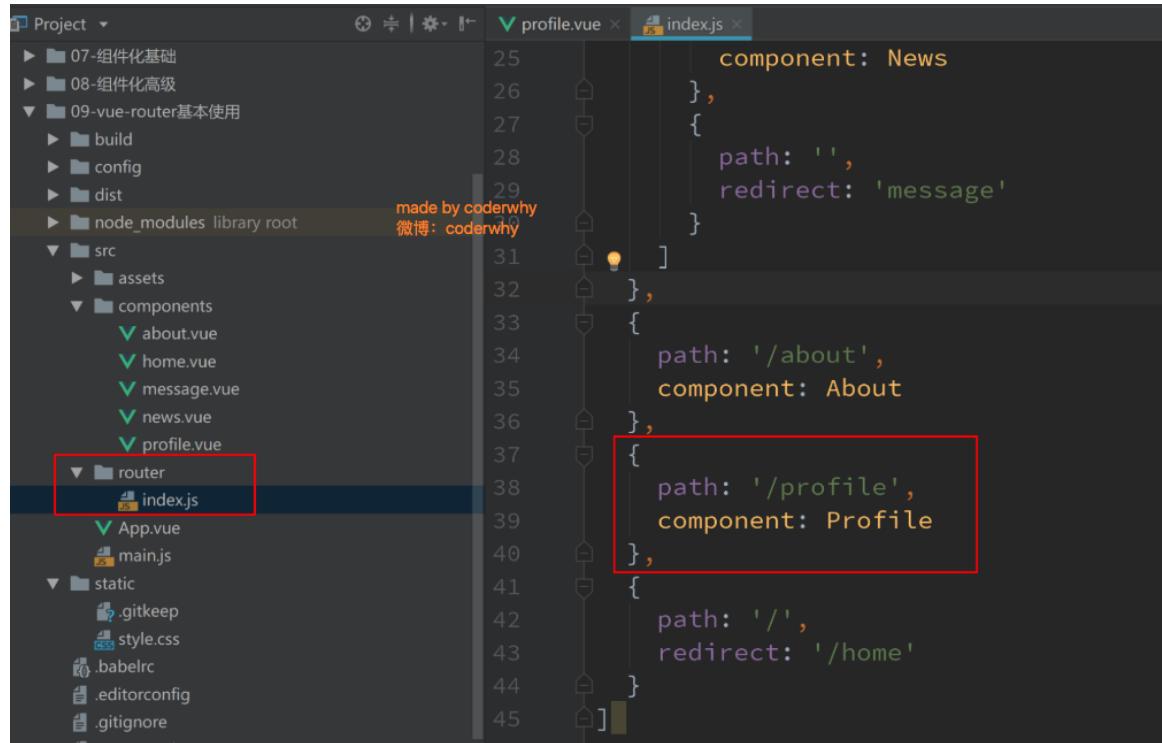
准备工作

■ 为了演示传递参数, 我们这里再创建一个组件, 并且将其配置好

□ 第一步: 创建新的组件Profile.vue

□ 第二步: 配置路由映射

□ 第三步: 添加跳转的<router-link>



```

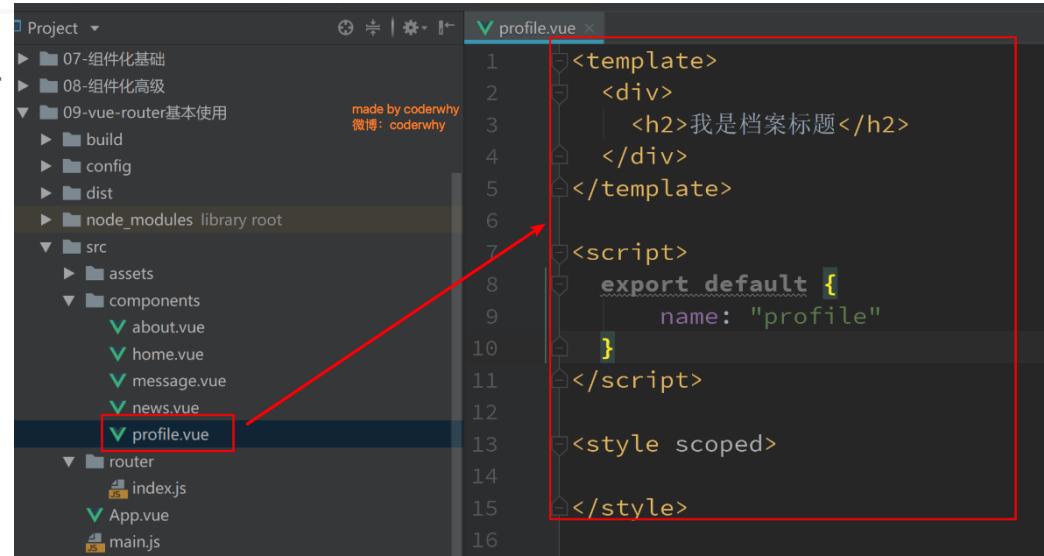
Project
07-组件化基础
08-组件化高级
09-vue-router基本使用
build
config
dist
node_modules library root
src
  assets
  components
    about.vue
    home.vue
    message.vue
    news.vue
    profile.vue
  router
    index.js
    App.vue
    main.js
static
  .gitkeep
  style.css
.babelrc
.editorconfig
.gitignore

```

```

25      component: News
26    },
27    {
28      path: '',
29      redirect: 'message'
30    }
31  ],
32  ],
33  [
34    {
35      path: '/about',
36      component: About
37    },
38    {
39      path: '/profile',
40      component: Profile
41    },
42    {
43      path: '/',
44      redirect: '/home'
45    }

```



```

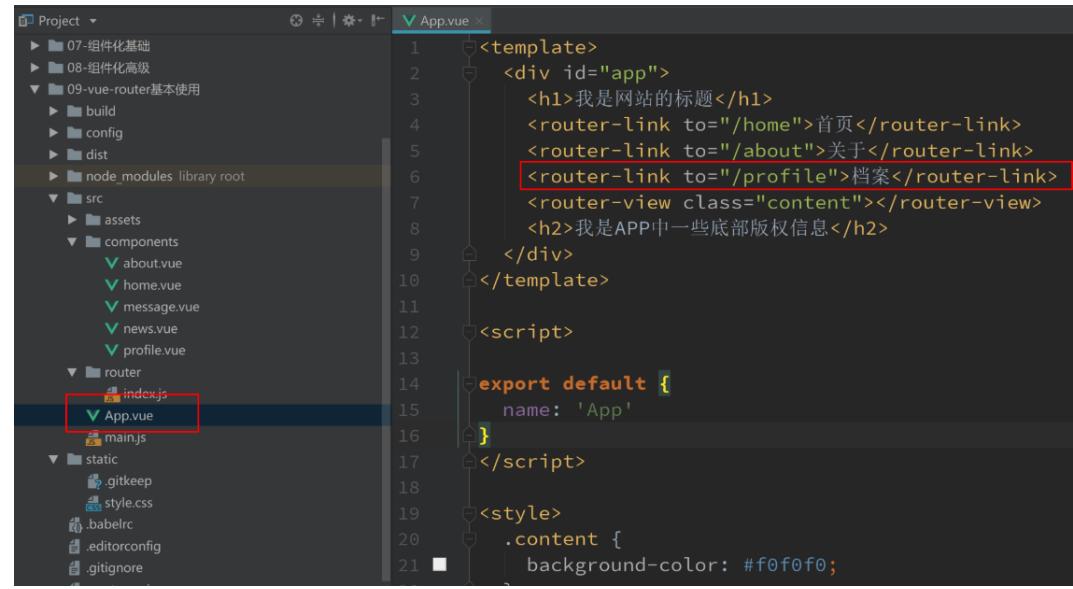
Project
07-组件化基础
08-组件化高级
09-vue-router基本使用
build
config
dist
node_modules library root
src
  assets
  components
    about.vue
    home.vue
    message.vue
    news.vue
    profile.vue
  router
    index.js
    App.vue
    main.js

```

```

1 <template>
2   <div>
3     <h2>我是档案标题</h2>
4   </div>
5 </template>
6 <script>
7   export default {
8     name: "profile"
9   }
10 </script>
11 <style scoped>
12
13
14
15 </style>
16

```



```

Project
07-组件化基础
08-组件化高级
09-vue-router基本使用
build
config
dist
node_modules library root
src
  assets
  components
    about.vue
    home.vue
    message.vue
    news.vue
    profile.vue
  router
    index.js
    App.vue
    main.js
static
  .gitkeep
  style.css
.babelrc
.editorconfig
.gitignore

```

```

1 <template>
2   <div id="app">
3     <h1>我是网站的标题</h1>
4     <router-link to="/home">首页</router-link>
5     <router-link to="/about">关于</router-link>
6     <router-link to="/profile">档案</router-link>
7     <router-view class="content"></router-view>
8     <h2>我是APP中一些底部版权信息</h2>
9   </div>
10 </template>
11 <script>
12
13
14   export default {
15     name: 'App'
16   }
17 </script>
18 <style>
19   .content {
20     background-color: #f0f0f0;
21   }
22 </style>

```



传递参数的方式

- 传递参数主要有两种类型: params和query
- **params的类型:**
 - 配置路由格式: `/router/:id`
 - 传递的方式: 在path后面跟上对应的值
 - 传递后形成的路径: `/router/123, /router/abc`
- **query的类型:**
 - 配置路由格式: `/router`, 也就是普通配置
 - 传递的方式: 对象中使用query的key作为传递方式
 - 传递后形成的路径: `/router?id=123, /router?id=abc`
- 如何使用它们呢? 也有两种方式: `<router-link>` 的方式和JavaScript代码方式



传递参数方式一: <router-link>

The screenshot shows a code editor with a dark theme. On the left is a file tree:

- Project
- node_modules library root
- src
 - assets
 - components
 - about.vue
 - home.vue
 - message.vue
 - news.vue
 - profile.vue
 - router
 - index.js
 - App.vue
 - main.js- static
 - .gitkeep
 - style.css
- .babelrc
- .editorconfig
- .gitignore
- .postcssrc.js
- index.html

In the center, the `App.vue` file is open. It contains the following code:

```
<template>
  <div id="app">
    <h1>我是网站的标题</h1> params
    <router-link to="/home">首页</router-link>
    <router-link to="/about">关于</router-link>
    <router-link :to="{>
      path: '/profile/' + 123,>
      query: {name: 'why', age: 18}>
    }">档案</router-link>
    <router-view class="content"></router-view>
    <h2>我是APP中一些底部版权信息</h2> query
  </div>
</template>
<script>
```

A red box highlights the `:to` prop of the third `<router-link>` component. A red arrow points from the word "params" in the first `<h1>` tag to the `:to` prop. Another red arrow points from the word "query" in the last `<h2>` tag to the `query` object in the `:to` prop.



传递参数方式二：JavaScript代码

The screenshot shows a code editor with a file tree on the left and a script component on the right.

File Tree:

- node_modules library root
- src
 - assets
 - components
 - about.vue
 - home.vue
 - message.vue
 - news.vue
 - profile.vue
 - router
 - index.js
 - App.vue
 - main.js
- static
 - .gitkeep
 - style.css
 - .babelrc
 - .editorconfig
 - .gitignore

Script Component (App.vue):

```
16 </template>
17
18 <script>
19
20 export default {
21   name: 'App',
22   methods: {
23     toProfile() {
24       this.$router.push({
25         path: '/profile/' + 123,
26         query: {name: 'why', age: 18}
27       })
28     }
29   }
30 }
31 </script>
```

A red box highlights the `this.$router.push({ path: '/profile/' + 123, query: {name: 'why', age: 18} })` line. A yellow lightbulb icon is positioned above the `push` method call.

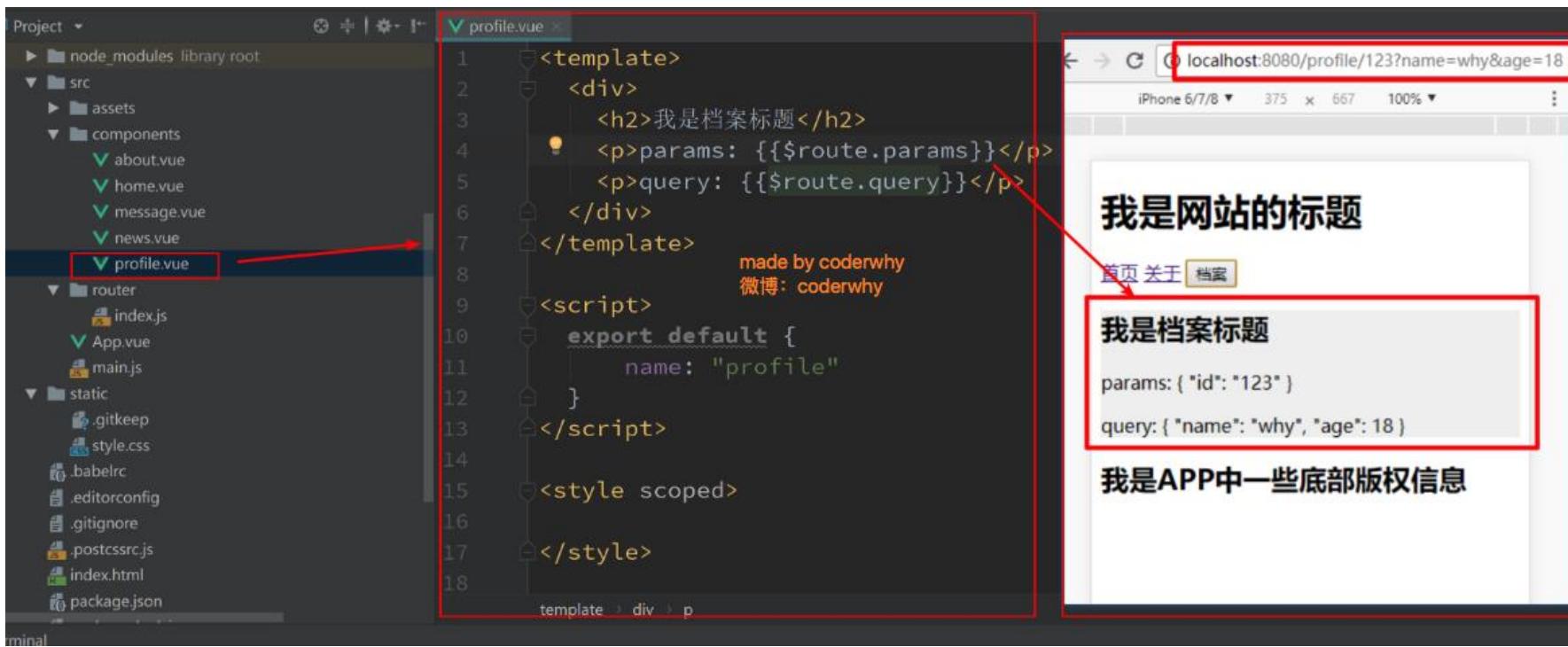
Watermark: made by coderwhy
微博: coderwhy

获取参数

■ 获取参数通过\$route对象获取的。

□ 在使用了 vue-router 的应用中，路由对象会被注入每个组件中，赋值为 this.\$route，并且当路由切换时，路由对象会被更新。

■ 通过\$route获取传递的信息如下：



The image shows a Vue.js project structure in a code editor. The project includes components like about.vue, home.vue, message.vue, news.vue, and profile.vue. The profile.vue component is selected and shown in the code editor. The code uses `params: {{$route.params}}` and `query: {{$route.query}}` to access route parameters and query strings. The rendered output in the browser shows the title "我是网站的标题" and the subtitle "我是档案标题". The browser URL is `localhost:8080/profile/123?name=why&age=18`. A red box highlights the browser's address bar and the rendered subtitle.

```
<template>
  <div>
    <h2>我是网站的标题</h2>
    <p>params: {{$route.params}}</p>
    <p>query: {{$route.query}}</p>
  </div>
</template>
<script>
  export default {
    name: "profile"
  }
</script>
<style scoped>
</style>
```

\$route和\$router是有区别的

■ \$route和\$router是有区别的

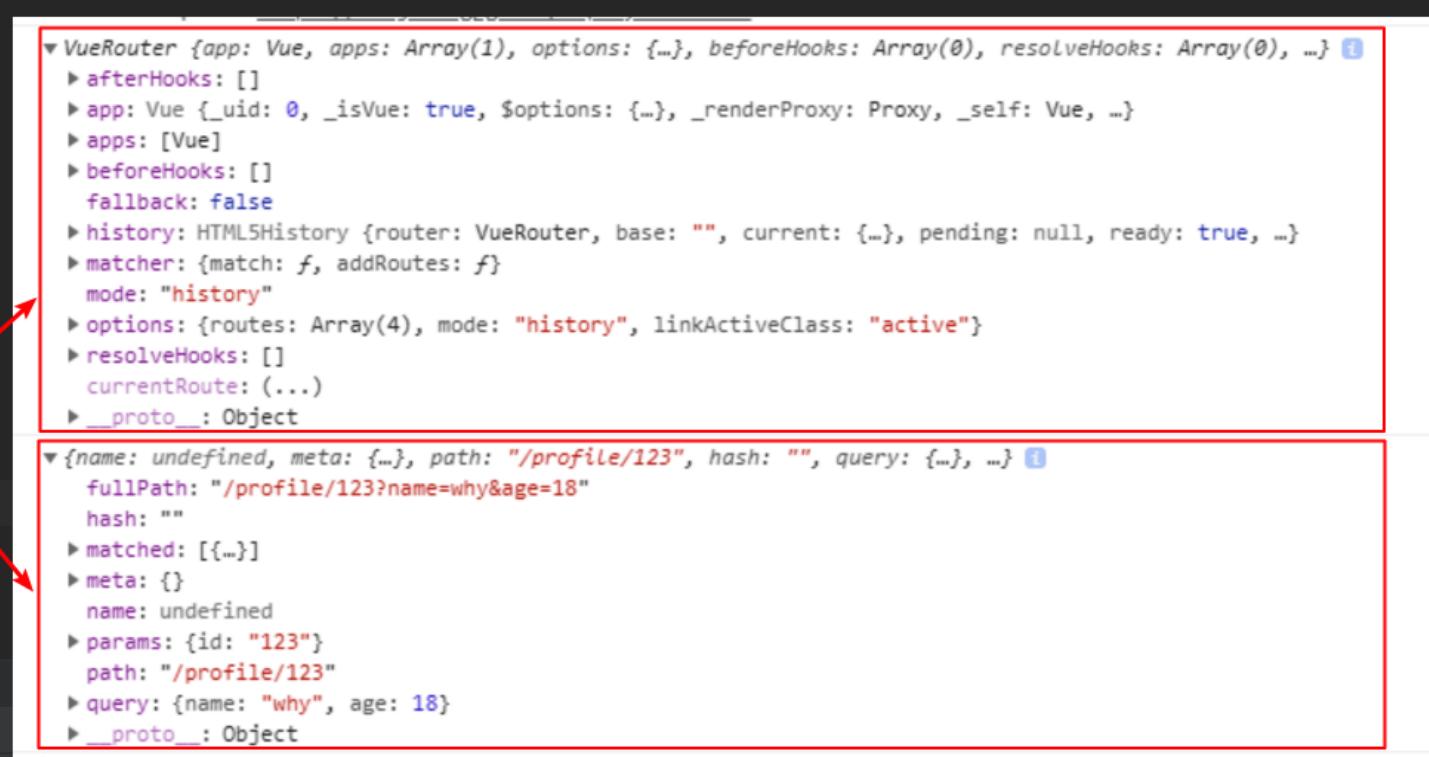
- \$router为VueRouter实例，想要导航到不同URL，则使用\$router.push方法
- \$route为当前router跳转对象里面可以获取name、path、query、params等

```
<p>params: {{$route.params}}</p>
<p>query: {{$route.query}}</p>
</div>
</template>

made by coderwhy
微博: coderwhy

<script>
export default {
  name: "profile",
  mounted() {
    console.log(this.$router);
    console.log(this.$route);
  }
}
</script>

<script>
local (1)
```



```
▼ VueRouter {app: Vue, apps: Array(1), options: {...}, beforeHooks: Array(0), resolveHooks: Array(0), ...} ⓘ
  ► afterHooks: []
  ► app: Vue {_uid: 0, _isVue: true, $options: {...}, _renderProxy: Proxy, _self: Vue, ...}
  ► apps: [Vue]
  ► beforeHooks: []
  ► fallback: false
  ► history: HTML5History {router: VueRouter, base: "", current: {...}, pending: null, ready: true, ...}
  ► matcher: {match: f, addRoutes: f}
  ► mode: "history"
  ► options: {routes: Array(4), mode: "history", linkActiveClass: "active"}
  ► resolveHooks: []
  ► currentRoute: (...)

  ► __proto__: Object

▼ {name: undefined, meta: {...}, path: "/profile/123", hash: "", query: {...}, ...} ⓘ
  ► fullPath: "/profile/123?name=why&age=18"
  ► hash: ""
  ► matched: [{}]
  ► meta: {}
  ► name: undefined
  ► params: {id: "123"}
  ► path: "/profile/123"
  ► query: {name: "why", age: 18}
  ► __proto__: Object
```

为什么使用导航守卫?

■ 我们来考虑一个需求: 在一个SPA应用中, 如何改变网页的标题呢?

- 网页标题是通过<title>来显示的, 但是SPA只有一个固定的HTML, 切换不同的页面时, 标题并不会改变.
- 但是我们可以通过JavaScript来修改<title>的内容>window.document.title = '新的标题'.
- 那么在Vue项目中, 在哪里修改? 什么时候修改比较合适呢?

■ 普通的修改方式:

- 我们比较容易想到的修改标题的位置是每一个路由对应的组件.vue文件中.
- 通过mounted声明周期函数, 执行对应的代码进行修改即可.
- 但是当页面比较多时, 这种方式不容易维护(因为需要在多个页面执行类似的代码).

■ 有没有更好的办法呢? 使用导航守卫即可.

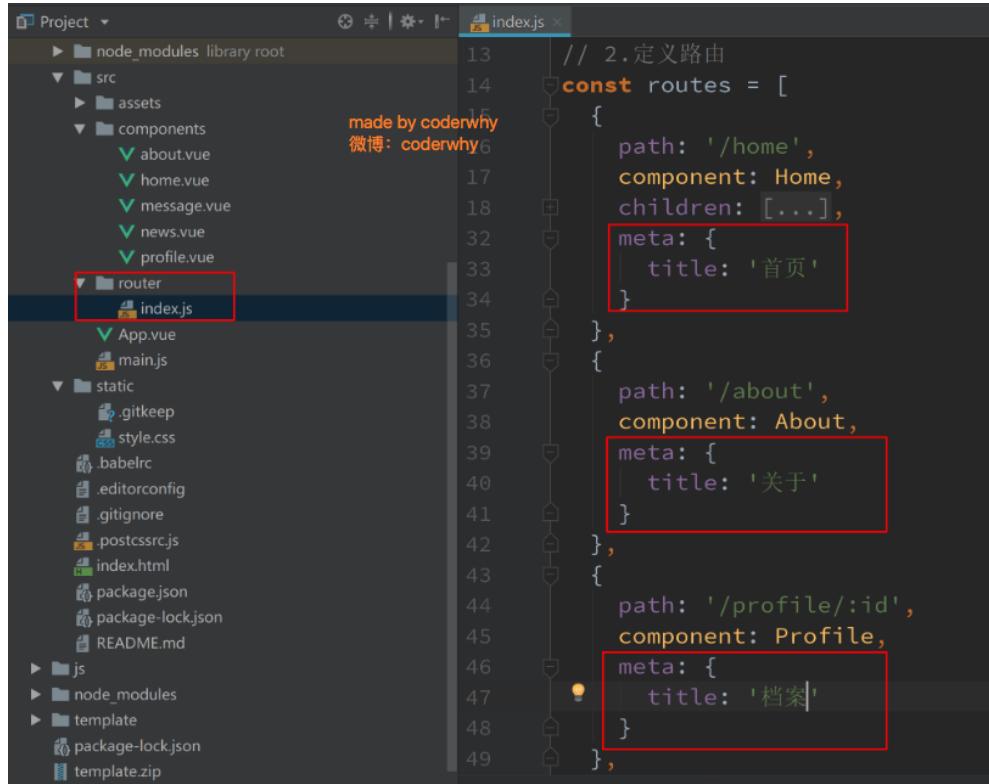
■ 什么是导航守卫?

- vue-router提供的导航守卫主要用来监听监听路由的进入和离开的.
- vue-router提供了beforeEach和afterEach的钩子函数, 它们会在路由即将改变前和改变后触发.

导航守卫使用

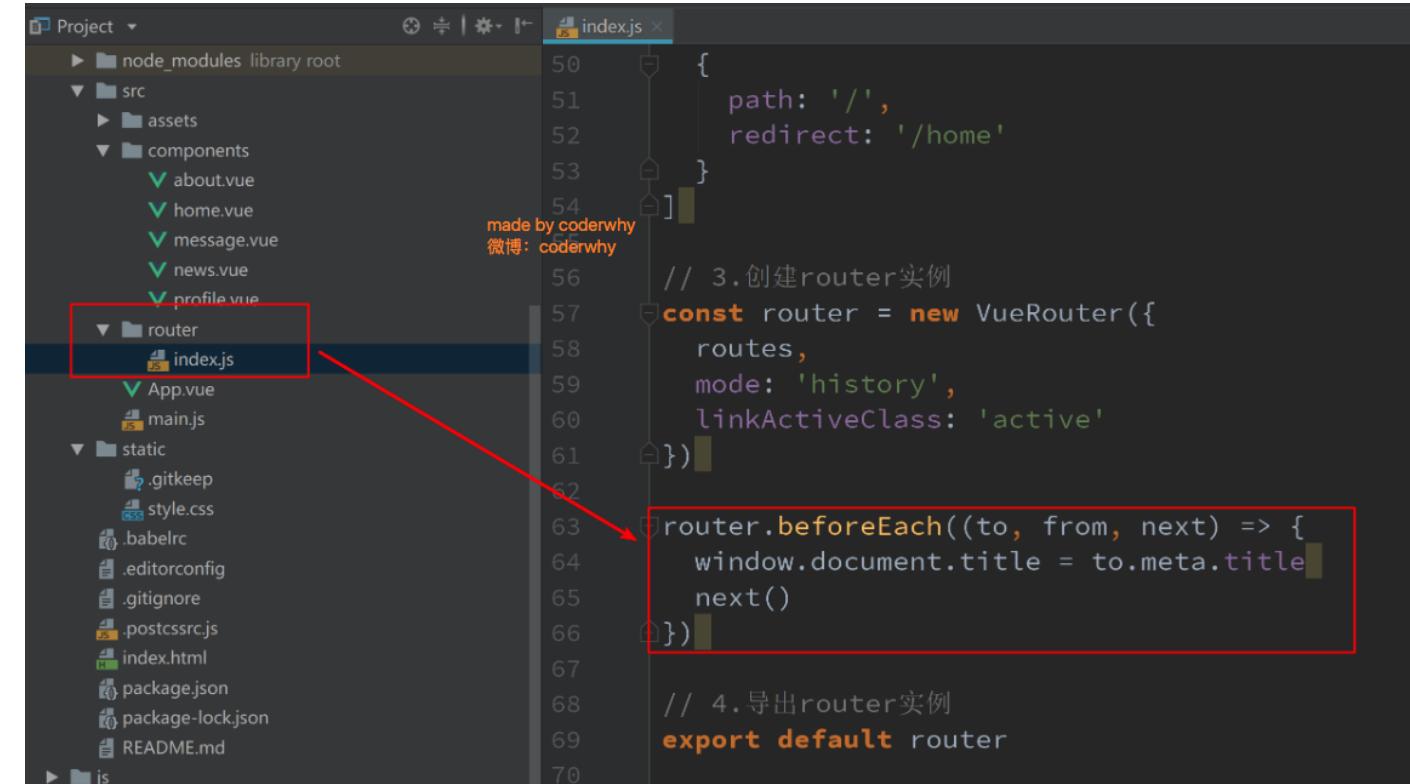
■ 我们可以利用beforeEach来完成标题的修改.

- 首先, 我们可以在钩子当中定义一些标题, 可以利用meta来定义
- 其次, 利用导航守卫, 修改我们的标题.



```

13 // 2. 定义路由
14 const routes = [
15   {
16     path: '/home',
17     component: Home,
18     children: [...],
19     meta: {
20       title: '首页'
21     }
22   },
23   {
24     path: '/about',
25     component: About,
26     meta: {
27       title: '关于'
28     }
29   },
30   {
31     path: '/profile/:id',
32     component: Profile,
33     meta: {
34       title: '档案'
35     }
36   },
37 ],
38 }
39 
```



```

50 path: '/',
51 redirect: '/home'
52 }
53 ]
54 ]
55
56 // 3. 创建router实例
57 const router = new VueRouter({
58   routes,
59   mode: 'history',
60   linkActiveClass: 'active'
61 })
62
63 router.beforeEach((to, from, next) => {
64   window.document.title = to.meta.title
65   next()
66 })
67
68 // 4. 导出router实例
69 export default router
70 
```

■ 导航钩子的三个参数解析:

- to: 即将要进入的目标的路由对象.
- from: 当前导航即将要离开的路由对象.
- next: 调用该方法后, 才能进入下一个钩子.



导航守卫补充

- 补充一:如果是后置钩子, 也就是`afterEach`, 不需要主动调用`next()`函数.
- 补充二: 上面我们使用的导航守卫, 被称之为**全局守卫**.
 - 路由独享的守卫.
 - 组件内的守卫.
- 更多内容, 可以查看官网进行学习:
 - <https://router.vuejs.org/zh/guide/advanced/navigation-guards.html#%E8%B7%AF%E7%94%B1%E7%8B%AC%E4%BA%AB%E7%9A%84%E5%AE%88%E5%8D%AB>



keep-alive遇见vue-router

- keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。
 - 它们有两个非常重要的属性：
 - include - 字符串或正则表达，只有匹配的组件会被缓存
 - exclude - 字符串或正则表达式，任何匹配的组件都不会被缓存
- router-view 也是一个组件，如果直接被包在 keep-alive 里面，所有路径匹配到的视图组件都会被缓存：

```
<keep-alive>
  <router-view>
    <!-- 所有路径匹配到的视图组件都会被缓存！ -->
  </router-view>
</keep-alive>
```

- 通过create声明周期函数来验证



TabBar实现思路

■ 1. 如果在下方有一个单独的TabBar组件，你如何封装

- 自定义TabBar组件，在APP中使用
- 让TabBar出于底部，并且设置相关的样式

■ 2.TabBar中显示的内容由外界决定

- 定义插槽
- flex布局平分TabBar

■ 3.自定义TabBarItem，可以传入 图片和文字

- 定义TabBarItem，并且定义两个插槽：图片、文字。
- 给两个插槽外层包装div，用于设置样式。
- 填充插槽，实现底部TabBar的效果

首页





TabBar实现思路

■ 4.传入高亮图片

- 定义另外一个插槽，插入active-icon的数据
- 定义一个变量isActive，通过v-show来决定是否显示对应的icon

■ 5.TabBarItem绑定路由数据

- 安装路由：npm install vue-router —save
- 完成router/index.js的内容，以及创建对应的组件
- main.js中注册router
- APP中加入<router-view>组件

■ 6.点击item跳转到对应路由，并且动态决定isActive

- 监听item的点击，通过this.\$router.replace()替换路由路径
- 通过this.\$route.path.indexOf(this.link) !== -1来判断是否是active

■ 7.动态计算active样式

- 封装新的计算属性：**this.isActive ? {'color': 'red'} : {}**



代码实现

```
<template>
  <div id="tab-bar">
    <slot></slot>
  </div>
</template>

<script>
  export default {
    name: "TabBar"
  }
</script>

<style scoped>
#tab-bar {
  /* 本身的样式 */
  background-color: #f6f6f6;
  height: 49px;
  border-top: 1px solid #eee;
  box-shadow: 0px -1px 1px rgba(150,150,150,.08);

  /* 定位相关 */
  position: fixed;
  left: 0;
  right: 0;
  bottom: 0;

  /* 利用flex进行布局 */
  display: flex;
  text-align: center;
}
</style>
```

```
<template>
  <div id="tab-bar-item" @click="itemClick">
    <div class="item-icon" v-if="isActive"><slot name="icon"></slot></div>
    <div class="item-active-icon" v-if="!isActive"><slot name="active-icon"></slot></div>
    <div class="item-text" :style="activeStyle"><slot name="text"></slot></div>
  </div>
</template>

<script>
  export default {
    name: "HYTabBarItem",
    props: {
      link: {
        type: String,
        required: true
      },
      activeColor: {
        type: String,
        default: '#ff5777'
      }
    },
    computed: {
      isActive() {
        return this.$route.path.indexOf(this.link) === -1
      },
      activeStyle() {
        return this.isActive ? {} : {'color': this.activeColor}
      }
    },
    methods: {
      itemClick() {
        this.$router.replace(this.link)
      }
    }
  }
</script>
```