**Vue Router**

是 Vue 官方的客户端路由解决方案。

客户端路由的作用是在单页应用 (SPA) 中将浏览器的 URL 和用户看到的内容绑定起来。当用户在应用中浏览不同页面时，URL 会随之更新，但页面不需要从服务器重新加载。

Vue Router 基于 Vue 的组件系统构建，你可以通过配置路由来告诉 Vue Router 为每个 URL 路径显示哪些组件。

Vue Router 是 Vue.js 的“导航系统”，它让你可以在一个单页应用（SPA）中：

* 在多个“页面”之间切换而不刷新页面
* 给每个页面配置不同的路由路径（URL）
* 管理组件之间的跳转和传参
* 做权限控制、过渡动画等功能

**核心概念**

概念 作用

routes 路由表 定义路径和组件的对应关系

<router-view /> 显示当前路由对应的组件

<router-link /> 导航组件，点击后改变路由

useRouter() 拿到 router 实例，用来跳转页面等

useRoute() 获取当前路由信息（path、query 等）

**使用步骤：**

**首先需要定义项目的目录结构：**

src/

├── App.vue

├── main.js

├── router/

│ └── index.js

├── views/

│ ├── Home.vue

│ └── Note.vue

1. 安装 Vue Router

如果你还没安装：

npm install vue-router

2.main.js

import { createApp } from 'vue'

import App from './App.vue'

import router from './router'

createApp(App).use(router).mount('#app')

3. 配置路由（通常在router/index.js）

import { createRouter, createWebHistory } from 'vue-router'

import Home from './components/Home.vue'

import Note from './components/Note.vue'

const routes = [

{ path: '/', component: Home },

{ path: '/note', component: Note }

]

//定义路由表

const router = createRouter({

//用createRouter方法创建一个路由实例，用于挂载到你的 Vue 应用中（通过 .use(router)）

history: createWebHistory(),

//设置路由的模式为 HTML5 的 history 模式，也就是 URL 中没有 # 的方式

routes

//路由表（数组），定义了哪些路径对应哪些组件

})

export default router

1. 在 App.vue 中使用

<template>

<nav>

<router-link to="/">首页</router-link> |

<router-link to="/note">写笔记</router-link>

</nav>

<router-view /> //显示路由加载内容

</template>

<script setup>

// 无需逻辑，router-view 会显示当前页面组件

</script>

1. 定义首页 Home.vue

<template>

<div>

<h2>这是首页</h2>

<button @click="goToNote">去写笔记</button>

</div>

</template>

<script setup>

import { useRouter } from 'vue-router'

const router = useRouter() // 用来跳转

function goToNote() {

router.push({ path: '/note', query: { id: 123 } })

}

</script>

1. 定义跳转目标页（即Note.vue）

<template>

<div>

<h2>写笔记页面</h2>

<p>你传来的参数是：<strong>{{ route.query.id }}</strong></p>

<button @click="goBack">返回</button>

</div>

</template>

<script setup>

import { useRouter, useRoute } from 'vue-router'

const router = useRouter() //要跳回主页（跳转）

const route = useRoute() //要拿参数，如 route.query.id

//注意！！此时的useRouter() 和 useRoute()的作用：

//useRouter() 是 “我想跳去哪里”（跳转用）

//useRoute() 是 “我现在在哪儿 + 有什么参数”（读取当前路由状态）

function goBack() {

router.push('/')

}

</script>

**启动后的效果：**

* 打开首页看到按钮，点击跳转到 /note?id=123
* 在写笔记页看到 你传来的参数是：123
* 点击“返回”跳回首页

**查询参数（query）**

查询参数指的是 URL 中 ?后面的部分

即：

/note?id=123&title=hello

相当于：

{

id: '123',

title: 'hello'

}

在上文的例子的第四步中，用 route.query 来访问这些参数

**查询参数 VS 动态参数 对比**

| **项目** | **查询参数（Query）** | **动态参数（Params）** |
| --- | --- | --- |
| 格式 | /note?id=123 | /note/123 |
| 获取方式 | route.query.id | route.params.id |
| 是否可选 | ✅ 通常是可选的 | ❌ 一般是必填的 |
| 用途 | 搜索、筛选、分页等 | 详情页、编辑页等 |
| 是否在路由 path 中声明 | ❌ 不需要 | ✅ 必须定义 :id |

也就是说：查询参数适用于“可选信息”，比如筛选、搜索、分页，灵活、不影响页面路径结构。

**带参数的动态路由匹配--动态参数（Params）**

它的作用是让“页面根据不同的参数显示不同的内容”

也就是说：用一个组件，可以展示不同的数据页面

就是 Vue Router 中这样写的路由：

{ path: '/note/:id', component: Note }

**它的意思是：4**

/note/123、/note/abc、/note/999 都可以匹配上这个路由，只要它是 /note/某个东西。

这个 :id 就是“动态参数”——它不是写死的，而是访问时由 URL 决定

**动态参数的读取和使用和查询参数（query）类似：**

route.params.id

用来访问这些参数

例如：

const noteId = route.params.id

**动态参数（Params）的变化 和 watch监听器的结合使用**

注意！！当用户从 /users/aaa 导航到 /users/bbb 时，相同的组件实例将被重复使用。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。不过，这也意味着组件的生命周期钩子不会被调用。

也就是说，当获取user信息的逻辑如下时：

mounted() {

// 拿用户 ID 并加载数据

fetchUser(this.$route.params.id)

}

第一次进入 /users/aaa 会执行这个逻辑

但从 /users/aaa 切换到 /users/bbb，不会再执行一次

所以，这时页面不会更新，依然显示 aaa 的资料

为了解决这个问题，就需要用到监听器watch（），去实时追踪$route.params.id的变化，当它变了就主动更新数据：

<script setup>

import { useRoute, watch } from 'vue-router'

const route = useRoute()

// 监听动态参数变化

watch(() => route.params.id, (newId) => {

fetchUser(newId)

})

// 首次加载

fetchUser(route.params.id)

function fetchUser(id) {

console.log('正在加载用户：', id)

// ...执行 API 请求或本地查找

}

</script>

这样就能实现每次切换用户，用户信息也随之切换

**嵌套路由（Nested Routes）**

Vue Router 的核心功能之一，适用于构建多级页面结构，比如：

* 用户中心 ➜ 资料页 / 设置页
* 文章详情页 ➜ 评论 / 相关推荐
* 后台管理系统的左侧菜单 + 主体内容区域

**案例：**

/user/profile /user/settings

┌────────┐ ┌─────────┐

│ User │ │ User │

│ ┌─────┐ │ │ ┌──────┐ │

│ │ Profile │ │ ────> │ │ settings │ │

│ │ │ │ │ │ │ │

│ └─────┘ │ │ └──────┘ │

└────────┘ └─────────┘

1. 定义index.js

import User from '../views/User.vue'

import Profile from '../views/Profile.vue'

import Settings from '../views/Settings.vue'

const routes = [

{

path: '/user',

component: User,

children: [

{

path: 'profile',

component: Profile

},

{

path: 'settings',

component: Settings

}

]

}

]

注意！！children 里是子路由，路径不要加 /，它是相对父路径的。

1. 定义User.vue（父组件）

<template>

<div>

<h2>用户中心</h2>

<router-link to="/user/profile">资料</router-link> |

<router-link to="/user/settings">设置</router-link>

<!-- 子路由显示在这里 -->

<router-view />

</div>

</template>

1. 定义子组件

**Profile.vue**

<template>

<h3>这里是用户资料页</h3>

</template>

**Settings.vue**

<template>

<h3>这里是用户设置页</h3>

</template>

**嵌套的命名路由**

在多层嵌套的应用中，比如后台系统、用户中心、内容管理等，给 **嵌套路由中也给每个子路由配置 name**，这样你就可以非常灵活地通过名字跳转到子页面。也就是说：给每个路由配置一个 name，你就可以用 ：

router.push({ name: 'xxx' })

来跳转，而不必写死路径字符串。

**比如将上文中的例子改成：**

const routes = [

{

path: '/user', name: 'user', component: User,

children: [

{ path: 'profile', name: 'user-profile', component: Profile },

{ path: 'settings', name: 'user-settings', component: Settings }

]

}

]

**怎么跳转到嵌套的命名路由：**

// 跳转到用户资料页

router.push({ name: 'user-profile' })

// 或者带参数（如果路径中包含 :username 等动态参数）

router.push({ name: 'user-settings', params: { username: 'jack' } })

**在模板中使用命名路由：**

<router-link :to="{ name: 'user-profile' }">

去用户资料页

</router-link>

这样写比硬编码路径（to="/user/profile"）更安全、更灵活。

**忽略父组件**

我们还可以仅利用路由的父子关系，但不嵌套路由组件。这对于将具有公共路径前缀的路由分组在一起或使用更高级的功能时很有用，例如：路由独享的守卫或路由元信息。

为了实现这一点, 直接在父路由中省略 component 和 components 选项

const routes = [

{

path: '/user',

children: [

{ path: 'profile', name: 'user-profile', component: Profile },

{ path: 'settings', name: 'user-settings', component: Settings }

]

}

]

由于父级没有指定路由组件，顶级 <router-view> 将跳过父级并仅使用子路由组件

**编程式导航**

用 JavaScript 的方式，通过 Vue Router 的实例在代码中控制页面跳转。简单说：不是点 <router-link>，而是自己在脚本中用 router.push() 等方法。比模板里的 <router-link> 更灵活，适合在点击按钮、逻辑判断、表单提交后跳转页面。

常用方法：

| **方法** | **作用** |
| --- | --- |
| router.push() | 跳转到一个新地址（可以带参数） |
| router.replace() | 跳转但不保留历史记录（不会添加浏览器返回） |
| router.go(n) | 类似浏览器的 history.go(n) |

**在浏览器里前进/后退，也就是返回上一页（或复数页）：**

router.go(-100) // 返回上...页（如果没有那么多记录，静默失败）

router.go(1) // 前进一步

**实际应用场景：**

场景 代码示例

登录成功后跳转首页 router.push('/home')

表单提交后跳转并带参数 router.push({ name: 'result', query: { id: 1 } })

删除操作后跳转列表页并不保留历史 router.replace('/list')

详情页跳转下一项 router.push({ name: 'item', params: { id: nextId } })

**命名视图**

有时候想同时 (同级) 展示多个视图，而不是嵌套展示，例如创建一个布局，有 sidebar (侧导航) 和 main (主内容) 两个视图，这个时候命名视图就派上用场了。你可以在界面中拥有多个单独命名的视图，而不是只有一个单独的出口。如果 router-view 没有设置名字，那么默认为 default。

**比如，在app.vue中的template中：**

<router-view class="view header" name="header" />

<router-view class="view main-content" />

<router-view class="view sidebar" name="sidebar" />

这些 <router-view> 标签会在模板里占位，并对应渲染指定的视图组件

**在index.js中：**

const routes = [

{

path: '/settings',

components: {

default: SettingsContent,

sidebar: SettingsSidebar,

header: HeaderComponent

}

}

]

注意！！

* 不是 component，而是 components（复数）！
* 每个键名就是你在模板中 <router-view name="xxx" /> 的名字。
* 没有写 name 的 <router-view>，会显示 default。

**你访问 /dashboard 时，会渲染：**

* SettingsContent → 到默认 <router-view />
* SettingsSidebar → 到 <router-view name="sidebar" />
* HeaderComponent → 到 <router-view name="header" />

**嵌套命名视图**

我们也有可能使用命名视图创建嵌套视图的复杂布局。这时你也需要命名用到的嵌套 router-view 组件。我们以一个设置面板为例：



* Nav 只是一个常规组件。
* UserSettings 是一个视图组件。
* UserEmailsSubscriptions、UserProfile、UserProfilePreview 是嵌套的视图组件。

UserSettings 组件的 <template> 部分应该是类似下面的这段代码:

<!-- UserSettings.vue -->

<div>

<h1>User Settings</h1>

<NavBar />

<router-view />

<router-view name="helper" />

</div>

可以通过这个路由配置来实现上面的布局：

{

path: '/settings',

// 你也可以在顶级路由就配置命名视图

component: UserSettings,

children: [{

path: 'emails',

component: UserEmailsSubscriptions

}, {

path: 'profile',

components: {

default: UserProfile,

helper: UserProfilePreview

}

}]

}

**重定向**

重定向也是通过 routes 配置来完成，下面例子是从 /home 重定向到 /：

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

重定向的目标也可以是一个命名的路由：

const routes = [{ path: '/home', redirect: { name: 'homepage' } }]

甚至是一个方法，动态返回重定向目标：

const routes = [

{

// /search/screens -> /search?q=screens

path: '/search/:searchText',

redirect: to => {

// 方法接收目标路由作为参数

// return 重定向的字符串路径/路径对象

return { path: '/search', query: { q: to.params.searchText } }

},

},

{

path: '/search',

// ...

},

]

注意！！在写 redirect 的时候，可以省略 component 配置，因为它从来没有被直接访问过，所以没有组件要渲染。比如：

const routes = [

{ path: '/', redirect: '/home' }, // 根路径自动跳到 /home

{ path: '/home', component: HomeView } //component已经在/home中定义了

];

唯一的例外是嵌套路由：如果一个路由记录有 children 和 redirect 属性，它也应该有 component 属性。比如：

{

path: '/parent',

component: ParentLayout, // ❗这里必须有

redirect: '/parent/child',

children: [

{

path: 'child',

component: ChildView

}

]

}

原因：

* 如果你写了 children，Vue Router 会尝试在 <router-view> 中渲染这些子路由。
* 但如果你没有提供 component，那 Vue Router 不知道该在哪个模板中挂载子路由。
* 这样就会报错或者渲染失败。

**别名（alias）**

当用户访问“别名路径”时，页面显示内容与原路径一样，但 URL 不变。

用途：

* 给一个页面设置多个访问入口
* 适用于兼容旧链接、不同语言URL 等需求

const routes = [

{ path: '/main', component: MainPage, alias: '/index' }

];

注意：用户访问 /index，页面显示的是 /main 的内容，但**地址栏仍显示 /index**。

重定向 和 别名

| **功能** | **redirect** | **alias** |
| --- | --- | --- |
| 作用 | 自动跳转到其他路由 | 提供当前路由的另一种路径 |
| URL 是否变化 | ✅ 变化 | ❌ 不变 |
| 使用场景 | 默认跳转、历史路径兼容等 | 多语言路径、SEO别名、简写路径等 |

**将 props 传递给路由组件**

Vue Router 中，可以通过 props 属性将参数以 prop 的形式传递给路由组件，这样做的好处是：

* 组件更清晰、更易复用
* 不必在组件内部依赖 $route，提高组件的独立性

**三种方式传参：**

1. props: true 将 route.params 作为 props 传入组件

{

path: '/user/:id',

component: User,

props: true

}

<!-- 组件中可以这样写 -->

<template>

<div>用户ID：{{ id }}</div>

</template>

<script setup>

defineProps(['id'])

</script>

1. props: {} 静态对象方式

{

path: '/hello',

component: Hello,

props: { name: 'Vue', age: 5 }

}

组件中会收到 name 和 age 作为 props。

1. props: (route) => ({}) 函数模式（更灵活）

适合处理查询参数、多个 param 等：

{

path: '/user/:id',

component: User,

props: route => ({ id: route.params.id, queryId: route.query.id })

}

这样你就可以把 params 和 query 都自由组合成 props 传入。