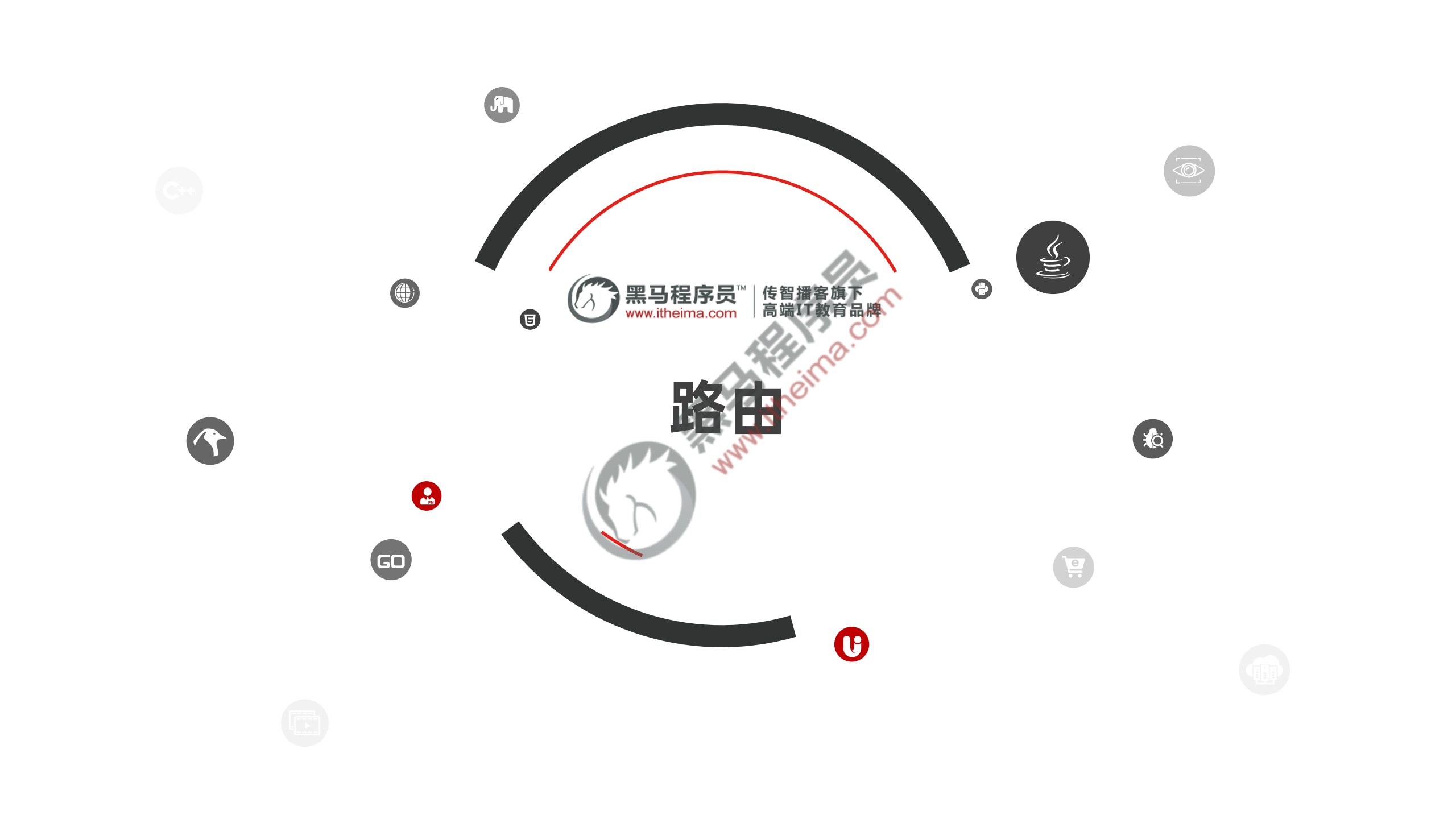

黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
路由
www.itheima.com

目录 Contents

前端路由的概念与原理

1. 什么是路由

路由（英文：router）就是对应关系。路由分为两大类：

① 后端路由

② 前端路由

前端路由的概念与原理

2. 回顾：后端路由

后端路由指的是：请求方式、请求地址与 function 处理函数之间的对应关系。在 node.js 课程中，express 路由的基本用法如下：

```
1 const express = require('express')
2 const router = express.Router()
3
4 router.get('/userlist', function(req, res) { /* 路由的处理函数 */ })
5 router.post('/adduser', function(req, res) { /* 路由的处理函数 */ })
6
7 module.exports = router
```

3. SPA 与前端路由

SPA 指的是一个 web 网站只有唯一的一个 HTML 页面，所有组件的展示与切换都在这唯一的一个页面内完成。此时，不同组件之间的切换需要通过前端路由来实现。

结论：在 SPA 项目中，不同功能之间的切换，要依赖于前端路由来完成！

4. 什么是前端路由

通俗易懂的概念：Hash 地址与组件之间的对应关系。

前端路由的概念与原理

5. 前端路由的工作方式

① 用户点击了页面上的路由链接

② 导致了 URL 地址栏中的 Hash 值发生了变化

③ 前端路由监听了到 Hash 地址的变化

④ 前端路由把当前 Hash 地址对应的组件渲染都浏览器中

结论：前端路由，指的是 Hash 地址与组件之间的对应关系！

6. 实现简易的前端路由

步骤1：导入并注册 MyHome、MyMovie、MyAbout 三个组件。示例代码如下：

```
import MyHome from './components/MyHome.vue'
import MyMovie from './components/MyMovie.vue'
import MyAbout from './components/MyAbout.vue'

export default {
    components: {
        MyHome,
        MyMovie,
        MyAbout,
    },
}
```

6. 实现简易的前端路由

步骤2：通过 <component> 标签的 is 属性，动态切换要显示的组件。示例代码如下：

```vue
<template>
  <h1>App 组件</h1>

  <component :is="comName"></component>
</template>

export default {
  data() {
    return {
      comName: 'my-home', // 要展示的组件的名称
    }
  },
}
```

6. 实现简易的前端路由

步骤3：在组件的结构中声明如下 3 个 <a> 链接，通过点击不同的 <a> 链接，切换浏览器地址栏中的 Hash 值：

```
<a href="#/home">Home</a> 
<a href="#/movie">Movie</a> 
<a href="#/about">About</a>
```

6. 实现简易的前端路由

步骤4：在 created 生命周期函数中监听浏览器地址栏中 Hash 地址的变化，动态切换要展示的组件的名称：

```
created() {
    window.onhashchange = () => {
      switch (location.hash) {
        case '#/home': // 点击了“首页”的链接
          this.comName = 'my-home'
          break
        case '#/movie': // 点击了“电影”的链接
          this.comName = 'my-movie'
          break
        case '#/about': // 点击了“关于”的链接
          this.comName = 'my-about'
          break
      }
    }
},
```

目录 Contents

vue-router 的基本用法

1. 什么是 vue-router

vue-router 是 vue.js 官方给出的路由解决方案。它只能结合 vue 项目进行使用，能够轻松的管理 SPA 项目中组件的切换。

2. vue-router 的版本

vue-router 目前有 3.x 的版本和 4.x 的版本。其中：

- vue-router 3.x 只能结合 vue2 进行使用
- vue-router 4.x 只能结合 vue3 进行使用

vue-router 3.x 的官方文档地址：https://router.vuejs.org/zh/

vue-router 4.x 的官方文档地址：https://next.router.vuejs.org/

vue-router 的基本用法

3. vue-router 4.x 的基本使用步骤

① 在项目中安装 vue-router

② 定义路由组件

③ 声明路由链接和占位符

④ 创建路由模块

⑤ 导入并挂载路由模块

vue-router 的基本用法

3.1 在项目中安装 vue-router

在 vue3 的项目中，只能安装并使用 vue-router 4.x。安装的命令如下：

```
npm install vue-router@next -S
```

vue-router 的基本用法

3.2 定义路由组件

在项目中定义 MyHome.vue、MyMovie.vue、MyAbout.vue 三个组件，将来要使用 vue-router 来控制它们的展示与切换：

MyHome.vue

```vue
<template>
  <h3>MyHome组件</h3>
</template>

<script>
export default {
  name: 'MyHome',
}
</script>

<style>
</style>
```

MyMovie.vue

```vue
<template>
  <h3>MyMovie组件</h3>
</template>

<script>
export default {
  name: 'MyMovie',
}
</script>

<style>
</style>
```

MyAbout.vue

```vue
<template>
  <h3>MyAbout组件</h3>
</template>

<script>
export default {
  name: 'MyAbout',
}
</script>

<style>
</style>
```

3.3 声明路由链接和占位符

可以使用 <router-link> 标签来声明路由链接，并使用 <router-view> 标签来声明路由占位符。示例代码如下：

```
<template>
  <h1>App 组件</h1>
  <!-- 声明路由链接 -->
  <router-link to="/home">首页</router-link> 
  <router-link to="/movie">电影</router-link> 
  <router-link to="/about">关于</router-link>

  <!-- 声明路由占位符 -->
  <router-view></router-view>
</template>
```

3.4 创建路由模块

在项目中创建 router.js 路由模块，在其中按照如下 4 个步骤创建并得到路由的实例对象：

① 从 vue-router 中按需导入两个方法

② 导入需要使用路由控制的组件

③ 创建路由实例对象

④ 向外共享路由实例对象

⑤ 在 main.js 中导入并挂载路由模块

vue-router 的基本用法

从 vue-router 中按需导入两个方法

```
// 1. 从 vue-router 中按需导入两个方法
//    createRouter 方法用于创建路由的实例对象
//    createWebHashHistory 用于指定路由的工作模式（hash 模式）
import { createRouter, createWebHashHistory } from 'vue-router'
```

vue-router 的基本用法

导入需要使用路由控制的组件

```
// 2. 导入组件，这些组件将要以路由的方式，来控制它们的切换
import Home from './components/MyHome.vue'
import Movie from './components/MyMovie.vue'
import About from './components/MyAbout.vue'
```

vue-router 的基本用法

创建路由实例对象

```
// 3. 创建路由实例对象
const router = createRouter({
  // 3.1 通过 history 属性指定路由的工作模式
  history: createWebHashHistory(),
  // 3.2 通过 routes 数组，指定路由规则
  routes: [
    // path 是 hash 地址，component 是要展示的组件
    { path: '/home', component: Home },
    { path: '/movie', component: Movie },
    { path: '/about', component: About },
  ],
})
```

vue-router 的基本用法

向外共享路由实例对象

```
// 4. 向外共享路由实例对象，
//    供其它模块导入并使用
export default router
```

vue-router 的基本用法

在 main.js 中导入并挂载路由模块

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import './index.css'
4 // 1. 导入路由模块
5 import router from './router'
6
7 const app = createApp(App)
8
9 // 2. 挂载路由模块
10 //    app.use() 方法用来挂载“第三方的插件模块”
11 app.use(router)
12
13 app.mount('#app')
```

目录 Contents

vue-router 的高级用法

1. 路由重定向

路由重定向指的是：用户在访问地址 A 的时候，强制用户跳转到地址 C，从而展示特定的组件页面。

通过路由规则的 redirect 属性，指定一个新的路由地址，可以很方便地设置路由的重定向：

```
const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    // 其中，path 表示需要被重定向的“原地址”，redirect 表示将要被重定向到的“新地址”
    { path: '/', redirect: '/home' },
    { path: '/home', component: Home },
    { path: '/movie', component: Movie },
    { path: '/about', component: About },
  ],
})
```

2. 路由高亮

可以通过如下的两种方式，将激活的路由链接进行高亮显示：

① 使用默认的高亮 class 类

② 自定义路由高亮的 class 类

2.1 默认的高亮 class 类

被激活的路由链接，默认会应用一个叫做 router-link-active 的类名。开发者可以使用此类名选择器，为激活的路由链接设置高亮的样式：

```css
/* 在 index.css 全局样式表中，重新 router-link-active 的样式 */
.router-link-active {
  background-color: red;
  color: white;
  font-weight: bold;
}
```

vue-router 的高级用法

2.2 自定义路由高亮的 class 类

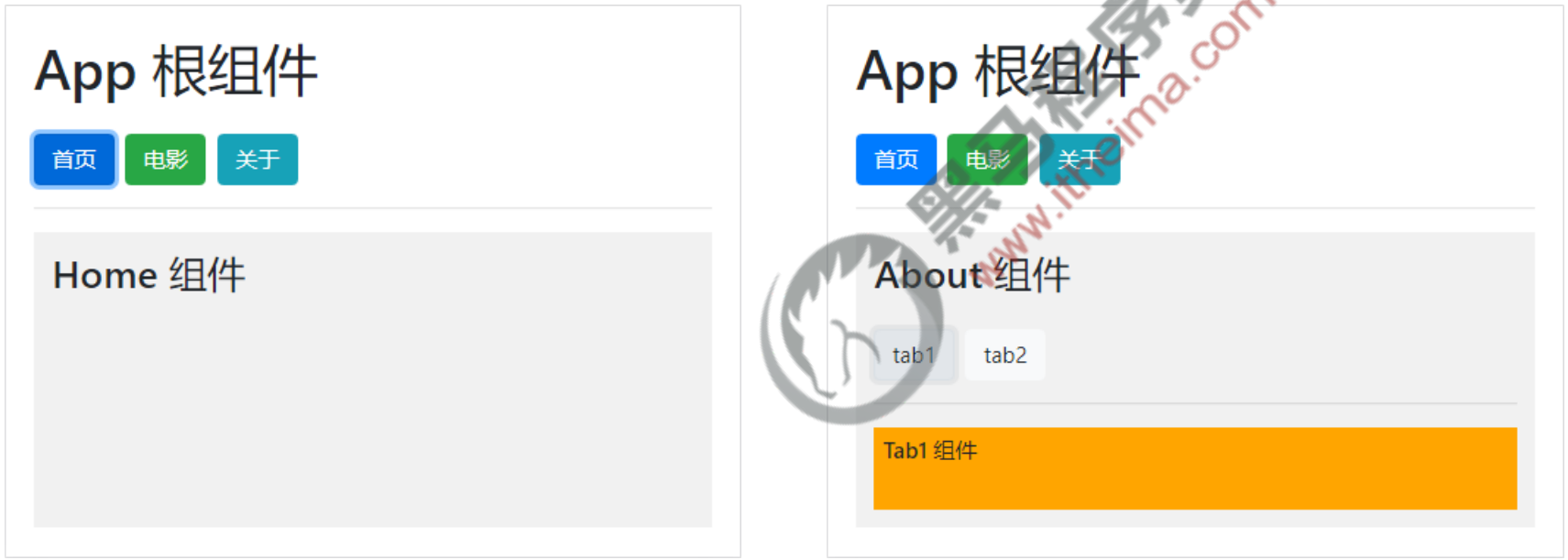
在创建路由的实例对象时，开发者可以基于 linkActiveClass 属性，自定义路由链接被激活时所应用的类名：

```
const router = createRouter({
  history: createWebHashHistory(),
  // 指定被激活的路由链接，会应用 router-active 这个类名，
  // 默认的 router-link-active 类名会被覆盖掉
  linkActiveClass: 'router-active',
  routes: [
    { path: '/', redirect: '/home' },
    { path: '/home', component: Home },
    { path: '/movie', component: Movie },
    { path: '/about', component: About },
  ],
})
```

vue-router 的高级用法

3. 嵌套路由

通过路由实现组件的嵌套展示，叫做嵌套路由。



① 声明子路由链接和子路由占位符

② 在父路由规则中，通过 children 属性嵌套声明子路由规则

3.1 声明子路由链接和子路由占位符

在 About.vue 组件中，声明 tab1 和 tab2 的子路由链接以及子路由占位符。示例代码如下：

```vue
<template>
  <h3>MyAbout组件</h3>
  <!-- 在关于页面中，声明两个子路由链接 -->
  <router-link to="/about/tab1">tab1</router-link> 
  <router-link to="/about/tab2">tab2</router-link>

  <!-- 在关于页面中，声明 tab1 和 tab2 的路由占位符 -->
  <router-view></router-view>
</template>
```

3.2 通过 children 属性声明子路由规则

在 router.js 路由模块中，导入需要的组件，并使用 children 属性声明子路由规则。示例代码如下：

```
import Tab1 from './components/tabs/MyTab1.vue'
import Tab2 from './components/tabs/MyTab2.vue'

const router = createRouter({
  routes: [
    { // about 页面的路由规则（父级路由规则）
      path: '/about',
      component: About,
      children: [ // 通过 children 属性嵌套子级路由规则
        { path: 'tab1', component: Tab1 }, // 访问 /about/tab1 时，展示 Tab1 组件
        { path: 'tab2', component: Tab2 }, // 访问 /about/tab2 时，展示 Tab2 组件
      ],
    },
  ],
})
```

注意：子路由规则的 path 不要以 / 开头！

4. 动态路由匹配

思考：有如下 3 个路由链接：

```
<router-link to="/movie/1">电影1</router-link>
<router-link to="/movie/2">电影2</router-link>
<router-link to="/movie/3">电影3</router-link>
```

定义如下 3 个路由规则，是否可行???

```
{ path: '/movie/1', component: Movie }
{ path: '/movie/2', component: Movie }
{ path: '/movie/3', component: Movie }
```

缺点：路由规则的复用性差。

4.1 动态路由的概念

动态路由指的是：把 Hash 地址中可变的部分定义为参数项，从而提高路由规则的复用性。在 vue-router 中使用英文的冒号（:）来定义路由的参数项。示例代码如下：

```
// 路由中的动态参数以 : 进行声明，冒号后面的是动态参数的名称
{ path: '/movie/:id', component: Movie }

// 将以下 3 个路由规则，合并成了一个，提高了路由规则的复用性
{ path: '/movie/1', component: Movie }
{ path: '/movie/2', component: Movie }
{ path: '/movie/3', component: Movie }
```

4.2 $route.params 参数对象

通过动态路由匹配的方式渲染出来的组件中，可以使用 $route.params 对象访问到动态匹配的参数值。

```vue
<template>
  <!-- $route.params 是路由的“参数对象” -->
  <h3>MyMovie组件 --- {{$route.params.id}}</h3>
</template>

<script>
export default {
  name: 'MyMovie',
}
</script>
```

4.3 使用 props 接收路由参数

为了简化路由参数的获取形式，vue-router 允许在路由规则中开启 props 传参。示例代码如下：

```
// 1、在定义路由规则时，声明 props: true 选项，
//    即可在 Movie 组件中，以 props 的形式接收到路由规则匹配到的参数项
{ path: '/movie/:id', component: Movie, props: true}

<template>
  <!-- 3、直接使用 props 中接收的路由参数 -->
  <h3>MyMovie组件 --- {{id}}</h3>
</template>

<script>
export default {
  props: ['id'] // 2、使用 props 接收路由规则中匹配到的参数项
}
</script>
```

5. 编程式导航

通过调用 API 实现导航的方式，叫做编程式导航。与之对应的，通过点击链接实现导航的方式，叫做声明式导航。例如：

- 普通网页中点击 <a> 链接、vue 项目中点击 <router-link> 都属于声明式导航
- 普通网页中调用 location.href 跳转到新页面的方式，属于编程式导航

vue-router 的高级用法

5.1 vue-router 中的编程式导航 API

vue-router 提供了许多编程式导航的 API，其中最常用的两个 API 分别是：

① this.$router.push('hash 地址')

- 跳转到指定 Hash 地址，从而展示对应的组件

② this.$router.go(数值 n)

- 实现导航历史的前进、后退

5.2 $router.push

调用 this.$router.push() 方法，可以跳转到指定的 hash 地址，从而展示对应的组件页面。示例代码如下：

```
<template>
  <h3>MyHome组件</h3>
  <button @click="gotoMovie(3)">go to Movie</button>
</template>

<script>
export default {
  methods: {
    gotoMovie(id) { // id 参数是电影的 id 值
      this.$router.push(`/movie/${id}`)
    },
  },
}
</script>
```

5.2 $router.go

调用 this.$router.go() 方法，可以在浏览历史中进行前进和后退。示例代码如下：

```
<template>
  <h3>MyMovie组件 --- {{id}}</h3>
  <button @click="goBack">后退</button>
</template>

<script>
export default {
  props: ['id'],
  methods: {
    goBack() { this.$router.go(-1) } // 后退到之前的组件页面
  },
}
</script>
```

6. 命名路由

通过 name 属性为路由规则定义名称的方式，叫做命名路由。示例代码如下：

```
{
    path: '/movie/:id',
    // 使用 name 属性为当前的路由规则定义一个“名称”
    name: 'mov',
    component: Movie,
    props: true,
}
```

注意：命名路由的 name 值不能重复，必须保证唯一性！

vue-router 的高级用法

6.1 使用命名路由实现声明式导航

为 <router-link> 标签动态绑定 to 属性的值，并通过 name 属性指定要跳转到的路由规则。期间还可以用 params 属性指定跳转期间要携带的路由参数。示例代码 如下：

```
<template>
  <h3>MyHome组件</h3>
  <router-link :to="{ name: 'mov', params: { id: 3 } }">go to Movie</router-link>
</template>

<script>
export default {
  name: 'MyHome',
}
</script>
```

6.2 使用命名路由实现编程式导航

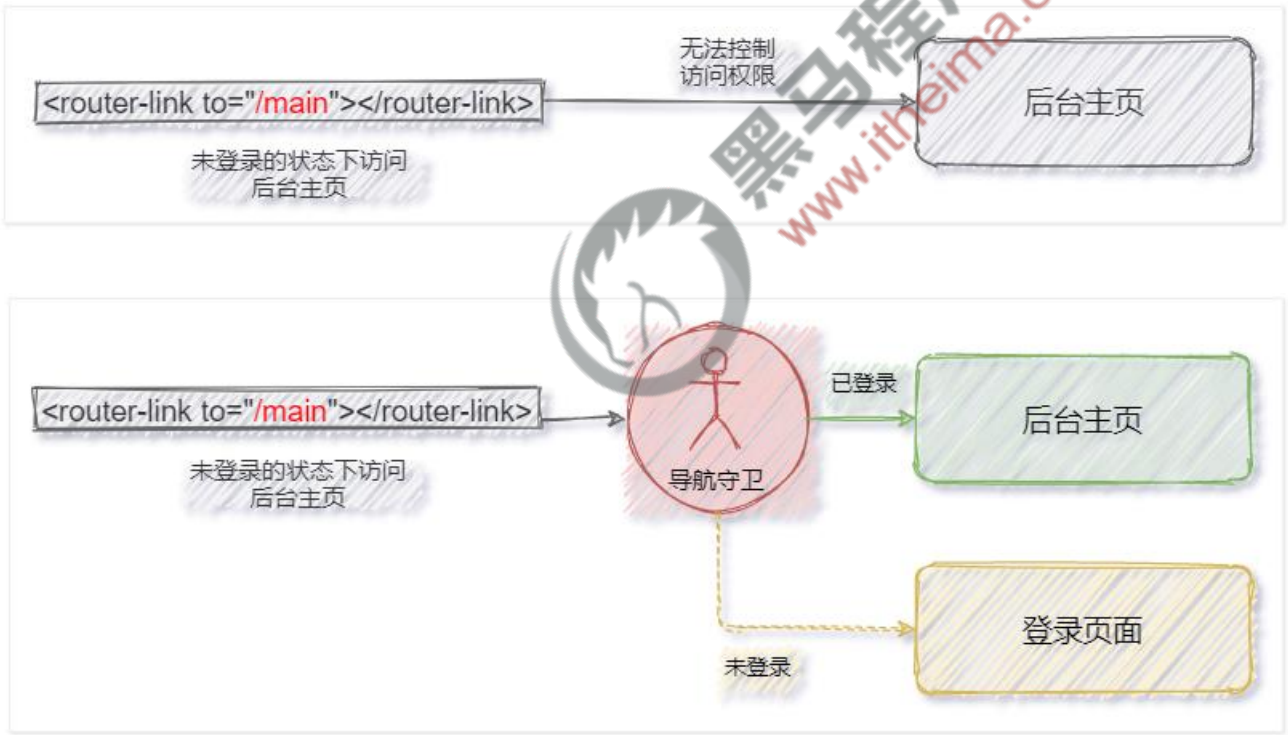
调用 push 函数期间指定一个配置对象，name 是要跳转到的路由规则、params 是携带的路由参数：

```
<template>
  <h3>MyHome组件</h3>
  <button @click="gotoMovie(3)">go to Movie</button>
</template>

<script>
export default {
  methods: {
    gotoMovie(id) {
      this.$router.push({ name: 'mov', params: { id: 3 } })
    },
  },
}
</script>
```

7. 导航守卫

导航守卫可以控制路由的访问权限。示意图如下：



vue-router 的高级用法

7.1 如何声明全局导航守卫

全局导航守卫会拦截每个路由规则，从而对每个路由进行访问权限的控制。可以按照如下的方式定义全局导航守卫：

```
// 创建路由实例对象
const router = createRouter({ ... })

// 调用路由实例对象的 beforeEach 函数，声明"全局前置守卫"
// fn 必须是一个函数，每次拦截到路由的请求，都会调用 fn 进行处理
// 因此 fn 叫做 "守卫方法"
router.beforeEach(fn)
```

7.2 守卫方法的 3 个形参

全局导航守卫的守卫方法中接收 3 个形参，格式为：

```
// 创建路由实例对象
const router = createRouter({ ... })

// 全局前置守卫
router.beforeEach((to, from, next) => {
    // to 目标路由对象
    // from 当前导航正要离开的路由对象
    // next 是一个函数，表示放行
})
```

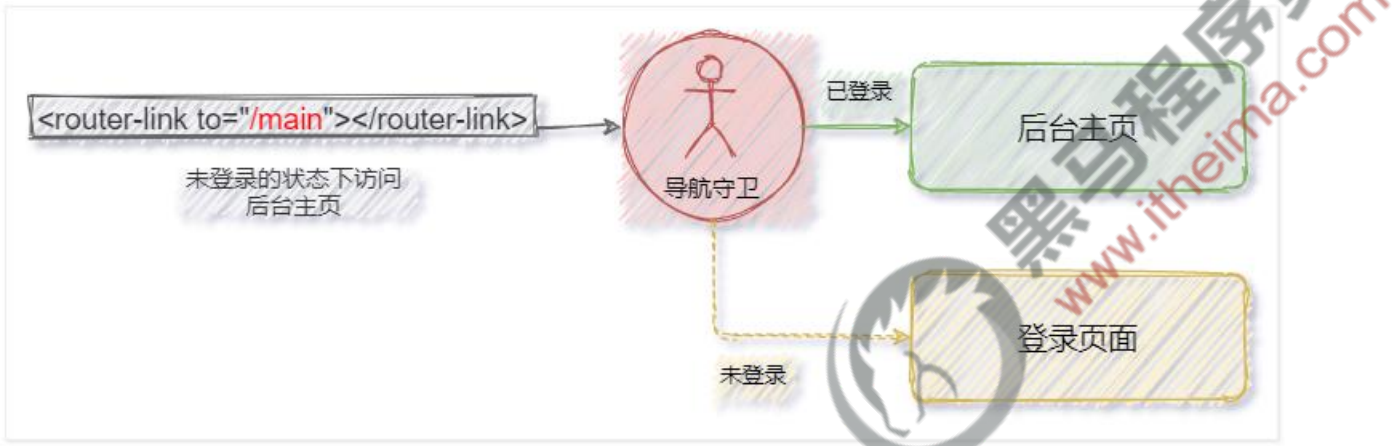
注意：

① 在守卫方法中如果不声明 next 形参，则默认允许用户访问每一个路由！

② 在守卫方法中如果声明了 next 形参，则必须调用 next() 函数，否则不允许用户访问任何一个路由！

7.3 next 函数的 3 种调用方式

参考示意图，分析 next 函数的 3 种调用方式最终导致的结果：



直接放行：next()

强制其停留在当前页面：next(false)

强制其跳转到登录页面：next('/login')

7.4 结合 token 控制后台主页的访问权限

```
// 全局前置守卫
router.beforeEach((to, from, next) => {
  const token = localStorage.getItem('token') // 1. 读取 token
  if (to.path === '/main' && !token) {        // 2. 想要访问“后台主页”且 token 值不存在
    // next(false) // 3.1 不允许跳转
    next('/login') // 3.2 强制跳转到“登录页面”
  } else {
    next()         // 3.3 直接放行，允许访问“后台主页”
  }
})
```

目录 Contents

后台管理案例

1. 案例效果

后台管理案例

1. 案例效果

2. 案例用到的知识点

- 命名路由
- 路由重定向
- 导航守卫
- 嵌套路由
- 动态路由匹配
- 编程式导航

① 能够知道如何在 vue 中配置路由

- createRouter、app.use(router)

② 能够知道如何使用嵌套路由

- 通过 children 属性进行路由嵌套、子路由的 hash 地址不要以 / 开头

③ 能够知道如何实现动态路由匹配

- 使用冒号声明参数项、this.$route.params、props: true

④ 能够知道如何使用编程式导航

- this.$router.push、this.$router.go(-1)

⑤ 能够知道如何使用全局导航守卫

- 路由实例.beforeEach((to, from, next) => { })

黑马程序员

www.itheima.com

传智播客旗下高端IT教育品牌