

# 录Contents

- ◆ 前端路由的概念与原理
- ◆ vue-router 的基本使用
- ◆ vue-router 的常见用法
- ◆ 后台管理案例

# ■ 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。

# ■ 前端路由的概念与原理

## 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！

# ■ 前端路由的概念与原理

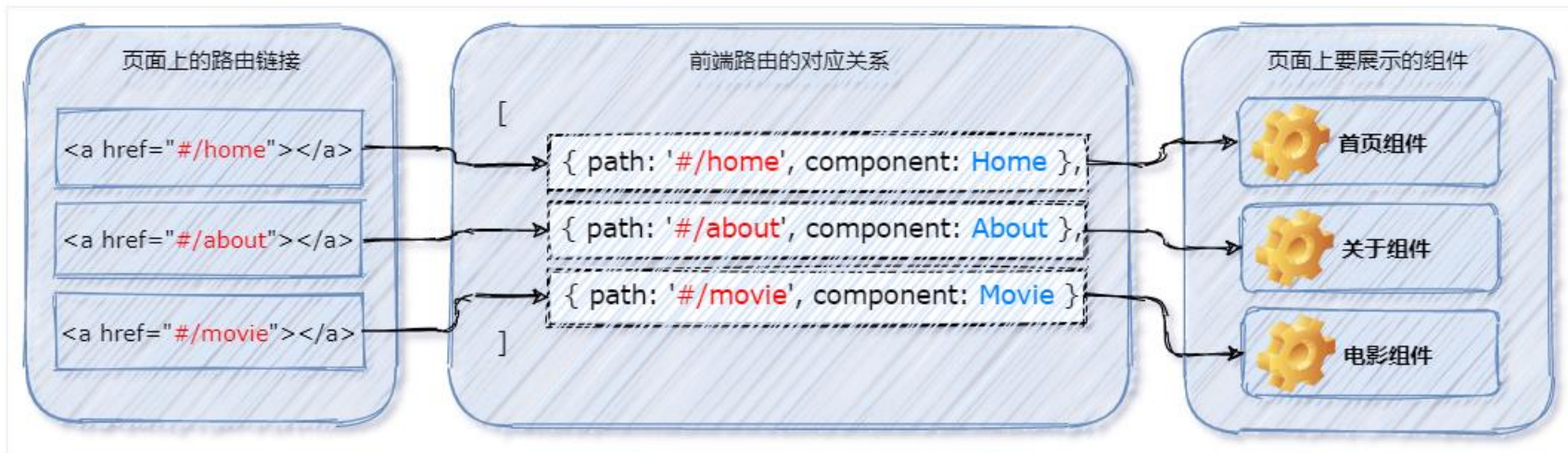
## 4. 什么是前端路由

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** 渲染到浏览器中



结论：前端路由，指的是 **Hash 地址** 与 **组件之间的对应关系**！

# ■ 前端路由的概念与原理

## 6. 实现简易的前端路由

步骤1：通过 `<component>` 标签，结合 `comName` 动态渲染组件。示例代码如下：

```
1 <!-- 通过 is 属性，指定要展示的组件的名称 -->
2 <component :is="comName"></component>
3
4 export default {
5   name: 'App',
6   data() {
7     return {
8       // 要展示的组件的名称
9       comName: 'Home'
10    }
11  }
12 }
```

# ■ 前端路由的概念与原理

## 6. 实现简易的前端路由

步骤2：在 App.vue 组件中，为 `<a>` 链接添加对应的 hash 值：

```
1 <a href="#/home">Home</a>&nbsp;
2 <a href="#/movie">Movie</a>&nbsp;
3 <a href="#/about">About</a>
```




# ■ 前端路由的概念与原理

## 6. 实现简易的前端路由

步骤3：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：

```
1 created() {  
2   window.onhashchange = () => {  
3     switch (location.hash) {  
4       case '#/home': // 点击了“首页”的链接  
5         this.comName = 'Home'  
6         break  
7       case '#/movie': // 点击了“电影”的链接  
8         this.comName = 'Movie'  
9         break  
10      case '#/about': // 点击了“关于”的链接  
11        this.comName = 'About'  
12        break  
13      }  
14    }  
15 }
```



# 录Contents

- ◆ 前端路由的概念与原理
- ◆ vue-router 的基本使用
- ◆ vue-router 的常见用法
- ◆ 后台管理案例

# 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。

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

# vue-router 的基本用法

## 2. vue-router 安装和配置的步骤

- ① 安装 vue-router 包
- ② 创建路由模块
- ③ 导入并挂载路由模块
- ④ 声明路由链接和占位符

# vue-router 的基本用法

## 2.1 在项目中安装 vue-router

在 vue2 的项目中，安装 vue-router 的命令如下：

```
1 npm i vue-router@3.5.2 -S
```

# vue-router 的基本用法

## 2.2 创建路由模块

在 `src` 源代码目录下，新建 `router/index.js` 路由模块，并初始化如下的代码：

```
1 // 1. 导入 Vue 和 VueRouter 的包
2 import Vue from 'vue'
3 import VueRouter from 'vue-router'
4
5 // 2. 调用 Vue.use() 函数，把 VueRouter 安装为 Vue 的插件
6 Vue.use(VueRouter)
7
8 // 3. 创建路由的实例对象
9 const router = new VueRouter()
10
11 // 4. 向外共享路由的实例对象
12 export default router
```

# vue-router 的基本用法

## 2.3 导入并挂载路由模块

在 src/main.js 入口文件中，导入并挂载路由模块。示例代码如下：

```
1 import Vue from 'vue'
2 import App from './App.vue'
3 // 1. 导入路由模块
4 import router from '@router'
5
6 new Vue({
7   render: h => h(App),
8   // 2. 挂载路由模块
9   router: router
10 }).$mount('#app')
```

# vue-router 的基本用法

## 2.4 声明路由链接和占位符

在 src/App.vue 组件中，使用 vue-router 提供的 `<router-link>` 和 `<router-view>` 声明路由链接和占位符：

```
1 <template>
2   <div class="app-container">
3     <h1>App 组件</h1>
4     <!-- 1. 定义路由链接 -->
5     <router-link to="/home">首页</router-link>
6     <router-link to="/movie">电影</router-link>
7     <router-link to="/about">关于</router-link>
8
9     <hr />
10
11    <!-- 2. 定义路由的占位符 -->
12    <router-view></router-view>
13  </div>
14 </template>
```




# vue-router 的基本用法

## 3. 声明路由的匹配规则

在 src/router/index.js 路由模块中，通过 **routes 数组** 声明路由的匹配规则。示例代码如下：

```
1 // 导入需要使用路由切换展示的组件
2 import Home from '@/components/Home.vue'
3 import Movie from '@/components/Movie.vue'
4 import About from '@/components/About.vue'
5
6 // 2. 创建路由的实例对象
7 const router = new VueRouter({
8   routes: [ // 在 routes 数组中，声明路由的匹配规则
9     // path 表示要匹配的 hash 地址； component 表示要展示的路由组件
10    { path: '/home', component: Home },
11    { path: '/movie', component: Movie },
12    { path: '/about', component: About }
13  ]
14 })
```



# 录Contents

- ◆ 前端路由的概念与原理
- ◆ vue-router 的基本使
- ◆ vue-router 的常见用法
- ◆ 后台管理案例

# vue-router 的常见用法

## 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。

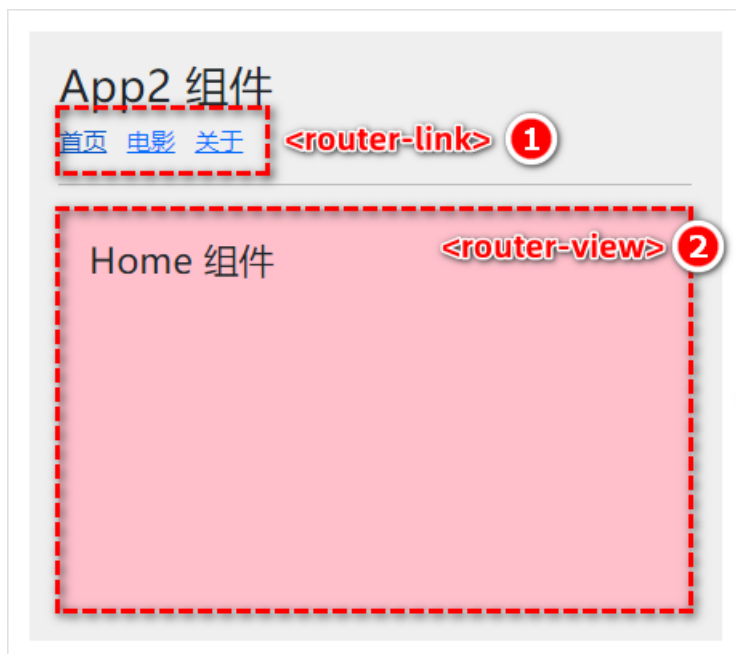
通过路由规则的 `redirect` 属性，指定一个新的路由地址，可以很方便地设置路由的重定向：

```
1 const router = new VueRouter({
2   // 在 routes 数组中，声明路由的匹配规则
3   routes: [
4     // 当用户访问 / 的时候，通过 redirect 属性跳转到 /home 对应的路由规则
5     { path: '/', redirect: '/home' },
6     { path: '/home', component: Home },
7     { path: '/movie', component: Movie },
8     { path: '/about', component: About }
9   ]
10 })
```

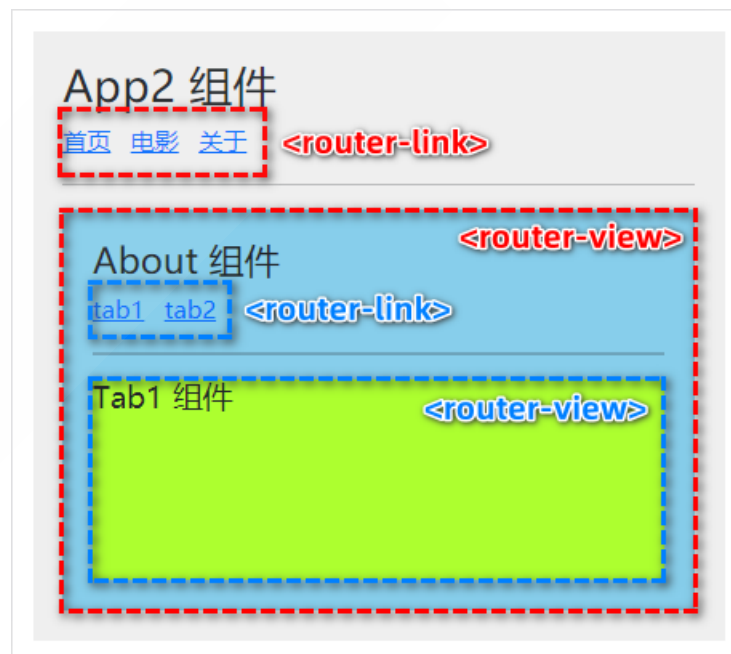
# vue-router 的常见用法

## 2. 嵌套路由

通过路由实现**组件的嵌套展示**，叫做嵌套路由。



点击**父级路由链接**显示模板内容



① 模板内容中又有**子级路由链接**

② 点击**子级路由链接**显示子级模板内容

# vue-router 的常见用法

## 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：

```
1 <template>
2   <div class="about-container">
3     <h3>About 组件</h3>
4     <!-- 1. 在关于页面中，声明两个子路由链接 -->
5     <router-link to="/about/tab1">tab1</router-link>
6     <router-link to="/about/tab2">tab2</router-link>
7
8     <hr />
9
10    <!-- 2. 在关于页面中，声明子路由的占位符 -->
11    <router-view></router-view>
12  </div>
13 </template>
```

# vue-router 的常见用法

## 3.2 通过 **children** 属性声明子路由规则

在 `src/router/index.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：

```
1 import Tab1 from '@/components/tabs/Tab1.vue'
2 import Tab2 from '@/components/tabs/Tab2.vue'
3
4 const router = new VueRouter({
5   routes: [
6     { // about 页面的路由规则（父级路由规则）
7       path: '/about',
8       component: About,
9       children: [ // 1. 通过 children 属性，嵌套声明子级路由规则
10         { path: 'tab1', component: Tab1 }, // 2. 访问 /about/tab1 时，展示 Tab1 组件
11         { path: 'tab2', component: Tab2 } // 2. 访问 /about/tab2 时，展示 Tab2 组件
12       ]
13     }
14   ]
15 })
```

# vue-router 的常见用法

## 4. 动态路由匹配

思考：有如下 3 个路由链接：

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

定义如下 3 个路由规则，是否可行???

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

缺点：路由规则的复用性差。

# vue-router 的常见用法

## 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：

```
1 // 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
2 { path: '/movie/:id', component: Movie }
3
4 // 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
5 { path: '/movie/1', component: Movie }
6 { path: '/movie/2', component: Movie }
7 { path: '/movie/3', component: Movie }
```



# vue-router 的常见用法

## 4.2 `$route.params` 参数对象

在动态路由渲染出来的组件中，可以使用 `this.$route.params` 对象访问到动态匹配的参数值。

```
1 <template>
2   <div class="movie-container">
3     <!-- this.$route 是路由的“参数对象” -->
4     <h3>Movie 组件 -- {{ this.$route.params.id }}</h3>
5   </div>
6 </template>
7
8 <script>
9 export default {
10   name: 'Movie'
11 }
12 </script>
```

# vue-router 的常见用法

## 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 // 1、在定义路由规则时，声明 props: true 选项，
2 //    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
3 { path: '/movie/:id', component: Movie, props: true }
4
5 <template>
6   <!-- 3、直接使用 props 中接收的路由参数 -->
7   <h3>MyMovie组件 --- {{id}}</h3>
8 </template>
9
10 <script>
11 export default {
12   props: ['id'] // 2、使用 props 接收路由规则中匹配到的参数项
13 }
14 </script>
```

# vue-router 的常见用法

## 5. 声明式导航 & 编程式导航

在浏览器中，**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

在浏览器中，**调用 API 方法**实现导航的方式，叫做**编程式导航**。例如：

- 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 的常见用法

## 5.1 vue-router 中的程式化导航 API

vue-router 提供了许多程式化导航的 API，其中最常用的导航 API 分别是：

- ① `this.$router.push('hash 地址')`
  - 跳转到指定 hash 地址，并**增加**一条历史记录
- ② `this.$router.replace('hash 地址')`
  - 跳转到指定的 hash 地址，并**替换掉当前的**历史记录
- ③ `this.$router.go(数值 n)`
  - 实现导航历史前进、后退

# vue-router 的常见用法

## 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：

```
1 <template>
2   <div class="home-container">
3     <h3>Home 组件</h3>
4     <button @click="gotoMovie">跳转到 Movie 页面</button>
5   </div>
6 </template>
7
8 <script>
9 export default {
10   methods: {
11     gotoMovie() { this.$router.push('/movie/1') }
12   }
13 }
14 </script>
```

# vue-router 的常见用法

## 5.3 \$router.replace

调用 `this.$router.replace()` 方法，可以跳转到指定的 hash 地址，从而展示对应的组件页面。

push 和 replace 的区别：

- push 会增加一条历史记录
- replace 不会增加历史记录，而是替换掉当前的历史记录

# vue-router 的常见用法

## 5.4 \$router.go

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

```
1 <template>
2   <h3>MyMovie组件 --- {{id}}</h3>
3   <button @click="goBack">后退</button>
4 </template>
5
6 <script>
7 export default {
8   props: ['id'],
9   methods: {
10     goBack() { this.$router.go(-1) } // 后退到之前的组件页面
11   },
12 }
13 </script>
```

# vue-router 的常见用法

## 5.5 \$router.go 的简化用法

在实际开发中，一般只会前进和后退一层页面。因此 vue-router 提供了如下两个便捷方法：

### ① \$router.back()

- 在历史记录中，**后退**到上一个页面

### ② \$router.forward()

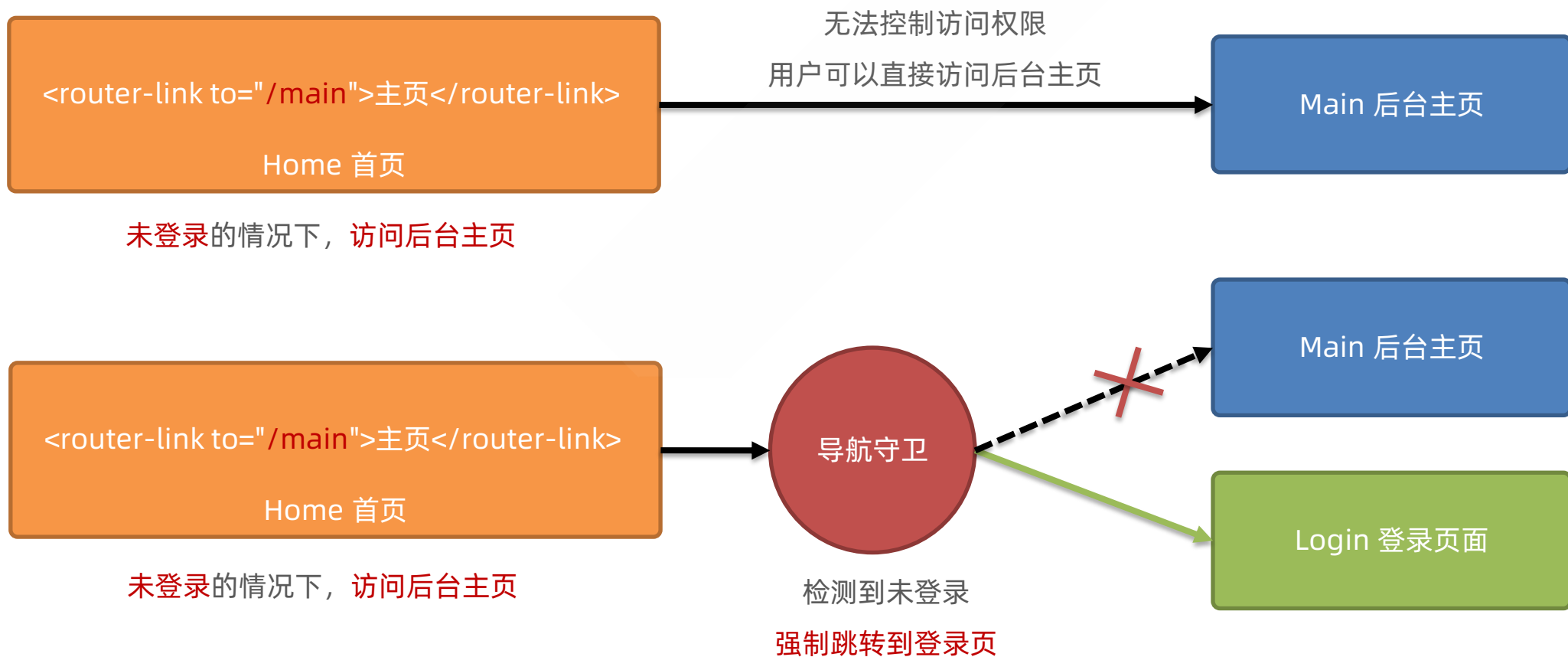
- 在历史记录中，**前进**到下一个页面



# vue-router 的常见用法

## 6. 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 的常见用法

## 6.1 全局前置守卫

每次发生路由的**导航跳转**时，都会触发**全局前置守卫**。因此，在全局前置守卫中，程序员可以对每个路由进行**访问权限**的控制：

```
1 // 创建路由实例对象
2 const router = new VueRouter({ ... })
3
4 // 调用路由实例对象的 beforeEach 方法，即可声明 “全局前置守卫”
5 // 每次发生路由导航跳转的时候，都会自动触发 fn 这个 “回调函数”
6 router.beforeEach(fn)
```

# vue-router 的常见用法

## 6.2 守卫方法的 3 个形参

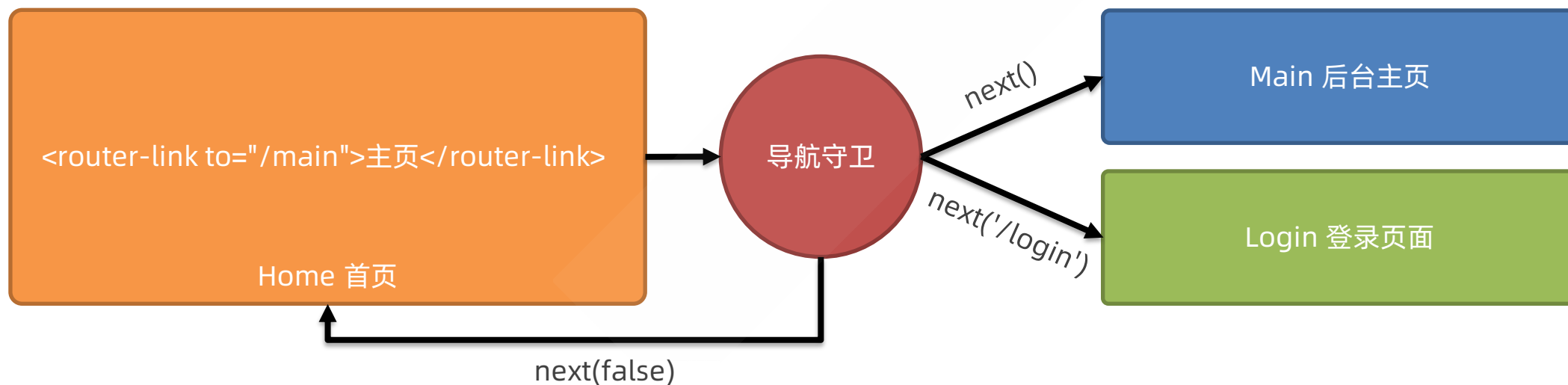
全局前置守卫的回调函数中接收 3 个形参，格式为：

```
1 // 创建路由实例对象
2 const router = new VueRouter({ ... })
3
4 // 全局前置守卫
5 router.beforeEach((to, from, next) => {
6   // to 是将要访问的路由的信息对象
7   // from 是将要离开的路由的信息对象
8   // next 是一个函数，调用 next() 表示放行，允许这次路由导航
9 })
```

# vue-router 的常见用法

## 6.3 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：



当前用户**拥有**后台主页的访问权限，直接放行：next()


当前用户**没有**后台主页的访问权限，**强制其跳转到登录页面**：next('/login')

当前用户**没有**后台主页的访问权限，**不允许跳转到后台主页**：next(false)

# vue-router 的常见用法

## 6.4 控制后台主页的访问权限

```
1 router.beforeEach((to, from, next) => {
2   if (to.path === '/main') {
3     const token = localStorage.getItem('token')
4     if (token) {
5       next() // 访问的是后台主页, 且有 token 的值
6     } else {
7       next('/login') // 访问的是后台主页, 但是没有 token 的值
8     }
9   } else {
10    next() // 访问的不是后台主页, 直接放行
11  }
12 })
```



# 录Contents

- ◆ 前端路由的概念与原理
- ◆ vue-router 的基本使用
- ◆ vue-router 的常见用法
- ◆ 后台管理案例

## 1. 案例效果



## 1. 案例效果

<div> 黑马程序员 黑马后台管理系统</div> <div>www.itheima.com</div> <div>退出登录</div>				
用户管理	用户管理			
权限管理	#	姓名	年龄	操作
商品管理	1	嬴政	18	<a href="#">详情</a>
订单管理	2	李斯	35	<a href="#">详情</a>
系统设置	3	吕不韦	50	<a href="#">详情</a>
	4	赵姬	48	<a href="#">详情</a>



## 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
- 程式化导航



# 总结

- 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
- 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 能够知道如何使用程式导航
  - this.\$router.push、this.\$router.go
- ⑤ 能够知道如何使用导航守卫
  - 路由实例.beforeEach((to, from, next) => { /\* 必须调 next 函数 \*/ })

