

Vue-Router

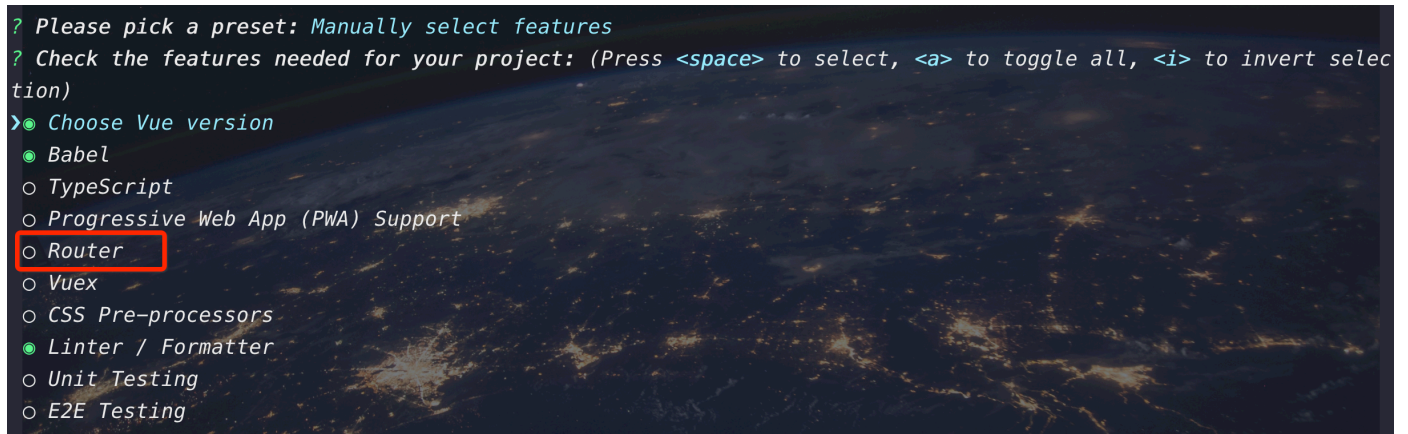
Vue Router是Vue.js官方的路由管理器。它和Vue.js的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

1. 嵌套的路由/视图表
2. 模块化的、基于组件的路由配置
3. 路由参数、查询、通配符
4. 基于 Vue.js 过渡系统的视图过渡效果
5. 带有自动激活的 CSS class 的链接
6. HTML5 历史模式或 hash 模式，在 IE9 中自动降级

1.基础使用

安装

1)可以在搭建脚手架时选择对应的路由选项



```
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> Choose Vue version
  ● Babel
  ○ TypeScript
  ○ Progressive Web App (PWA) Support
  ○ Router
  ○ Vuex
  ○ CSS Pre-processors
  ● Linter / Formatter
  ○ Unit Testing
  ○ E2E Testing
```

2)也可在项目中使用`npm install vue-router --save`进行安装，自己进行配置

使用

```
//引入Vue
import Vue from 'vue'
//引入vue-router
import VueRouter from 'vue-router'
//要显示的页面组件
import Home from '../views/Home.vue'
// 使用vue-router
Vue.use(VueRouter)

//定义路由与页面组件的映射关系
const routes = [
  {
    //路由路径
    path: '/',
    //命名路由
    name: 'Home',
    // 页面组件
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js)
    // for this route
    // which is lazy-loaded when the route is visited.
    // 懒加载路由
  }
]
```

```
      component: () => import(/* webpackChunkName: "about"
*/ './views/About.vue')
    }
  ]

// 创建路由实例对象
const router = new VueRouter({
  // 路由模式
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

// 导出路由
export default router
```

```
//在main.js中进行路由的注册
import Vue from 'vue'
import App from './App.vue'
import router from './router'

Vue.config.productionTip = false

new Vue({
  //在main.js文件中的vue实例中进行路由的注册使用
  router,
  render: h => h(App)
}).$mount('#app')
```

在对应的组件中进行路由的配置使用

```
<template>
  <div id="app">
    <div id="nav">
      <!-- 使用router-link标签进行路由跳转的配置 -->
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
    <!-- 使用router-view标签进行对应页面组件的显示，此标签必须使用，否则页面无法显示 -->
    <router-view/>
  </div>
</template>
```

2.动态路由

需要把某种模式匹配到的所有路由，全部映射到同一个组件。

例如：有一个user组件，对于所有id不同的用户，都要使用这个组件来渲染，那么可以在vue-router的路由路径中使用动态路径参数来达到这个效果。

```
//在router->index.js中进行相关配置
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
```

```
    component: () => import('../views/About.vue')
  },
  //动态路由配置
  {
    path: '/user/:id',
    name: 'User',
    component: () => import('../views/User.vue')
  }
}
```

//在App.vue中使用动态路由

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link> |
      //动态路由需要传递相应的参数
      <router-link to="/user/123">User</router-link>
    </div>
    <router-view />
  </div>
</template>
```

//在User.vue中可以接收动态路由传递的参数

```
<template>
  <div>
    这是用户界面
    //可以接收动态路由传递的参数
    {{ $route.params.id }}
  </div>
</template>
```

3. 嵌套路由

实际生活中的应用界面，通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件：

```
//在router->index.js下配置用户页面的嵌套子路由
{
  path: '/user',
  name: 'User',
  component: () => import('../views/User.vue'),
  // 嵌套路由
  children: [
    // 用户子页面 1
    {
      // 子路由的path只需要填写其自己的路径即可 vue会自动帮我们补全
      // 路径
      path: 'user1',
      name: 'User1',
      component: () => import('../views/User1.vue'),
    },
    // 用户子页面 2
    {
      path: 'user2',
      name: 'User2',
      component: () => import('../views/User2.vue'),
    }
  ]
}
```

```
//在用户界面设置子路由跳转
<template>
  <div>
    这是用户界面
    {{ $route.params.id }}
    <!-- 此处要补全路径 -->
    <router-link to="/user/user1">用户1</router-link>
    <router-link to="/user/user2">用户2</router-link>
    <router-view></router-view>
  </div>
</template>
```

4. 程式化导航

除了使用 `<router-link>` 创建 a 标签来定义导航链接，还可以借助 router 的实例方法，通过编写代码来实现。

`this.$router.push()`

跳转到指定路由，会向history栈添加一个新的记录，当用户点击浏览器后退按钮的时候，可以回到跳转前的url。

```
<template>
  <div id="app">
    <!-- 通过绑定点击事件 在js方法中通过编程的方式进行路由的跳转 -->
    <!-- 这种方式叫作 程式化导航 -->
    <button @click="homeHandler">首页</button>
    <button @click="aboutHandler">关于</button>
    <router-view></router-view>
  </div>
</template>
<script>
```

```
export default {
  methods: {
    homeHandler() {
      this.$router.push({
        path: "/",
      });
    },
    aboutHandler() {
      this.$router.push({
        path: "/about",
      });
    },
  },
};
</script>
```

当使用程式导航时，我们发现，点击用一个按钮的时候控制台会报错，这是路由的机制导致的，解决方案有两种

1.指定vue-router的版本

```
npm i vue-router@3.0 --save
```

2.在 main.js里添加一段代码

```
import Router from 'vue-router'
const routerPush = Router.prototype.push
Router.prototype.push = function push(location) {
  return routerPush.call(this, location).catch(error=> error)
}
```


this.\$router.repalcce()

跟 router.push 很像，唯一的不同就是，它不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录。

this.\$router.go(n)

这个方法的参数是一个整数，意思是在 history 记录中向前或者后退多少步。

```
// 在浏览器记录中前进一步，等同于 history.forward()  
router.go(1)  
// 后退一步记录，等同于 history.back()  
router.go(-1)  
// 前进 3 步记录  
router.go(3)  
// 如果 history 记录不够用，则会报错  
router.go(-100)  
router.go(100)
```

5.命名路由

有时候，通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。你可以在创建 Router 实例的时候，在 routes 配置中给某个路由设置名称。

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

要链接到一个命名路由，可以给 router-link 的 to 属性传一个对象：

```
<router-link :to="{ name: 'user', params: { userId: 123 } }">User</router-link>
```

等价于代码调用 router.push()：

```
router.push({ name: 'user', params: { userId: 123 } })
```

这两种方式都会跳转到/user/123路径。

6.重定向

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

```
const router = new VueRouter({
  routes: [
    {
      path: '/a',
      redirect: '/b',
      // 或者 redirect: { name: 'b' }
      // 或者
      /*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
    }*/
}
]
})

```

“重定向”的意思是，当用户访问 /a 时，URL 将会被替换成 /b，然后匹配路由为 /b。

7.路由组件传参

路由传递是指，从A页面跳转到B页面时，将A页面中的变量传递给B页面使用，传递参数的方式有两种：

path-query传参

使用path与query结合的方式传递参数时，参数会被拼接在浏览器地址栏中，并且刷新页面后数据也不会丢失。

```

<template>
  <div id="app">
    <!-- 通过绑定点击事件 在js方法中通过编程的方式进行路由的跳转 -->
    <!-- 这种方式叫作 程式化导航 -->
    <button @click="homeHandler">首页</button>
    <button @click="aboutHandler">关于</button>
    <router-view></router-view>
  </div>
</template>
<script>
export default {
  data() {

```

```
    return {
      msg: "你好，关于页面",
    };
  },
  methods: {
    homeHandler() {
      this.$router.push({
        path: "/",
      });
    },
    aboutHandler() {
      // 跳转传参
      this.$router.push({
        // 跳转的路由路径
        path: "/about",
        // 想要传递的参数 使用query对象的形式进行传递
        query: {
          msg: this.msg,
        },
      });
    },
  },
};
</script>
```

```
//在about页面接收传递过来的参数 about.vue
<template>
  <div class="about">
    <h1>{{ $route.query.msg }}</h1>
  </div>
</template>
```

name-params传参

使用name与params结合的方式传递参数时，参数不会被拼接在浏览器地址栏中显示，并且刷新页面后数据会丢失。

```
<template>
  <div id="app">
    <!-- 通过绑定点击事件 在js方法中通过编程的方式进行路由的跳转 -->
    <!-- 这种方式叫作 编程式导航 -->
    <button @click="homeHandler">首页</button>
    <button @click="aboutHandler">关于</button>
    <router-view></router-view>
  </div>
</template>
<script>
export default {
  data() {
    return {
      msg: "你好，关于页面",
      msg2: "你好，首页",
    };
  },
  methods: {
    homeHandler() {
      // 跳转
      this.$router.push({
        // 配置路由时，设置的name属性 要保持一致 否则跳转会出问题
        name: "Home",
        // 参数使用params对象进行包裹
        params: {
          msg: this.msg2,
        },
      });
    },
  },
};
</script>
```

```
    });  
  },  
  aboutHandler() {  
    // 跳转传参  
    this.$router.push({  
      // 跳转的路由路径  
      path: "/about",  
      // 想要传递的参数 使用query对象的形式进行传递  
      query: {  
        msg: this.msg,  
      },  
    });  
  },  
},  
};  
</script>
```

//在Home.vue页面 可以接收传递过来的参数

```
<template>  
  <div class="home">  
    {{ $route.params.msg }}  
  </div>  
</template>
```