路由vue-router

官网地址:[https://router.vuejs.org/zh/]

一、路由简介

这里的路由并不是指我们平时所说的硬件路由器，这里的路由就是SPA（single page application）的路径管理器。vue的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。

传统的页面应用，是用一些超链接来实现页面切换和跳转的。在vue-router单页面应用中，则是路径之间的切换，也就是组件的切换。路由模块的本质 就是建立起url和页面之间的映射关系。

至于我们为啥不能用a标签，这是因为用Vue做的都是单页应用（当你的项目准备打包时，运行npm run build时，就会生成dist文件夹，这里面只有静态资源和一个index.html页面），所以你写的标签是不起作用的，你必须使用vue-router来进行管理。

二、实现原理

单页面应用(SPA)的核心之一是: 更新视图而不重新请求页面;
vue-router在实现单页面前端路由时，提供了两种方式：Hash模式和History模式；根据mode参数来决定采用哪一种方式。

1、vue-router实现原理之hash模式

hash 模式的原理是 onhashchange 事件(监测hash值变化)，可以在 window 对象上监听这个事件。

vue-router 默认 hash 模式 —— 使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。hash（#）是URL 的锚点，代表的是网页中的一个位置，单单改变#后的部分，浏览器只会滚动到相应位置，不会重新加载网页，也就是说hash 出现在 URL 中，但不会被包含在 http 请求中，对后端完全没有影响，因此改变 hash 不会重新加载页面；

2、vue-router实现原理之history模式

由于hash模式会在url中自带#，如果不想要很丑的 hash，我们可以用路由的 history 模式，只需要在配置路由规则时，加入"mode: 'history'"

这种模式充分利用了html5 history中新增的 pushState() 和 replaceState() 方法。这两个方法应用于浏览器记录栈，在当前已有的 back、forward、go 基础之上，它们提供了对历史记录修改的功能。只是当它们执行修改时，虽然改变了当前的 URL ，但浏览器不会立即向后端发送请求。

三、vue-router激活路由样式

给当前激活的菜单添加激活的样式

- 1. linkActiveClass

```
let router = new Router({
    linkActiveClass:'custom-active-class'
})
```

- 2. linkExactActiveClass需要路由精确匹配才会添加到对应菜单上

```
let router = new Router({
    linkExactActiveClass :'custom-exact-active-class'
})
```

四、编程式导航

使用一个声明出来的固定元素来实现跳转局限性大，必须要有某个固定的触发元素, 为了实现更加柔性的跳转, 我们就引申出编程式的导航。
使用场景eg:在一个路由组件里面点击可以切换到另一个路由组件。

```
// this.$router的基本方法
this.$router.push('目标路径') // 强制跳转至某一路径
this.$router.go(-1) //返回上一级
this.$router.replace('目标路径') // 路由替换
```

五、命名路由

之前我们已经学习过,用this.$router.push方式可以实现编程式路由,push方法除开可以传入字符串的参数外,还可以传入一个复杂的对象值。

```
this.$router.push("/home") // 字符串
this.$router.push({ path: 'home' }) // 对象
this.$router.push({ path: 'home' , query: {id : '123'}}) // 带查询参数就变成/home?
id=123
const userId = 123
this.$router.push({ path: `home/${userId}`}) //  /home/123
this.$router.push({ name: 'home', params: {id: '123'} }) //  /home/123
```

注意：如果使用path,params会被忽略

六、重定向和重命名

重定向的意思是，当用户访问/a的时候，url可以被替换成/b
重命名是访问路由/a时可以设置一个别名/d,那么当用户访问/d的时候访问的也是/a的页面

七、路由的导航守卫

vue-router 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中：全局的, 单个路由独享的, 或者组件级的。

每个守卫方法接收三个参数：

- to: Route: 即将要进入的目标 路由对象
- from: Route: 当前导航正要离开的路由
- next: Function: 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。
  如果next函数没有执行或是传入了false等值,这个跳转就会被终止掉

1、全局前置守卫

当一个导航触发时，全局前置守卫按照创建顺序调用。守卫是异步解析执行，此时导航在所有守卫 resolve 完之前一直处于 等待中。

```
// 可以使用 router.beforeEach 注册一个全局前置守卫
const router = new VueRouter({ ... })

router.beforeEach((to, from, next) => {
  // ...
})
```

2、路由独享守卫

```
// 可以在路由配置上直接定义 beforeEnter 守卫
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
        // ...
      }
    }
  ]
})
```

3、组件内的守卫

```
// 可以在路由组件内直接定义以下路由导航守卫
// beforeRouteEnter
// beforeRouteUpdate
// beforeRouteLeave
const Foo = {
  template: `...`,
  beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不! 能! 获取组件实例 `this`
    // 因为当守卫执行前，组件实例还没被创建
  },
  beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。
    // 可以访问组件实例 `this`
  },
  beforeRouteLeave (to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
  }
}
```

4、完整的导航解析流程

1. 导航被触发。
2. 在失活的组件里调用 beforeRouteLeave 守卫。
3. 调用全局的 beforeEach 守卫。
4. 在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。
5. 在路由配置里调用 beforeEnter。
6. 解析异步路由组件。
7. 在被激活的组件里调用 beforeRouteEnter。
8. 调用全局的 beforeResolve 守卫 (2.5+)。
9. 导航被确认。
10. 调用全局的 afterEach 钩子。
11. 触发 DOM 更新。
12. 调用 beforeRouteEnter 守卫中传给 next 的回调函数，创建好的组件实例会作为回调函数的参数传入。