可能比文档还详细--VueRouter完全指北

> 链接：https://juejin.cn/post/6844903665388486664#comment

前言

关于标题,应该算不上是标题党,因为内容真的很多很长很全面.主要是在官网的基础上又详细总结,举例了很多东西.确保所有新人都能理解!所以实际上很多东西是比官网还详细的.你想要的,在官网上没理解的,基本在这里都能找到解答!本来想分成两篇发的,但想想男人长点也没什么不好的.所以也希望各位收藏插眼标记(滑稽)

特点:本文主要是参考了官方文档.除了不常用的过渡动效和数据获取,都进行了分析说明.说明:每一节都在文档的基础上进行了更通俗的解释;例子:每一节都添加了单独的例子进行详细的说明,官方没有或复杂或略过的都有详细的说明.总结:每一节都会有tips注意点,实际开发的经验和总结.

使用方法:

如果是新手,就从头开始看,前5章的内容都是逐步让你熟悉VueRouter,并且第5章也给出了本地实际搭建的代码示例.

如果你对VueRouter有了一定了解,则可以根据目录,自行选择查看.或者全局搜索关键字快速定位.

吐槽:现在关于VueRouter相关能搜到的大部分都是copy完全摘抄官方文档拼凑的文章.很反感这样的文章!基本没什么营养!你在官网上不明白的在他那里基本也看不明白.

1, 概述

vue-router和vue.js是深度集成的,适合用于单页面应用.传统的路由是用一些超链接来实现页面切换和跳转.而vue-router在单页面应用中,则是组件之间的切换.**其本质就是:建立并管理url和对应组件之间的映射关系.**

2, 简单实例

这里简单先以官网的例子,了解vue-router有哪几大部分组成,对vue-router有一个初步印象.

HTML

首先是html部分.这里主要是两个作用:

1,router-link组件来导航,用户点击后切换到相关视图.

2,router-view组件来设置切换的视图在哪里渲染.(一个页面也可以有多个router-view分别展示特定的视图,并且支持嵌套)

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="app">
  <h1>Hello App!</h1>
  <p>
```

```
    <!-- 使用 router-link 组件来导航. -->
    <!-- 通过传入 `to` 属性指定链接. -->
    <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <!-- 路由出口 -->
  <!-- 路由匹配到的组件将渲染在这里 -->
  <router-view></router-view>
</div>
```

复制代码

JavaScript

这里实际项目写法代码可以参考我后面的第5章.下面我就简单总结下大概分那几步:

- 1,材料准备:
  - 1.1 引入准备好的'Vue'和'vue-router'(前提是已经npm)
  - 1.2 引入路由跳转的组件.下面例子中就是'Foo'和'Bar'
  - 1.3 启动全局组件VueRouter例如:Vue.use(VueRouter).这样vue-router才开始执行.
- 2, 配置路由实例:通过new VueRouter()详细配置每个路由的路径,对应的组件等等所有和路由相关的配置.
- 3, 将路由实例挂载到根实例上.new Vue({router}).$mount('#app')

```
// 0\. 如果使用模块化机制编程，导入Vue和VueRouter，要调用 Vue.use(VueRouter)

// 1\. 定义（路由）组件。
// 可以从其他文件 import 进来
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2\. 定义路由
// 每个路由应该映射一个组件。 其中"component" 可以是
// 通过 Vue.extend() 创建的组件构造器，
// 或者，只是一个组件配置对象。
// 我们晚点再讨论嵌套路由。
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3\. 创建 router 实例，然后传 `routes` 配置
// 你还可以传别的配置参数，不过先这么简单着吧。
const router = new VueRouter({
  routes // （缩写）相当于 routes: routes
})

// 4\. 创建和挂载根实例。
// 记得要通过 router 配置参数注入路由，
// 从而让整个应用都有路由功能
const app = new Vue({
```

```
  router
}).$mount('#app')

// 现在，应用已经启动了!

复制代码
```

通过注入路由器，我们可以在任何组件内通过 `this.$router` 访问路由器，也可以通过 `this.$route` 访问当前路由对象：

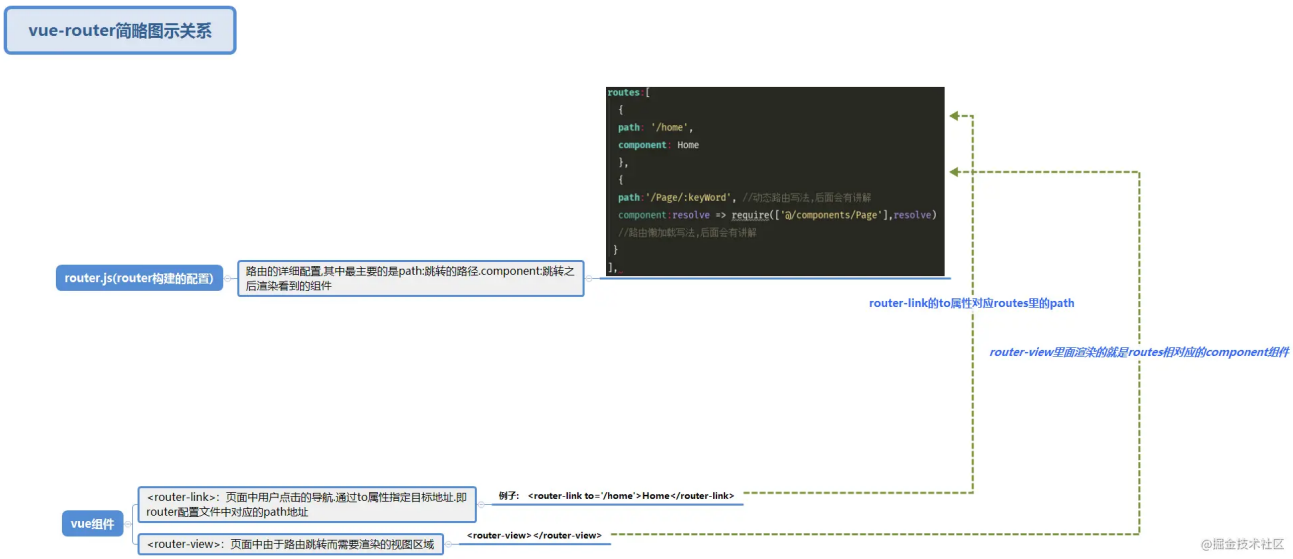
Tips:这里我简单说明下 `$router` 和 `$route` 的区别:

- `$router` 是指整个**路由实例**,你可以操控整个路由,通过'$router.push'往其中添加任意的路由对象.
- `$route` :是指当前路由实例('$router')跳转到的**路由对象**;
- 路由实例可以包含多个路由对象.它们是父子包含关系.

```
// Home.vue
export default {
  computed: {
    username () {
      // 我们很快就会看到 `params` 是什么
      return this.$route.params.username
    }
  },
  methods: {
    goBack () {
      window.history.length > 1
        ? this.$router.go(-1)
        : this.$router.push('/')
    }
  }
}
复制代码
```

3, vue-router简略图示关系(有问题需要修改)

图片中router.js就是路由实例配置文件



PS:

这里图说的有点不准确,to并不是仅仅对应path值.要分具体情况,

详情查看官方to的api.

4, 学习的准备工作 !important

4.1 硬件准备

学习一个新知识,只看然后想想.嗯,大概是这样的.这样是没有任何效果的.没有实际敲过,使用过,都算不上是你的知识.所以,**强烈建议大家边看边敲边思考总结.** 这里提供两种方式:

- 官网在线实例:点击这里通过在线实例,按照教程边看边敲!
- 通过我第5章的教程,在自己本地的实际项目上进行构建,查看效果.

这两种方式都很好.个人建议有条件的选第二种,更加直观,更加贴近自己的实际项目需求.

4.2 思想准备

学习配置vue-router,一定要考虑两个层面:

- Html:对应的就是vue-router里的`<router-link>`和`<router-view>`
- JavaScript:就是具体路由实例的配置. 整个vue-router的配置也是围绕这两个层面进行展开的,学习时一定要有这个意识,才更容易理解和使用!

5, 实际项目构建示例

5.1,下载vue-router :

```
    npm i vue-router -s
```

复制代码

5.2, 在vue组件内配置router-link和router-view

- router-link:映射路由.就是创建a标签来定义路由导航的链接(用户通过点击实现跳转).通过to属性指定目标地址.默认渲染成带有正确链接的标签.
- router-view:就是在标签内渲染你路由匹配到的视图组件.router-view支持嵌套router-view,并且支持多个router-view分别渲染不同的component.详细点击文档命名视图.

```
<!--这里引用官方例子的写法-->
<div id='app'>
    <p>
        <router-link to="/user/foo">/user/foo</router-link>
        <router-link to="/user/bar">/user/bar</router-link>
    </p>
    <router-view>PS:写在这里,即router-view里的内容是不会显示在页面上的!</router-view>
</div>
复制代码
```

关于to的更详细用法: 可以参考官方的api文档.用法很简单这里我就不重复了.

5.3, 配置路由实例 router.js文件

```
//第一步:引入必要的文件
import Vue from 'vue';//加载全局组件时,都需要引入Vue
import Router from 'vue-router';//引入vue-router
//引入在路由中需要用到的组件
import User from '@/components/user/User' //这里省略了.vue
...

//第二步:加载Router
Vue.use(Router);//加载全局组件Router

//第三步:配置路由实例
export default new Router({
    //mode:'history', //路由模式:默认为hash,如果改为history,则需要后端进行配合
    //base:'/',//基路径:默认值为'/'.如果整个单页应用在/app/下,base就应该设为'/app/'.一般可以写成__dirname,在webpack中配置.
    routes:[{
        path: '/user',
        name: 'user', //给路由命名,设置的name要唯一!
        component: User//就是第一步import的组件
        },{
            //路由懒加载:单页面应用,首页时,加载内容时间过长.运用懒加载对页面组件进行划分,减少首页加载时间
            path:'/Page',
            name:'Page',
            component:resolve => require(['@/components/Page'],resolve)
            //此时component则不需要在第一步import
        }]
})
复制代码
```

5.4, 在main.js中引入router.js并挂载到Vue实例

建议在实际项目中将router配置单独的js文件,更加清晰,不要都混在main.js中.

```
import router from './router'

new Vue({
  el: '#app',
  router,//不简写就是router:router,
  store,
  template: '<App/>',
  components: {
  App

  } })
复制代码
```

5.5, router,routes,route傻傻分不清?

- 1, router:一般指的就是路由实例.如$router.
- 2, routes:指router路由实例的routes API.用来配置多个route路由对象.
- 3, route:指的就是路由对象.例如;$route指的就是当前路由对象.

5.6, 小结

至此,一个基本的vue-router已经完成.这里只是简单示范,了解其实现步骤. 总结一下编写vue路由时,记住要**做这三类事: 1,准备工作:** 在main.js中引入router.js挂载到Vue实例中. **2,配置路由实例(重点):** 在router.js中引入Vue,vue-router,配置路由实例. **3,组件内配置:** 就是配置router-link和router-view.

6, vue-router的两种模式

一般单页面应用是(SPA)不会请求页面而是只更新视图. vue-router提供了两种方式来实现前端路由:Hash模式和History模式,可以用mode参数来决定使用哪一种方式.

6.1,Hash模式

vue-router默认使用Hash模式.使用url的hash来模拟一个完整的url. `此时url变化时,浏览器是不会重新加载的.` Hash(即#)是url的锚点,代表的是网页中的一个位置,仅仅改变#后面部分,浏览器只会滚动对应的位置,而不会重新加载页面. `#仅仅只是对浏览器进行指导,而对服务端是完全没有作用的!它不会被包括在http请求中,故也不会重新加载页面.` 同时**hash发生变化时,url都会被浏览器记录下来,这样你就可以使用浏览器的后退了.**

总而言之:Hash模式就是通过改变#后面的值,实现浏览器渲染指定的组件.

6.2,History模式

如果你不喜欢hash这种#样式.可以使用history模式.这种模式利用了HTML5 History新增的**pushState()和replaceState()方法.** 除了之前的back,forward,go方法,这两个新方法可以应用在浏览器历史记录的增加替换功能上.使用History模式,通过历史记录修改url,但它不会立即向后端发送请求. **注意点:** 虽然History模式可以丢掉不美观的#,也可以正常的前进后退,但是刷新f5后,此时浏览器就会访问服务器,在没有后台支持的情况下,此时就会得到一个404!官方文档给出的描述是:"不过这种模式要玩好,还需要后台配置支持.因为我们的应用是单个客户端应用,如果后台没有正确的配置,当用户直接访问时,就会返回404.所以呢,你要在服务端增加一个覆盖所有情况的的候选资源;如果url匹配不到任何静态资源,则应该返回同一个index.html页面."

总而言之:History模式就是通过pushState()方法来对浏览器的浏览记录进行修改,来达到不用请求后端来渲染的效果.不过建议,实际项目还是使用history模式.

例子:

```
const router = new VueRouter({
  mode: 'history', //如果这里不写,路由默认为hash模式
  routes: [...]
})
复制代码
```

7,动态路由匹配

当我们经常需要把某种模式匹配到所有的路由,全部都映射到同个组件.例如:我们有一个User组件,对于所有ID各不相同的用户,都要使用这个组件来渲染.这时我们就可以配置动态路由来实现. **动态路由匹配本质上就是通过url进行传参**

7.1,路由对象属性介绍:

为了下面理解的方便这里简单介绍下常用的路由对象属性,在组件内可以通过 `this.$route` (不是 `$router` !)进行访问.

$route.path 类型: `string` 字符串，对应当前路由的路径，总是解析为绝对路径，如 `"/foo/bar"` 。

$route.params 类型: `Object` 一个 key/value 对象，包含了动态片段和全匹配片段，如果没有路由参数，就是一个空对象。

$route.query 类型: `Object` 一个 key/value 对象，表示 URL 查询参数。例如，对于路径 `/foo?user=1` ，则有 `$route.query.user == 1` ，如果没有查询参数，则是个空对象。

$route .name 当前路由的名称，如果有的话。**这里建议最好给每个路由对象命名,方便以后编程式导航.不过记住name必须唯一!**

$route.hash 类型: `string` 当前路由的 hash 值 (带 `#`) ，如果没有 hash 值，则为空字符串。

$route.fullPath 类型: `string` 完成解析后的 URL，包含查询参数和 hash 的完整路径。

$route.matched 类型: `Array<RouteRecord>` 一个数组，包含当前路由的所有嵌套路径片段的**路由记录** 。路由记录就是 `routes` 配置数组中的对象副本 (还有在 `children` 数组)。 **$route.redirectedFrom** 如果存在重定向，即为重定向来源的路由的名字。

7.2 使用params进行配置:

举个例子:

```
routes:[{
    //动态路径参数,以冒号开头
    path:'/user/:id',
    component:User
}]
复制代码
```

这样,就是使用params进行配置.像/user/foo和/user/bar都将映射到相同的路由.

- 一个路径参数使用':'冒号进行标记.
- 当匹配到一个路由时,参数就会被设置到 `this.$route.params` ,可以在每个组件内使用.例如/user/foo在 `this.$route.params.id` 就为foo

这里以官方的表格示例进行展示

模式	匹配路径	$route.params
/user/:username	/user/evan	`{ username: 'evan' }`
/user/:username/post/:post_id	/user/evan/post/123	`{ username: 'evan', post_id: 123 }`

这里再举一个稍微变化一下的例子,来加深理解:

```
routes:[
    {path:'/user/:shot/foo/:id', component:shotCat}
]
复制代码
<p>
    <router-link to="/user/shot/foo">/user/shot/foo</router-link>    <!--无法匹配到对应路
由-->
    <router-link to="/user/shot/cat/foo">/user/shot/cat/foo</router-link>    <!--无法匹配
到对应路由-->
    <router-link to="/user/foo/foo/foo">/user/foo/foo/foo</router-link> <!--成功匹
配,$route.params.shot为foo;$route.params.cat为foo;-->
    <router-link to="/user/shot/foo/cat">/user/shot/foo/cat</router-link><!--成功匹
配,$route.params.shot为shot;$route.params.cat为cat;-->
</p>
<router-view></router-view>
复制代码
```

tips:

- 有时候,同一个路径可以匹配多个路由,此时,匹配的优先级就按照路由的定义顺序.谁先定义的,谁的优先级就最高.
- 由于路由参数对组件实例是复用的.例如:

  ```
  /user/foo
  ```

  和

```
/user/bar
```

在使用路由参数时,复用的都是

```
User
```

组件.此时组件的生命周期钩子不会再被调用。如果你想路径切换时,进行一些初始化操作时,可以用以下两种解决办法:

- 在组件内 watch `$route` 对象:

```
const User = {
 template: '...',
 watch: {
   '$route' (to, from) {
     // 对路由变化作出响应...
   }
 }
}
复制代码
```

- 使用2.2版本中的 beforeRouteUpdate 路由守卫:

```
const User = {
  template: '...',
  beforeRouteUpdate (to, from, next) {
    // react to route changes...
    // don't forget to call next()
  }
}
复制代码
```

7.3,通过query进行配置传参.

在项目里我们可以通过上面提到的params进行传参.同时也可以用query进行传参. 举个例子: `<router-link to="/user?id=foo">foo</router-link>` vue-route会自动将?后的id=foo封装进this.$route.query里. 此时,在组件里this.$route.query.id值为'foo'. ==除了通过`router-link`的`to`属性. query也可以通过后面讲到的编程式导航进行传参==

8, 编程式导航

什么是编程式导航,编程式导航就是在vue组件内部通过`this.$router`访问路由实例,并通过this.$router.push导航到不同的url,进行路由映射,所以 **它的作用是和是一毛一样的!** 当然,**前提是你已经在routes里配置了对应的路由对象.**

一般什么时候用到编程式导航? 如果,你想在路由跳转前做点其他事情,例如权限验证等.但是用`<router-link>`的话,就直接跳转了.此时就可以使用编程式导航!

8.1, 编程式导航的写法

编程式导航一般都是用到router.push方法.该方法的参数可以是一个字符串路径,或者一个描述地址的对象.例如:

```
//字符串
this.$router.push('home')

//对象
this.$ruter.push({path:'home'})

//命名路由
this.$router.push({name:'user',params:{userId:2333}})

//带查询参数,变成/register?plan=private
this.$router.push({path:'register',query:{plan:'private'}})

复制代码
```

注意:==原谅色警告==:**path和params是不能同时生效的!**,否则params会被忽略掉.所以使用对象写法进行params传参时,要么就是`path`加冒号`:`,要么就是像上例中的'命名路由'.通过name和params进行传参.**然而query却并不受影响,有没有path都可以进行传参.**

8.2, router.replace方法

router.replace和router.push很像,写法一样.但实际效果不一样.push是向history里添加新记录.**而replace是直接将当前浏览器history记录替换掉!**

那最直接的后果是什么呢? 举个例子:

- 用push方法,页面1跳转到页面2,你使用浏览器的后退可以回到页面1
- 用replace方法,页面1被替换成页面2,你使用浏览器的后退,**此时你回不到页面1**,只能回到页面1的前一页,页面0.

那什么时候会用到replace呢? 当你不想让用户回退到之前的页面时,常见于权限验证,验证后就不让用户回退到登录页重复验证.

8.3, router.go(n)方法

这个方法的参数就是一个整数,意思是在history记录中前进或后退多少步.类似window.history.go(n).这样就能控制页面前进或者后退多少步.

9 对动态路由和编程式导航相关的补充小结:

补充: 实际上不通过routes配置,也可以用下面这种方法直接在`router-link`上通过`to`进行传参. **关于to的更详细用法:** 可以参考官方的api文档.用法很简单这里我就不重复了.

```
routes:[
    {name:'shotCat',path:'/shotCat', component:shotCat}
]
复制代码
<router-link :to="{ name:'shotCat',params:{paramId:'hello'},query:
{queryId:'world'}}">helloWorld</router-link> <!--此时通过name匹配到路由对象shotCat.-->
<router-link :to="{ path:'/shotCat',params:{paramId:'hello'},query:
{queryId:'world'}}">helloWorld</router-link>  <!--此时通过path匹配到路由对象shotCat.但
是!!!!!此时`paramId`并不能添加到`$route.params`里,只有`queryId`成功添加到`$route.query`-->
复制代码
```

通过两个`router-link`.可以发现这种写法和编程式导航的规则一毛一样, **path和params是不能同时生效的!** 所以建议大家最好给每个路由对象进行命名!** `query`是path和name都可以正常传参的. 这里大家可以通过这个官方在线例子进行修改验证`{{$route.params}}`和`{{$route.query}}`是否成功传递. **小结:**

- 1,`<router-link :to="{ }">`等同于`this.$router.psh()`. **path和params是不能同时存在的!**,想通过params,就得加上name属性.`query`不受影响.
- 2,

  ```
  <router-link :to="{ }">
  ```

  和

  ```
  this.$router.psh()
  ```

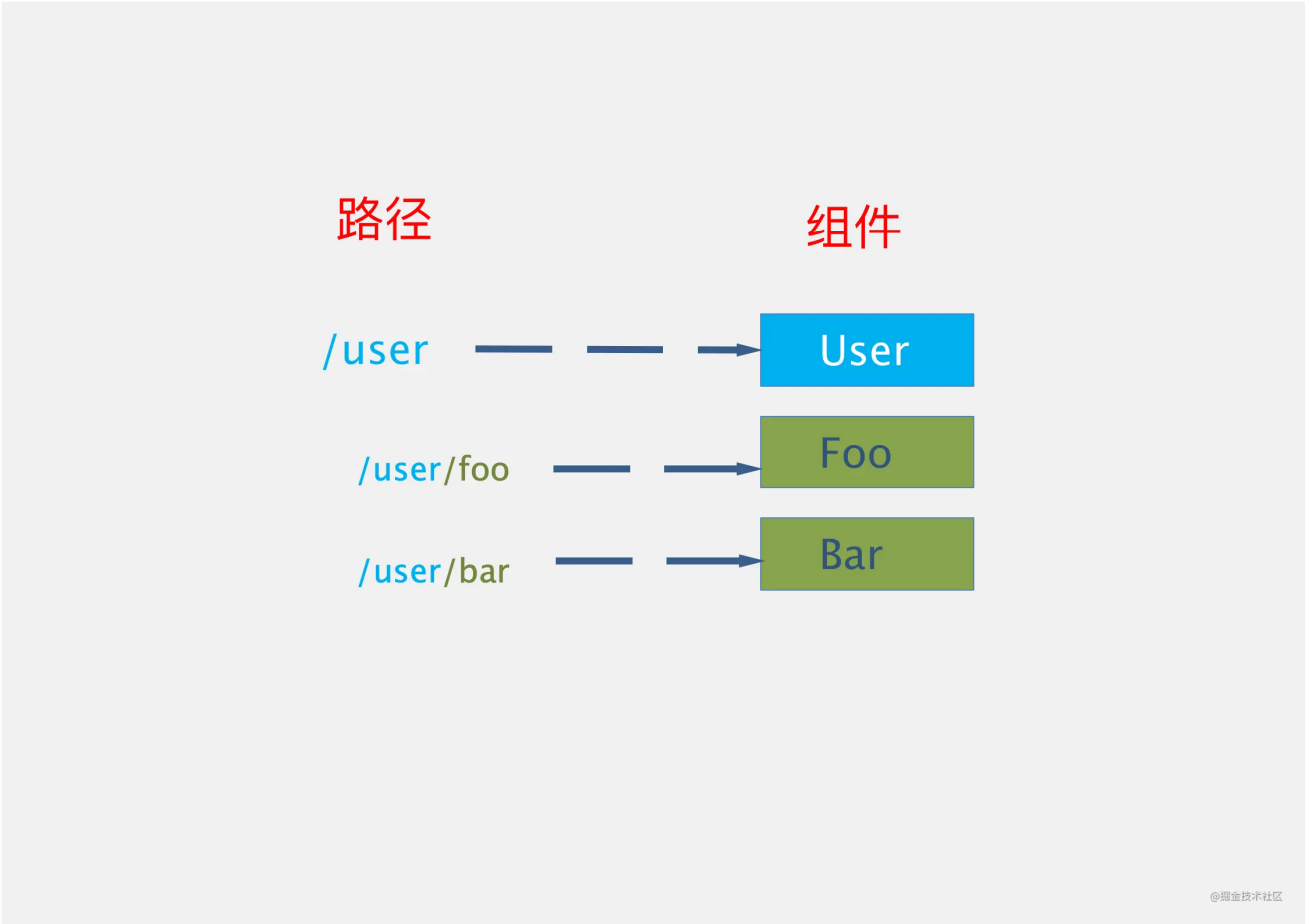
  的实际效果也是一样的.

  - 2.1 **params参数都不会显示在url地址栏中**.除了在路由中通过routes进行配置的.所以用户**刷新页面后,params参数就会丢失!**
  - 2.2 **query参数可以正常显示在url地址栏中**.刷新页面后也不会丢失

通过`to`虽然可以进行`params`,`query`传参.但是注意此时**页面url并不会改变!**.所以你刷新页面后,参数就没有了.

10 嵌套路由与单组件多视图

- **嵌套路由**:就是父路由嵌套子路由.url上就是/user嵌套两个子路由后就是/user/foo和/uer/bar.用一张图表示就是:



- **单组件多视图**:就是一个组件里有多个视图进行展示.即包含有多个`<router-view/>`

10.1 嵌套路由

讲之前,必须先清楚这样一件事,**一个对应展示的就是一个组件** 因此实现嵌套路由有两个要点:

- 路由对象中定义子路由(嵌套子路由)
- 组件内`<router-view/>`的使用.

这两点也对应了我第4章中关于思想准备的说明.

下面我的示例还是以上图嵌套路由为例进行讲解. 理解了的可以看官方示例是嵌套路由搭配动态路由使用的例子,大家可以在此基础上自己修改尝试.

10.1.1 路由对象中定义子路由

```
const router = new VueRouter({
  routes: [
    { path: '/user', component: User,name:'user',
    //嵌套路由就写在children配置中,写法和rutes一样.
      children: [
        { path: '', component: UserDefault ,name:'default',
            //children:[{}]    也可以继续添加children嵌套
        },
```

```
        //如果/user下没有匹配到其他子路由时,User的<router-view>是什么都不会显示的,如果你想让它显示点什么.可以将path:''.设为空.此时UserDefault就是默认显示的组件.

        { path: 'foo', component: UserFoo,name:'foo'},
        //此时path等同于'/user/foo',子路由会继承父路由的路径.但是不能写成path:'/foo'.因为以 / 开头的嵌套路径会被当作根路径,也就是说此时foo成了根路径.而不是user.

        { path: 'bar', component: UserBar,name:'bar' }
      ]
    }
  ]
})
复制代码
```

10.1.2 组件内`<router-view/>`的使用.

```
<div id="app">
 <p>
   <router-link to="/user">/user</router-link>
   <router-link to="/user/foo">/user/foo</router-link>
   <router-link to="/user/bar">/user/bar</router-link>
 </p>
 <router-view></router-view> <!--这里展示的是User组件;同样User的<router-view/>也被嵌套在里面-->
</div>
复制代码
const User = {
 template: `
   <div class="user">
     <h2>User</h2>
     <router-view></router-view>
   </div>
 `
}
//User的<router-view>里展示的就是子路由foo,bar的组件还有default默认组件

const UserDefault = { template: '<div>default</div>' }
const UserFoo = { template: '<div>foo</div>' }
const UserBar = { template: '<div>bar</div>' }
复制代码
```

10.2 单组件多视图

如果一个组件有多个视图,来展示多个子组件.这个时候就需要用到**命名视图** 官方的在线示例在这里,大家可以在此基础上自己修改尝试. 直接上我的例子:

```
<div id="app">
 <h1>Named Views</h1>
 <p>
   <router-link to="/avenger">复仇者联盟</router-link>
 </p>
```

```
 <router-view ></router-view>
 <router-view name="ironMan"></router-view>
 <router-view name="captainAmerica"></router-view>
</div>
<!--这里我们给其中两个视图命名为ironMan和captainAmerica;没有设置name的视图,会获得默认命名为
default>
复制代码
const router = new VueRouter({
  routes: [
    { path: '/avenger', name:'avenger',components: {
        default: stanLee,
        ironMan: ironMan,
        captainAmerica: captainAmerica
      }
    //如果有多个视图需要展示时,以前的component换成components(加上s!!),写成对象形式.左边的ironMan
指的就是<router-view>里设置的name="ironMan";右边的则指的是下面的组件ironMan.
    }
  ]
})

const stanLee = { template: '<div>斯坦李</div>' }
const ironMan = { template: '<div>钢铁侠</div>' }
const captainAmerica = { template: '<div>美国队长</div>' }
复制代码
```

嵌套命名视图: 官方的在线示例,结合了嵌套路由和命名视图.就是两种写法的组合.**建议初学者通过这个例子加深理解.**

11 重定向和别名:

11.1 重定向配置:

重定向其实就是通过路由.拦截path,然后替换url跳转到redirect所指定的路由上. 重定向是通过 routes 配置来完成,

```
//从 /a 重定向到 /b
const router = new VueRouter({
    routes:[
        {path:'/a',redirect:'/b'}
    ]
})

///从 /a 重定向到 命名为'foo'的路由
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: { name: 'foo' }}
  ]
})

//甚至是一个方法，动态返回重定向目标:
const router = new VueRouter({
```

```
  routes: [
    { path: '/a', 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
      const { hash, params, query } = to
      //这里使用了ES6的解构写法,分别对应了to的hash模式,params,query参数.这里解构就不具体说明了.
        if (query.to === 'foo') {
          return { path: '/foo', query: null }
        }
        if (hash === '#baz') {
          return { name: 'baz', hash: '' }
        }
        if (params.id) {
          return '/with-params/:id'
        } else {
          return '/bar'
        }
    }}
  ]
})

复制代码
```

11.2 别名

重定向是替url换路径,达到路由跳转.那别名就是一个路由有两个路径.两个路径都能跳转到该路由. 举个栗子:你可能大名叫'赵日天',但你的小名(别名)可能就叫'二狗子'.但'赵日天'和'二狗子'指代的是同一个人(路由). 别名是在rutes里的alias进行配置:

```
const router = new VueRouter({
//这时,路径'/fxxksky'和'/two-dogs' 都会跳转到A
  routes: [
    { path: '/fxxksky', component: A, alias: '/two-dogs' }
    //当有多个别名时,alias也可以写成数组形式.  alias: ['/two-dogs', 'three-dogs','four-
dogs','five-dogs']
  ]
})
复制代码
```

12 路由组件传参

路由传参,可以通过前面介绍的params和query进行传参.但这两种传参方式,本质上都是把参数放在url上,通过改变url进行的.这样就会造成参数和组件的高度耦合. 如果我想传参的时候,可以更自由,摆脱url的束缚.这时就可以使用rute的props进行解耦.提高组件的复用,同时不改变url.

下面就以例子进行讲解: **PS:** 这部分官方没有在线实例,大家可以将我下面例子的代码将之前的在线例子进行覆盖就可以了

```
//路由配置:
```

```
const Hello = {
  props: ['name'], //使用rute的props传参的时候,对应的组件一定要添加props进行接收,否则根本拿不到传参
  template: '<div>Hello {{ $route.params}}和{{this.name}}</div>'
  //如果this.name有值,那么name已经成功成为组件的属性,传参成功
}

const router = new VueRouter({
mode: 'history',
  routes: [
    { path: '/', component: Hello }, // 没有传参  所以组件什么都拿不到
    { path: '/hello/:name', component: Hello, props: true }, //布尔模式: props 被设置为true, 此时route.params (即此处的name)将会被设置为组件属性。
    { path: '/static', component: Hello, props: { name: 'world' }}, // 对象模式: 此时就和params没什么关系了.此时的name将直接传给Hello组件.注意:此时的props需为静态!
    { path: '/dynamic/:years', component: Hello, props: dynamicPropsFn }, // 函数模式: 1,这个函数可以默认接受一个参数即当前路由对象.2,这个函数返回的是一个对象.3,在这个函数里你可以将静态值与路由相关值进行处理.
    { path: '/attrs', component: Hello, props: { name: 'attrs' }}
  ]
})

function dynamicPropsFn (route) {
  return {
    name: (new Date().getFullYear() + parseInt(route.params.years)) + '!'
  }
}

new Vue({
  router,
  el: '#app'
})
复制代码
<!--html部分-->
    <div id="app">
      <h1>Route props</h1>
      <ul>
        <li><router-link to="/">/</router-link></li>
        <li><router-link to="/hello/you">/hello/you</router-link></li>
        <li><router-link to="/static">/static</router-link></li>
        <li><router-link to="/dynamic/1">/dynamic/1</router-link></li>
        <li><router-link to="/attrs">/attrs</router-link></li>
      </ul>
      <router-view></router-view>
    </div>
复制代码
```

13 路由懒加载

vue主要用于单页面应用，此时webpack会打包大量文件,这样就会造成首页需要加载资源过多,首屏时间过长,给用户一种不太友好的体验. 如果使用路由懒加载,仅在你路由跳转的时候才加载相关页面.这样首页加载的东西少了,首屏时间也减少了. vueRouter的懒加载主要是靠**Vue 的异步组件**和 **Webpack 的代码分割功能**，轻松实现路由组件的懒加载。 这里写法其实我在第5.3章里的例子已经写过,比较简单只需要将组件以promise形式引入即可.

```
routes:[
      path:'/',
      name:'HelloWorld',
      component:resolve=>require(['@/component/HelloWorld'],resolve)
  ]
  //此时HelloWorld组件则不需要在第一步import进来
复制代码
```

13.1 把组件按组分块

把组件按组分块可以把路由下的所有组件都打包在同个异步块 (chunk) 中,并且在f12的network里面看到动态加载的组件名字. **前提条件:**

- Webpack版本 > 2.4
- 需要在webpack.base.conf.js里面的output里面的filename下面加上chunkFileName

```
output: {
 path: config.build.assetsRoot,
 filename: '[name].js',
 // 需要配置的地方
 chunkFilename: '[name].js',
 publicPath: process.env.NODE_ENV === 'production'
   ? config.build.assetsPublicPath
   : config.dev.assetsPublicPath
}
复制代码
```

此时在引入组件时的写法需要使用 命名 chunk，一个特殊的注释语法来提供 chunk name

```
const Foo = () => import(/* webpackChunkName: "group-foo" */ './Foo.vue')
const Bar = () => import(/* webpackChunkName: "group-foo" */ './Bar.vue')
const Baz = () => import(/* webpackChunkName: "group-foo" */ './Baz.vue')
复制代码
```

14 导航守卫

路由导航守卫,通俗点说就是路由钩子.作用也和生命周期钩子类似,在路由跳转过程进行操作控制. 导航守卫有很多钩子,这里我就不照搬官方文档了.这里大家自己先点击阅读完导航守卫,然后再看我的总结和代码例子.

14.1 导航守卫分类

- 1,全局守卫:

  :异步执行,每个路由跳转都会按顺序执行.

- router.beforeEach 全局前置守卫
- router.beforeResolve 全局解析守卫(2.5.0+) 在beforeRouteEnter调用之后调用.
- router.afterEach 全局后置钩子 进入路由之后 **注意:不支持next()**,只能写成这种形式 router.afterEach((to, from) => {});

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对象
- **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。
  - **next()**: 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 **confirmed** (确认的)。
  - **next(false)**: 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 `from` 路由对应的地址。
  - **next('/') 或者 next({ path: '/' })**: 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 `next` 传递任意位置对象，且允许设置诸如 `replace: true`、`name: 'home'` 之类的选项以及任何用在 `router-link` 的 `to` prop 或 `router.push` 中的选项。
  - **next(error)**: (2.4.0+) 如果传入 `next` 的参数是一个 `Error` 实例，则导航会被终止且该错误会被传递给 `router.onError()` 注册过的回调。

官方介绍的比较简单,没有实际栗子,下面我就通过栗子再进行详细的说明

```
//1,可以在main.js 或者在单独的路由配置文件router.js中进行设置
    router.beforeEach((to, from, next) => {
    ...
      next();
    });

//2,也可以在组件内部设置
    this.$router.beforeEach((to, from, next) => {
    ...
      next();
    });

//3,对函数及next()的详细使用说明
    router.beforeEach((to, from, next) => {
    //首先to和from 其实是一个路由对象,所以路由对象的属性都是可以获取到的(具体可以查看官方路由对象的
api文档).
    //例如:我想获取获取to的完整路径就是to.path.获取to的子路由to.matched[0].
      next();//使用时,千万不能漏写next!!!
    //next()  表示直接进入下一个钩子.
    //next(false)  中断当前导航
    //next('/path路径')或者对象形式next({path:'/path路径'})  跳转到path路由地址
    //next({path:'/shotcat',name:'shotCat',replace:true,query:{logoin:true}...})  这种对
象的写法,可以往里面添加很多.router-link 的 to prop 和 router.push 中的选项(具体可以查看api的官方
文档)全都是可以添加进去的,再说明下,replace:true表示替换当前路由地址,常用于权限判断后的路由修改.
    //next(error)的用法,(需2.4.0+)
    }).catch(()=>{
  //跳转失败页面
  next({ path: '/error', replace: true, query: { back: false }})
})
//如果你想跳转报错后,再回调做点其他的可以使用 router.onError()
```

```
router.onError(callback => {
      console.log('出错了!', callback);
    });
复制代码
```

- 2,路由独享的守卫:

  即路由对象独享的守卫

  - beforeEnter:路由只独享这一个钩子,在rutes里配置

```
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
        // 使用方法和上面的beforeEach一毛一样
      }
    }
  ]
})
复制代码
```

- 3,组件内的守卫:

  注意:这类路由钩子是写在组件内部的,

  - beforeRouteEnter 进入路由前,此时实例还没创建,无法获取到zhis
  - beforeRouteUpdate (2.2) 路由复用同一个组件时
  - beforeRouteLeave 离开当前路由,此时可以用来保存数据,或数据初始化,或关闭定时器等等
  -

这里官方的例子说明的很详细,这里就直接进行引用了.

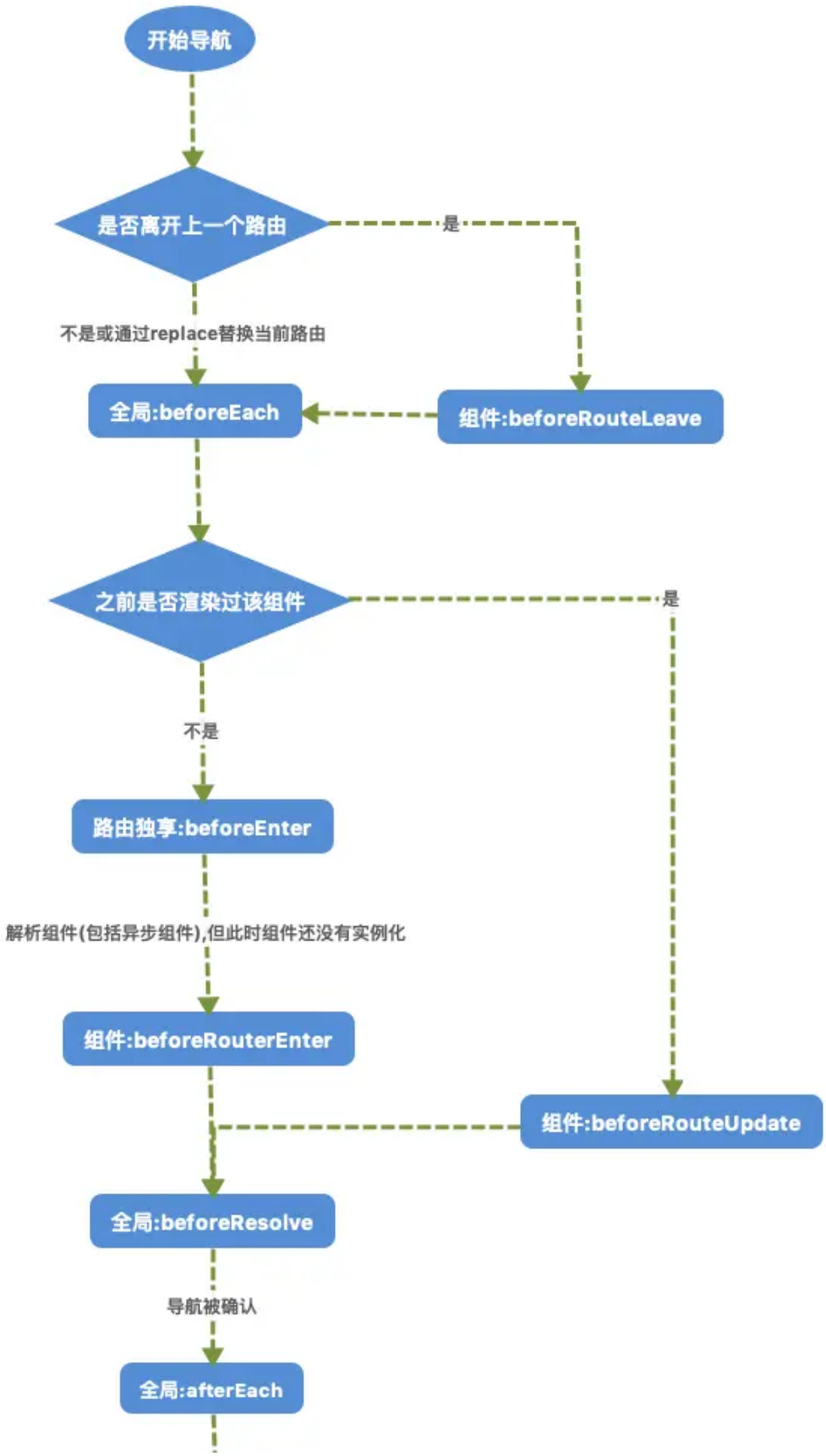
```
//在组件内部进行配置,这里的函数用法也是和beforeEach一毛一样
const Foo = {
  template: `...`,
  beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不!能!获取组件实例 `this`
    // 因为当守卫执行前，组件实例还没被创建
  },
  beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，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`
  }
```

```
}
```
复制代码

14.2 完整的导航解析流程

官方文档给的说明都是文字形式的,不是特别直观.这里就不copy了.这里就直接以流程图的形式进行展示(这里参考了这位同学的图,在此感谢!).

导航解析流程


开始导航
是否离开上一个路由
是
不是或通过replace替换当前路由
全局:beforeEach
组件:beforeRouteLeave
之前是否渲染过该组件
是
不是
路由独享:beforeEnter
解析组件(包括异步组件),但此时组件还没有实例化
组件:beforeRouterEnter
组件:beforeRouteUpdate
全局:beforeResolve
导航被确认
全局:afterEach

15 路由元信息

15.1 什么是路由元信息

一句话概括:路由配置的meta对象里的信息. 官方栗子:

```
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      children: [
        {
          path: 'bar',
          component: Bar,
          // a meta field
          meta: { requiresAuth: true }
        }
      ]
    }
  ]
})
复制代码
```

从栗子可以看出就是给路由添加了一个自定义的meta对象,并在里面设置了一个requiresAuth状态为true.

15.2 它有什么用

从下面的另一个官方栗子里已经给出了答案.

```
router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.requiresAuth)) {
    //对matched不了解的建议看官方api文档,或我7.1节的说明
        //数组some方法,如果meta.requiresAuth为ture,则返回true.此时,说明进入该路由前需要判断用户是否已经登录
    if (!auth.loggedIn()) {   //如果没登录,则跳转到登录页
      next({
        path: '/login',
```

```
        query: { redirect: to.fullPath }  //官方例子的这个小细节很好,通过query将要跳转的路由路径保存下来,待完成登录后,就可以直接获取该路径,直接跳转到登录前要去的路由
      })
    } else {
      next()
    }
  } else {
    next() // 确保一定要调用 next()
  }
})
复制代码
```

我们可以通过在meta里设置的状态,来判断是否需要进行登录验证.如果meta里的requiresAuth为true,则需要判断是否已经登录,没登录就跳转到登录页.如果已登录则继续跳转.

此时,可能会有同学说,前面说的path,params,query都可以存储信息,作为登录验证的状态标记.的确,它们也可以达到同样的效果.如果是少量单个的验证,使用它们问题不大. 但如果是多个路由都需要进行登录验证呢? path,params,query是把信息显性地存储在url上的.并且多个路径都把一个相同的状态信息加在url上.这样就使url不再单纯,并且也很不优雅美观. 所以要优雅要隐性地传递信息,就使用meta对象吧!

16 滚动行为

当年切换路由时,可以使页面滚动到你想要的某个地方,或者是保持之前滚动的位置,这时你就需要使用scrollBehavior这个方法.

注意点:

- 这里控制和记住的滚动位置都是仅对整个组件页面而言的,并不包含你组件里面其他的滚动条.
- 这里路由的模式只能是history.因为它使用了History新增的pushState().具体可以看我第6章的说明.

```
const router = new VueRouter({
mode:'history',//这个不能忘,默认是hash模式
  routes: [...],
  scrollBehavior (to, from, savedPosition) {
     // to: 要进入的目标路由对象, 到哪里去.和导航守卫的beforeEach一样
     //from: 离开的路由对象, 哪里来
     //savedPosition: 点击前进/后退的时候记录值{x:?,y:?}.并且只有通过浏览器的前进后退才会触发.
    // return 期望滚动到哪个的位置 { x: number, y: number }或者是{ selector: string, offset? : { x: number, y: number }},这里selector接收字符串形式的hash,如'#foo',同时你还可以通过offset设置偏移,版本需要大于2.6+
    //举个实例
    if(savePosition) { //如果是浏览器的前进后退就,返回之前保存的位置
      return savePosition;
    }else if(to.hash) {//如果存在hash,就滚动到hash所在位置
      return {selector: to.hash}
    }else{
      return {x:0,y:0}//否则就滚动到顶部
    }
  }
})
```