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：

- 嵌套的路由/视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- 基于 Vue.js 过渡系统的视图过渡效果
- 细粒度的导航控制

- 带有自动激活的 CSS class 的链接
- HTML5 历史模式或 hash 模式，在 IE9 中自动降级
- 自定义的滚动条行为

起步

用 Vue.js + Vue Router 创建单页应用，是非常简单的。使用 Vue.js ，我们已经可以通过组合组件来组成应用程序，当你要把 Vue Router 添加进来，我们需要做的是，将组件 (components) 映射到路由 (routes)，然后告诉 Vue Router 在哪里渲染它们

> 安装
>
> `npm i vue-router -S`
>
> 在main.js中
>
> ```
> 1 import Vue from 'vue'
> 2 import VueRouter from 'vue-router'
> 3
> 4 //如果使用模块化机制编程，导入Vue和VueRouter，要调用 Vue.use(VueRouter)
> 5 Vue.use(VueRouter)
> ```
>
> 推荐使用:vue add router 添加插件(记得提前提交)

基本使用

router.js

```
1 import Vue from 'vue'
2 //1.导入
3 import Router from 'vue-router'
4 import Home from './views/Home.vue'
```

```
import About from './views/About.vue'
//2.模块化机制 使用Router
Vue.use(Router)

//3.创建路由器对象
const router = new Router({
    //定义路由 一个路由应该映射一个组件
    routes:({
      path: '/home',
      component: Home
    },
    {
      path: '/about',
      component: About
    }
  )
})
export default router;
```

main.js

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'

Vue.config.productionTip = false

new Vue({
  // 4.挂载根实例 从而让整个应用都有路由功能
  router,
  render: h => h(App)
}).$mount('#app')

```

做好以上配置之后

在App.vue

```
<template>
  <div id="app">
    <div id="nav">
      <!-- 使用router-link组件来导航 -->
      <!-- 通过传入to属性指定连接 -->
      <!-- router-link默认会被渲染成一个a标签 -->
      <router-link to="/">首页</router-link> |
      <router-link to="/about">关于我</router-link> |
    </div>
    <!-- 路由出口 -->
    <!-- 路由匹配的组件将被渲染到这里 -->
    <router-view/>
  </div>
</template>
```

打开浏览器.,切换Home和About超链接,查看效果

命名路由

在配置路由的时候,给路由添加名字,访问时就可以动态的根据名字来进行访问

```
const router = new Router({
    routes:({
      path: '/home',
      name:"home",
      component: Home
    },
    {
      path: '/about',
      name:'about'
      component: About
    }
  )
})
```

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:'home'}">Home</router-link> |
<router-link :to="{name:'about'}">About</router-link> |
```

动态路由匹配

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。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。那么，我们可以在 `vue-router` 的路由路径中使用“动态路径参数”(dynamic segment) 来达到这个效果

User.vue

```
<template>
  <div>
    <h3>用户页面</h3>
  </div>
</template>

<script>
export default {
};
</script>

<style lang="scss" scoped>
</style>
```

路由配置

```
const router = new Router({
    routes:(
        {
            path: '/user/:id',
            name: 'user',
            component: User,
        },
    )
})
```

```
<router-link :to="{name:'user',params:{id:1}}">User</router-link> |
```

访问

http://localhost:8080/user/1

http://localhost:8080/user/2

查看效果

当匹配到路由时,参数值会被设置到this.$route.params,可以在每个组件中使用,于是，我们可以更新 `User` 的模板，输出当前用户的 ID：

```
<template>
  <div>
    <h3>用户页面{{$route.params.id}}</h3>
  </div>
</template>
```

响应路由参数的变化

提醒一下，当使用路由参数时，例如从 `/user/1 导航到` /user/2`，**原来的组件实例会被复用**。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。**不过，这也意味着组件的生命周期钩子不会再被调用**。

复用组件时，想对路由参数的变化作出响应的话，你可以简单地 watch (监测变化) `$route` 对象：

```
/*使用watch(监测变化) $route对象
 watch: {
        $route(to, from) {
            console.log(to.params.id);

        }
    }, */
// 或者使用导航守卫
beforeRouteUpdate(to,from,next){
    //查看路由的变化
    //一定要调用next,不然就会阻塞路由的变化
    next();
}
```

404路由

创建 `Bad404.vue` 组件

```
<template>
  <div>
    404 Bad Request
  </div>
</template>

<script>
  export default {

  }
</script>

<style lang="scss" scoped>

</style>
```

main.js

```
import BadRequest from '@/views/Bad404'
const router = new Router({
    routes:(
        //....
        // 匹配不到理由时,404页面显示
        {
            path: '*',
            component: BadRequest
        }
    )
})
```

当使用*通配符*路由时，请确保路由的顺序是正确的，也就是说含有***通配符的路由应该放在最后。***路由 `{ path: '*' }` 通常用于客户端 404 错误

匹配优先级

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。**

查询参数

类似像地址上出现的这种:http://localhos:8080/page?id=1&title=foo

```
import Page from '@/views/Page'
const router = new Router({
    routes:(
        //....
        {
            name:'/page',
            name:'page',
            component:Page
        }

    )
})
```

```
 <router-link :to="{name:'page',query:{id:1,title:'张三'}}">页面1</router-link> |
 <router-link :to="{name:'page',query:{id:2,title:'李四'}}">页面2</router-link> |
```

访问http://localhos:8080/page?id=1&title=张三查看Page

访问http://localhos:8080/page?id=2&title=李四查看Page

Page.vue

```
<template>
    <div>
        <h3>Page页面</h3>
        <h3>{{$route.query.id}}--{{$route.query.title}}</h3>
    </div>
</template>

<script>
    export default {
        created () {
            //查看路由信息对象
            console.log(this.$route);
        },
    }
</script>

<style lang="scss" scoped>

</style>
```

路由重定向和别名

例子是从 `/` 重定向到 `/home` :

```
const router = new Router({
    mode: 'history',
    routes: (
        // 重定向
        {
            path: '/',
            redirect: '/home'
        }
        {
            path: '/home',
```

```
        name: 'home',
        component: Home
      },
    )
})
```

重定向的目标也可以是一个命名的路由：

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

编程式导航

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

> **注意：在 Vue 实例内部，你可以通过 $router 访问路由实例。因此你可以调用 this.$router.push。**

声明式	编程式
`<router-link :to="...">`	`router.push(...)`

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.$router.push({ path: 'home' })

// 命名的路由
this.$router.push({ name: 'user', params: { userId: '123' }})

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

前进后退

```
// 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，那就默默地失败呗
router.go(-100)
router.go(100)
```

演示：在App.vue

```
<template>
  <div id="app">

    <header>
      <!--.....-->
      <button @click='handleClick1'>跳转用户1页面</button>
```

```
    <button @click='handleClick2'>跳转页面1</button>
    <button @click='goBack'>后退</button>
    <button @click='refresh'>刷新</button>
   </header>
   <router-view>
    <!-- 路由组件渲染的出口 -->
   </router-view>
  </div>
</template>

<script>


export default {
  name: 'App',
  methods: {
    handleClick1() {
     // 命名的路由
     this.$router.push({
       name:'user',
       params:{
         id:1
       }
     })
    },
    handleClick2(){
     // 带查询参数
     this.$router.push({
       name:"page",
       query:{
         id:1,
         title:'hello router'
       }
     })
    },
    goBack(){
     // 后退一步
```

```
      this.$router.go(-1);
    },
    // 刷新
    refresh(){
      this.$router.go(0);
    }
  },
  components: {
    // HelloWorld
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

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

```
/user/1/profile                     /user/1/posts
+------------------+                +----------------+
| User             |                | User           |
| +--------------+ |                | +------------+ |
| | Profile      | |  +-----------> | | Posts      | |
| |              | |                | |            | |
| +--------------+ |                | +------------+ |
+------------------+                +----------------+
```

router.js

```
import User from '@/views/User/User'
import Profile from '@/views/User/Profile'
import Posts from '@/views/User/Posts'

//配置
{
    path: '/user/:id',
    name: 'user',
    component: User,
    children:{
      //默认加载
      {
        path:'/',
        component:Profile
      },
       // 当 /user/:id/profile 匹配成功，
      // Profile 会被渲染在 User 的 <router-view> 中
      {
        path:"profile",
        component: Profile
      },
      // 当 /user/:id/posts 匹配成功，
      // Posts 会被渲染在 User 的 <router-view> 中
```

```
      {
        path: "posts",
        component: Posts
      }
    )

}
```

在 `User` 组件的模板添加一个 `<router-view>` :

```
<template>
  <div>
    <h3>用户界面</h3>
    <div class="userInfo">
      <p>用户名:{{userInfo.name}}</p>
      <p>年龄:{{userInfo.age}}</p>
    </div>

    <!-- 演示嵌套路由 -->
    <router-link to="/user/1/profile">profile</router-link> |
    <router-link to="/user/1/posts">posts</router-link>
    <hr>
    <!-- 嵌套路由中 使用命名路由 -->
    <router-link :to='{name:"profile"}'>profile</router-link>  |
    <router-link :to='{name:"posts"}'>profile</router-link>
    <router-view></router-view>
  </div>
</template>
```

一个网站一般都会有登录操作,做如下配置

App.vue

```
<router-link :to="{name:'login'}">登录</router-link>
```

Login.vue

```vue
<template>
  <div>
    登录组件
    <button @click='handleLogin'>登录</button>
  </div>
</template>

<script>
  export default {
    methods: {
      handleLogin() {
        console.log('登录操作')
      }
    },
  }
</script>

<style lang="scss" scoped>

</style>
```

路由配置中

```
{
    path: '/login',
    name: 'login',
    component: Login,
}
```

这个时候大家会发现，如果我们点击登录按钮跳转登录组件的时候，我们的导航栏还会存在,正常情况下会打开一个新的组件页面。更多考虑的是路由配置的问题。根据之前讲解的嵌套路由，做如下修改

新建views/Index.vue,将App.vue的代码进行迁移

```
<template>
  <div>
    <header>
        <!-- 命名路由 -->
      <router-link :to='{name:"home"}'>首页</router-link> |
      <router-link to='/about'>关于我</router-link> |
      <router-link to='/user/1'>用户1</router-link> |
      <router-link to='/user/2'>用户2</router-link> |
      <router-link to='/cart'>我的购物车</router-link> |
      <router-link :to='{name:"page",query:{id:1}}'>页面1</router-link> |
      <router-link :to='{name:"page",query:{id:2}}'>页面2</router-link> |

      <router-link :to='{name:"login"}'>登录</router-link>

    <router-view></router-view>
  </div>
</template>

<script>
  export default {

  }
</script>

<style  scoped>

```

```
27
28  </style>
```

路由配置进行修改

```
1   import Vue from 'vue'
2   import App from './App.vue'
3   // 1.引入路由模块
4   import VueRouter from 'vue-router'
5   import Home from '@/views/Home'
6   import About from '@/views/About'
7   import User from '@/views/User/User'
8   import Profile from '@/views/User/Profile'
9   import Posts from '@/views/User/Posts'
10  import BadRequest from '@/views/Bad404'
11  import Page from '@/views/Page'
12  import Login from '@/views/Login'
13  import Index from '@/views/Index'
14  import Cart from '@/views/Cart'
15
16  // 2.基于模块化实现 一定要调用Vue.use() 让Vue使用该路由
    插件
17  // 这样以后所以的子组件都可以获取路由实例中属性和方法
18  Vue.use(VueRouter);
19
20  const router = new VueRouter({
21
22      routes: [
23          // 路由重定向
24          {
25              path:'/',
26              redirect:'/home',
27              component:Index,
28              children:[
29                  {
30                      path: "/home",
```

```
            name: "home",
            component: Home,

        },
        {
            path: "/about",
            name: 'about',
            component: About
        },
        // 动态路由匹配配置
        {
            path: '/user/:id',
            name: 'user',
            component: User,
            children:[
                // 默认加载 如果有默认的子路由 父路由中尽量不要使用命名路由，即使使用命名路由也会加载子路由
                // 否则控制太会报警告
                // {
                //   path:'/',
                //   component:Profile
                // },
                {
                    path:"profile",
                    name:'profile',
                    component:Profile
                },
                {
                    path: "posts",
                    name: 'posts',
                    component: Posts
                }
            ]
        },
        {
            path: "/cart",
            name: 'cart',
```

```
                component: Cart
            },
            //查询参数配置
            {
                path: '/page',
                name: 'page',
                component: Page,
            },
        )
    },



    {
        path: '/login',
        name: 'login',
        component: Login,
    },
    {
        path: "*",
        component: BadRequest
    }
  )
})




new Vue({
  // 3.一定要挂载
  router,
  render: h => h(App),
}).$mount('#app')


```

导航守卫

“导航”表示路由正在发生改变。

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离开守卫。
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. 用创建好的实例调用 `beforeRouteEnter` 守卫中传给 `next` 的回调函数。

全局守卫

你可以使用 `router.beforeEach` 注册一个全局前置守卫

```
const router = new VueRouter({ ... })

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

有个需求,用户访问在浏览网站时,会访问很多组件,当用户跳转到 `/cart` ,发现用户没有登录,此时应该让用户登录才能查看,应该让用户跳转到登录页面,登录完成之后才可以查看我的购物车的内容,这个时候全局守卫起到了关键的作用

有两个路由 `/carts和` /login`

router.vue

```
import Cart from '@/views/Cart'
import Login from '@/views/Login'
const router = new VueRouter({
    routes:(
        {
            path: '/cart',
            name: 'cart',
            component: Cart
        },
        {
            path: "/login",
            name: "login",
            component: Login
        },
    )
})

// 全局守卫
router.beforeEach((to, from, next) => {
    //用户访问的是'/notes'
    if (to.path === '/cart') {
        //查看一下用户是否保存了登录状态信息
        let userInfo =
JSON.parse(localStorage.getItem('userInfo'))
        if (userInfo) {
            //如果有,直接放行
            next();
```

```
        } else {
            //如果没有,用户跳转登录页面登录
            next('/login')
        }
    } else {
        next();
    }
})
```

Login.vue

```
<template>
  <div>
    登录组件
    <button @click='handleLogin'>登录</button>
  </div>
</template>

<script>
  export default {
    methods: {
      handleLogin() {
        setTimeout(() => {

          let data = {
            ok:1,
            msg:"登录成功",
            userInfo:{
              name:'小马哥'
            }
          }

localStorage.setItem('userInfo',JSON.stringify(data.userInfo)
);


```

```
        // 跳转首页
        this.$router.push({
          path:'/'
        })

      }, 1000);
    }
  },
 }
</script>

<style lang="scss" scoped>

</style>
```

views/Index.vue

```
<!-- 全局守卫演示 -->
<div class="userLogin" v-if='!userInfo'>
   <router-link :to='{name:"login"}'>登录</router-link> |
</div>
<div v-else>
  {{userInfo.name}}
</div>
<button @click='handleLogout'>退出登录</button>
```

```
export default {
   data() {
      return {
         userInfo:
JSON.parse(window.localStorage.getItem('userInfo')) || 
null
      }
   },
```

```
  methods: {
    handleLogout() {
     setTimeout(() => {
         window.localStorage.removeItem('userInfo');
          // 刷新
         this.$router.go(0);
       }, 500);
    }
  },
}
```

路由元信息实现权限控制

给需要添加权限的路由设置meta字段

```
{
    path: '/cart',
    name: 'cart',
    component: () => import('@/views/Cart'),
    meta: {
      requirAuth: true
}
},
{
    // 路由独享的守卫
    path: '/notes',
    name: 'notes',
    component: () => import('@/views/Notes'),
    meta: {
      requirAuth: true
    }
},
```

```
// 全局守卫
router.beforeEach((to, from, next) => {
  if (to.matched.some(record =>
record.meta.requiresAuth)) {
    // 需要权限
    if(!localStorage.getItem('userInfo')){
      next({
        path:'/login',
      })
    }else{
      next();
    }

  } else {
    next();
  }
})
```

login.vue

```
//登录操作
handleLogin() {
    // 1.获取用户名和密码
    // 2.与后端发生交互
    setTimeout(() => {
          let data = {
            ok:1,
            msg:"登录成功",
            userInfo:{
              name:'小马哥'
            }
          }
        localStorage.setItem("userInfo",
JSON.stringify(data.userInfo));
        // 跳转到之前的页面
        this.$router.push({path: this.$route.query.redirect });
```

```
16      }, 1000);
17  }
```

学前端顾虑、问题等

1. 应届毕业生想要找前端工作需要会什么？

百科业务部_Web前端研发工... / 2k-3k

职位描述：

工作职责：
-负责百科百科产品及服务的Web端功能设计、开发和实现
-根据产品需求，负责技术的可行性研究及技术选型
-结合开发中发现的问题，对产品提出合理建议
-探索Web前端新技术并运用到产品开发中

职位要求：
-全日制计算机及相关专业毕业，本科及以上学历
-精通HTML、CSS、JS，熟悉页面架构和布局，对表现与数据分离、Web语义化等有深刻理解，熟悉HTML5/CSS3等常用技术
-有扎实的计算机和网络基础，熟悉面向对象化编程思想和设计模式
-有一定的前端框架设计经验，熟悉Web标准，了解用户交互的基本习惯和基本的优化
-有React/Angular/Vue或其它前端框架的使用经验
-熟悉ES6规范及语法，熟悉FIS/Webpack/Gulp等模块化、前端编译和构建工具
-了解至少一种后端语言
-良好的沟通与表达能力、思路清晰，较强的动手能力与逻辑分析能力

工作地址

北京 - 海淀区 - 西二旗 - 百度大厦　　查看地图

字节跳动招聘

Web前端开发实习生-企业级平台

6k-8k /北京 / 经验不限 / 本科及以上 / 实习

web前端

17:35 发布于拉勾网

职位诱惑：

弹性工作，免费三餐，租房补贴

职位描述：

职位职责：
1、对接字节跳动内部系统业务，负责前端开发工作；
2、优化前端功能设计，通过技术手段，提升用户体验并满足高性能要求；
3、充分了解需求，独立做任务细分和工作量评估，按时保质保量完成任务；
4、现有系统的体验优化和规范化。

职位要求：
1、本科及以上学历，计算机相关专业优先；
2、了解常用的数据结构和算法，可阅读英文技术文档；
3、可以脱离框架开发，了解div+css布局，ajax，前端性能优化方法；
4、具备MVVM框架开发经验（如React、Vue、AngularJS）加分；
5、学习能力强，善于发现问题，善于总结；
6、良好的沟通和团队协作能力、做事主动；责任心强、承诺必达。

工作地址

北京 - 海淀区 - 知春路118号知春大厦A座 查看地图

Web前端开发（北京）

8k-16k /北京 / 经验应届毕业生 / 本科及以上 / 全职

前端开发 Javascript Node.js

16:05 发布于拉勾网

职位诱惑：

完善福利；牛咖组队；1v1导师制度

职位描述：

岗位职责：

1. 参与或负责核心产品的前端开发工作，保障项目质量；

2. 研究前沿技术和创新思路，并让合适的技术应用到产品中去；

任职要求：

1. 计算机相关专业本科及以上学历，或非计算机相关专业、但是业余自修过计算机专业的所有必修课；

2. 熟悉各种Web前端技术，包括HTML(5)/CSS(3)/Javascript等，并有相关的项目开发经验或成果；

3. 理解React框架思想，并且至少了解一种后端语言，如Node.js、Python、Go等；

4. 对常见算法、数据结构有一定了解，热爱计算机编程，乐于持续学习和挑战新技术。

如果你是动画达人、像素眼、架构师、NPM狂魔等等，痴情于前端的某个子领域的优秀人才，可以忽视以上要求，请尽情来玩！

工作地址

北京 - 海淀区 - 蓝靛厂路金源时代商务中心B区写字楼805-806室 查看地图

2. 大二、大三的学生要不要现在就开始报班?

   花满意的钱 学更有价值的课

   更多的建议

   我就是这样过来，培训少走很多弯路，认识很多同行业的同学。

   自己看文档

   小马哥

   18年年初~19年年底。前端进阶 =》全栈的课

   没有前端经验 我不要

   js基础的同学 报名 送JavaScript从入门到进阶 我是你们的导师 徒弟

7999 交200押金 **集训营结束之前**缴清尾款 200抵消2000。 5999

支持分期 花呗分期

6000

5个月

1200

12个月 1个月 500

货比三家

50人

3. 转行发展前景
4. 女生适合做开发嘛？加班非常严重怎么办?

   薪水 2天

   后面买上车

   老家买上房