Vue-router

1. 起步

1.1 页面引入

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

<!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
<router-link to="/foo">Go to Foo</router-link>

<!-- 也可以绑定to -->
<router-link :to="...">
```

1.2 js引入

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

// 1. 可以从其他文件 import 进来
const Foo = { template: '<div>foo</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。
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3. 创建 router 实例，然后传 `routes` 配置
const router = new VueRouter({
  routes // (缩写) 相当于 routes: routes
})

// 4. 创建和挂载根实例。
const app = new Vue({
  router
}).$mount('#app')
```

- 添加路由器后，可以通过this获取相关设置

  `this.$router` 访问路由器；`this.$route` 访问当前路由

2. 易忘点

- 当 `<router-link>` 对应的路由匹配成功，将自动设置 class 属性值 `.router-link-active`
- 路由匹配**优先级**：路由的`定义顺序`

3. 用法进阶

3.1 动态路由

```
// 动态路径参数 以冒号开头
// 参数(id)值会被设置到 this.$route.params
{ path: '/user/:id', component: User }

// 设置多段“路径参数”
{ path: '/user/:username/post/:post_id', component: User }
```

- 从 `/user/foo` 导航到 `/user/bar`，**原来的组件实例会被复用，这也意味着组件的`生命周期钩子`不会再被调用**。
- 监控路由变化

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

  或者：

  ```
  const User = {
    template: '...',
    beforeRouteUpdate (to, from, next) {
      // react to route changes...
  ```

```
    // don't forget to call next()
  }
}
```

3.2 高级匹配模式

> `vue-router` 使用 path-to-regexp 作为路径匹配引擎，所以支持很多高级的匹配模式

3.3 路由配置-children

3.4 编程式路由

3.4.1 push

> `router.push(location, onComplete?, onAbort?)`

```
// 字符串
router.push('home')

// 对象
router.push({ path: 'home' })

// 命名的路由
// 如果希望在组件中使用this.$router.params，就必须使用命名路由
router.push({ name: 'user', params: { userId: 123 }})

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

> 注意：如果提供了 **path**，**params** 会被忽略，上述例子中的 **query** 并不属于这种情况。取而代之的是下面例子的做法，你需要提供路由的 **name** 或手写完整的带有参数的 **path**：

```
const userId = 123
```

```
router.push({ name: 'user', params: { userId }}) // -> /user/123
router.push({ path: `/user/${userId}` }) // -> /user/123
// 这里的 params 不生效
router.push({ path: '/user', params: { userId }}) // -> /user
```

> 在 `router.push` 或 `router.replace` 中提供 `onComplete` 和 `onAbort` 回调作为第二个和第三个参数：将会在 `导航成功` 完成 (在所有的异步钩子被解析之后) 或 `终止` (导航到相同的路由、或在当前导航完成之前导航到另一个不同的路由) 的时候进行相应的调用

3.4.2 replace

> `router.replace(location, onComplete?, onAbort?)`

3.4.3 go

> `router.go(n)`

```
// 如果 history 记录不够用，那就默默地失败呗
router.go(-100)
router.go(100)
```

3.4.4 其他

> Vue Router 的导航方法 (`push` 、 `replace` 、 `go`) 在各类路由模式 (`history` 、 `hash` 和 `abstract`) 下表现一致。

3.5 命名路由

3.6 命名视图

个人觉得命名视图有些鸡肋：貌似可以直接使用子组件代替；

```
// default-默认视图出口
components: { default: User, sidebar: Sidebar }
```

3.7 重定向

```
{ path: '/a', redirect: '/b' }

// 也可以是命名路由
{ path: '/a', redirect: { name: 'foo' }}
```

> 注意导航守卫并没有应用在跳转路由上，而仅仅应用在其目标上。在上面这个例子中，为 `/a` 路由添加一个 `beforeEach` 或 `beforeLeave` 守卫并不会有任何效果。

3.8 别名

```
// 访问"/hello"会跳转到"/"
{
  path: '/',
  name: 'HelloWorld',
  meta: {
    title: 'HelloWorld'
  },
  component: HelloWorld,
  alias: '/hello'
}
```

3.9 使用 `props` 将组件和路由解耦 - ✈

```
{
  path: '/user/:id',
  components: { default: User, sidebar: Sidebar },
  // 重点
  // 如果 props 被设置为 true，route.params 将会被设置为组件属性
  props: { default: true, sidebar: false }
}
```

3.10 关于404

可以设置一个通用界面（404页面），当匹配不到更加精确的页面时，就会匹配到通用页面：

```
{path: '*', name: '_404', component: _404}
```

3.11 导航守卫

3.11.1 全局前置守卫 - ✈

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

- **next**

  一定要调用该方法来 **resolve** 这个钩子；可以有多种调用形式：**next()**、**next(false)**、**next('/')**、**next({ path: '/' })**、**next(error)**

3.11.2 其他全局守卫

- `router.beforeResolve`
- `router.afterEach`

3.11.3 非全局守卫

- `beforeEnter`

```
  {
      path: '/foo',
      component: Foo,
      beforeEnter: (to, from, next) => {
        // ...
      }
    }
```

- 组件内守卫

```
const Foo = {
  template: `...`,
  beforeRouteEnter (to, from, next) {},
  beforeRouteUpdate (to, from, next) {},
  beforeRouteLeave (to, from, next) {}
}
```

关于组件内守卫的this问题

4. 填坑

4.1 组件复用

> 如果目的地和当前路由相同，只有参数发生了改变 (比如从一个用户资料到另一个 `/users/1` -> `/users/2`)，你需要使用 `beforeRouteUpdate` 来响应这个变化 (比如抓取用户信息)