vue-router

路由

1. 开始

HTML

```html
<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>
    <!-- use router-link component for navigation. -->
    <!-- specify the link by passing the `to` prop. -->
    <!-- `<router-link>` will be rendered as an `<a>` tag by default -->
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>
  </p>
  <!-- route outlet -->
  <!-- component matched by the route will render here -->
  <router-view></router-view>
</div>
```

JS

```js
// 0. If using a module system (e.g. via vue-cli), import Vue and VueRouter
// and then call `Vue.use(VueRouter)`.

// 1. Define route components.
// These can be imported from other files
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }

// 2. Define some routes
// Each route should map to a component. The "component" can
// either be an actual component constructor created via
// `Vue.extend()`, or just a component options object.
// We'll talk about nested routes later.
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar', component: Bar }
]

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = new VueRouter({
  routes // short for `routes: routes`
})

// 4. Create and mount the root instance.
// Make sure to inject the router with the router option to make the
// whole app router-aware.
const app = new Vue({
  router
}).$mount('#app')

// Now the app has started!
```

路由注册后, 我们就可以在vue的实例对象中通过
this.$router 来访问 路由对象router ,
this.$route 来访问 路由参数信息

2. 动态路由匹配

1. 注册路由的时候, ":"绑定动态参数名

```js
const User = {
  template: '<div>User</div>'
}

const router = new VueRouter({
  routes: [
    // dynamic segments start with a colon
    { path: '/user/:id', component: User }
  ]
})
```

2. 在对应的组件中, 参数会放在route对象的params属性中(切记, 是route, 不是router)

```js
const User = {
  template: '<div>User {{ $route.params.id }}</div>'
}
```

3. 多动态路由参数

pattern	matched path	$route.params
/user/:username	/user/evan	`{ username: 'evan' }`
/user/:username/post/:post_id	/user/evan/post/123	`{ username: 'evan', post_id: '123' }`

4. 检测动态参数改变 监听 `beforeRouteUpdate` 函数

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

5. 匹配通用路由

1. 全部通配

```
Error = () => import('./components/404.vue'),
router = new VueRouter({
    'routes': [
        { 'path': '*', 'component': Error }
    ]
});
```

2. 部分通配

```
User = () => import('./components/user.vue'),
router = new VueRouter({
    'routes': [
        { 'path': '/user-*', 'component': User },
    ]
});
```

3. 通配符得到的信息在参数 $route.params.pathMatch 中

```
<div>
    我是user-{{$route.params.pathMatch}}界面
</div>
```

3. 层级路由

路由之中嵌套子路由
比如, 用户 user路由中想嵌套不同的自己路由, 用来显示不同的用户名称

```
<div class="user">
  <h2>User {{ $route.params.id }}</h2>
  <router-view></router-view>
</div>
```

在user路由中增加自己路由: children

```
const router = new VueRouter({
  routes: [
    { path: '/user/:id', component: User,
      children: [
        {
          // UserProfile will be rendered inside User's <router-view>
          // when /user/:id/profile is matched
          path: 'profile',
          component: UserProfile
        },
        {
          // UserPosts will be rendered inside User's <router-view>
          // when /user/:id/posts is matched
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
})
```

4. 函数式编程

```
// literal string path
router.push('home')

// object
router.push({ path: 'home' })

// named route
router.push({ name: 'user', params: { userId: '123' } })

// with query, resulting in /register?plan=private
router.push({ path: 'register', query: { plan: 'private' } })
```

5. 给路由命名

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

用 router-link 通过路由名 来跳转到该路由

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

用函数式编程, 通过 路由名 来跳转到该路由

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

6. 给router-view 命名
注意: 是components, 不是 component

```
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>
```

```
const router = new VueRouter({
  routes: [
    {
      path: '/',
      components: {
        default: Foo,
        a: Bar,
        b: Baz
      }
    }
  ]
})
```

7. 路由重定向和别名

1. 通过路径重定向

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

2. 通过路由名 name 重定向

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

3. 通过函数 进行动态编程 重定向

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: to => {
      // the function receives the target route as the argument
      // return redirect path/location here.
    }}
  ]
})
```

4. 路由别名

```
const router = new VueRouter({
  routes: [
    { path: '/a', component: A, alias: '/b' }
  ]
})
```

8. 路由传值

1. 参考2, 动态路由传值

2. 路由属性传值

对象模式传值

```
const router = new VueRouter({
  routes: [
    { path: '/promotion/from-newsletter', component: Promotion, props: { newsletterPopup: false } }
  ]
})
```

函数模式传值

```
const router = new VueRouter({
  routes: [
    { path: '/search', component: SearchUser, props: (route) => ({ query: route.query.q }) }
  ]
})
```