07.11 【课件】react-router 和 vue-router

个人简介

大家可以称呼我奥利奥老师。曾就职于腾讯，百度等互联网企业，在 PC 端，移动端，to B， to C 业务均有丰富的开发经验。

擅长领域

1. 专题活动页面 PC/H5 开发；
2. 小程序开发；
3. B 端管理系统；
4. 网站性能优化；
5. 代码设计与优化；
6. 组件库的研发；

本节课的主题

react-router 和 vue-router 的实现原理解析，以及 react-router 使用方式详解。

底层核心原理

1. History 路由模式下，均基于 html5 新增的 history API，`pushState`，`replaceState`，其用法如下：

```
history.pushState(state, title, url);
history.replaceState(state, title, url);
```

即：跳转到 url 路径（与当前页面处在同一个域，形如一个网站的 location.pathname 部分），指定新页面的标题 title，但是浏览器目前都忽略这个值，因此这里一般使用 null，state 为关联新地址的状态对象（刷新页面并不会丢失）。二者的区别就是，pushState 会增加一条浏览器记录，而 replaceState 会替换当前历史记录。相同点是，**均不会刷新当前页面，也不会发生真正的跳转**，而是仅仅改变了地址栏的 URL（history、location 对象）。

2. Hash 路由模式下，基于 location.hash = pathString 来更新网站路径。pathString 代表网址中 # 号后面直到 search 的部分。与 history 不同的是，如果两次赋值一样的时候，并不会触发 hashchange 和 popstate 方法。

3. 在非浏览器环境，使用抽象路由实现导航的记录功能，如 react-router 的 memoryHistory，vue-router 的 abstract 封装。

vue-router 的原理

我们从使用方式入手：+

```
import Vue from 'vue';
import VueRouter from 'vue-router';
Vue.use(VueRouter);
const router = new VueRouter({
  mode: 'history',
  routes: [
      { path: '/app', component: App },
      {
        path: '/home',
        component: Home,
        children: [{
            path: 'home-sub',
            component: () => import('./HomeSub')
        }]
      }
    ]
});
```

```
new Vue({
  router
}).$mount('#app');

// template 内部
<div id="app">
  这里是公共内容，不属于任何路由的一部分
  <router-view></router-view>
</div>
...
```

我们知道，Vue.use 方法接收一个实现了 install 方法的对象（或类）作为参数，例如注册一个全局组件 Modal，假设我们已经实现了 modal.vue，那么

```
import Modal from './modal.vue';

const MyModal = {
    install(Vue){ // install 会被传入 Vue 参数
       Vue.component('Modal', Modal);
    }
  }
Vue.use(MyModal);
// 这样就可以在 template 中直接使用 <Modal></Modal> 了。
```

按照这个思路，我们来验证一个问题，在 VueRouter 根文件中：

```
import { install } from './install'

...

export default class VueRouter {

  static install: () => void

  ...

}

VueRouter.install = install
```

追溯到 install.js

```
import View from './components/view'

import Link from './components/link'

export function install(Vue) {

...

  Vue.mixin({

    beforeCreate() { // 这里使得 _route 属性具备响应能力

     Vue.util.defineReactive(this, '_route', this._router.history.current)

    },

    destroyed() {}

  })

}

...

Object.defineProperty(Vue.prototype, '$router', {

  get () { return this._routerRoot._router }

})
```

```
Object.defineProperty(Vue.prototype, '$route', {

  get () { return this._routerRoot._route }

})

// 走到下面两行代码已经水落石出了，这也是我们引入 vue-router 后

// 就能全局使用 <router-view> 和 <router-link> 的原因

Vue.component('RouterView', View)

Vue.component('RouterLink', Link)

}
```

不难得出结论，vue-router 是 Vue 应用的一个全局组件，一次注册便可以处处使用，且所有的 vue 实例都能访问到 Vue 原型对象上的 $router 和 $route 对象，因此在使用函数式跳转时是及其方便的。

那我们继续，在实例化 VueRouter 时，有 routes 和 mode 参数

```
const router = new VueRouter({

 mode: 'history',

 routes: []

})
```

进入 VueRouter 的构造器 constructor

```
constructor (options: RouterOptions = {}) {

...

this.fallback =

  mode === 'history' && !supportsPushState && options.fallback !== false

if (this.fallback) {

  mode = 'hash'
```

```
}

switch (mode) {

  case 'history':

    this.history = new HTML5History(this, options.base)

    break

  case 'hash':

    this.history = new HashHistory(this, options.base, this.fallback)

    break

  case 'abstract':

    this.history = new AbstractHistory(this, options.base)

    break

  }

}
```

可以看出，基于三种 mode，vue-router 分别用三个对象来初始化 this.history：HTML5History，HashHistory，AbstractHistory

而后续的几类路由跳转（go，push...）方法，也是调用了 this.history 的方法。源码方面不再赘述。我们可以根据原理实现一个简单的路由：

```
npm i -g @vue/cli 或 yarn global add @vue/cli

vue create vue-app
```

App.vue

```
<script>

import Layout from './components/Layout.vue'

import Home from './Home.vue'
```

```
import About from './About.vue'
import NoMatch from './components/404.vue'
const routes = {
  '/home': Home,
  '/about': About,
};
export default {
  name: 'App',
  data() {
    return {
      activeRoute: location.pathname
    }
  },
  computed: {
    computedView() {
      const Component = routes[this.activeRoute]
      return Component || NoMatch
    }
  },
  render(h) {
    return h(Layout, {}, [h(this.computedView)])
  }
}
```

```
</script>
```

Vue-router 的核心原理大概如此，我们接着学习 react-router。

react-router：不得不从 context 说起

首先安装一下：

```
yarn global add react-react-app
react-react-app react-app
yarn add react-router react-router-dom
```

react-router-dom 基于 react-router 封装，推荐使用 dom 形式路由，只安装后者也可。

react-router 为组件注入的方法不像 Vue 那样，将 router 的一些实例方法挂载到全局，组件可以通过 this.$router，this.$route 访问到。因此 react-router 的实现要结合 context API 来讲。顺便了解下 react 一贯推崇的 FC 编程的哲学。

```
import { createContext, useState, useContext } from 'react';
const Context = createContext();
function Parent() {
  const [value, setValue] = useState(1);
  const providerValue = { value, setValue };
  return <Context.Provider value={providerValue}>
    <SubComponent1 />
    <SubComponent2 />
  </Context.Provider>
}
function SubComponent1() {
  return <Context.Consumer>
```

```
    { ({ value, setValue }) =>  <div>
        { value }
        <button
            onClick={() => setValue(value + 1)}
        >点击增加</button>
    </div> }
  </Context.Consumer>
}
// 或者
function SubComponent2() {
  const { value, setValue } = useContext(Context);
  return <div>
      { value }
      <button onClick={() => setValue(value - 1)}>点击减少</button>
</div>
}
```

Context 避免了组件层层传递 props 的问题，实现了父级向后代组件通信的能力。鉴于此，我们可以探秘 react-router 映射的组件为何比常规（我们自己实现的）组件多出来一些路由方法，以及如何使常规组件拥有这些方法。

```
import { __RouterContext as Context } from 'react-router'
export default WraperApp(props) {
  return <Context.Consumer>
      { context => <App {...context} {...props} /> }
```

```
  </Context.Consumer>
}
// 因此可以直接使用 withRouter
import { withRouter } from 'react-router'
const WraperApp = withRouter(App);
```

Context 在 react-redux, mobx-react 库中依然是一个很重要的底层 API。掌握了它，后面的理解将会越来越容易！