day17笔记

1、react-router-dom

- react-router 核心库
- **react-router-dom DOM操作路由库**
- react-router-native ReactNative 的路由库
- 文档地址：https://reactrouter.com/web/guides/quick-start
- 如何使用

```
npm i react-router-dom
```

2、基础语法

2.1、基础使用

- 路由模式组件必须是顶层组件（**所有的路由组件都必须在路由模式组件包裹范围之内！！！**）
- 路由的配置

```
<HashRouter>
  <Header/>
  <div className="container">
    <Switch>
        <Route path="/" exact component={Index} ></Route>
        <Route path="/about" component={About} ></Route>
        <Route path="/news" exact >
            <News />
        </Route>
        <Route path="/news/:id" component={Info} ></Route>
        <Route path="/user"  component={User} ></Route>
        <Route path="/404">
            <Notfound />
        </Route>
        {/* 404的配置，一定要放在最下方！！！  */}
        <Route path="/*">
            <Redirect to='/404' />
        </Route>
    </Switch>
  </div>
</HashRouter>
```

- 常用组件
  - `BrowserRouter` history模式
  - `HashRouter` history模式
  - `Route` 路由映射组件
    - path
    - component
    - exact 严格模式
  - `Switch` 匹配到第一个匹配就跳出！
  - `NavLink` 导航链接，有激活class，默认为active！
    - exact
    - to
  - `Link` 普通链接
    - to
      - `to='/地址'`
      - `to={pathname:'/地址',search:'?a=b&c=d',hash:'#e=f&g=h'}`
  - `Redirect`
    - to 重定向

2.2、动态路由

- 配置路由

```
<Route path="/news/:id" component={Info} ></Route>
```

- 设置链接

```
<ul>
  <li><Link to='/news/111'>新闻111</Link></li>
  <li><Link to='/news/222'>新闻222</Link></li>
  <li><Link to='/news/333'>新闻333</Link></li>
  <li><Link to='/news/444'>新闻444</Link></li>
</ul>
```

- 获取参数

```
// 动态路由的数据!
console.log(this.props.match.params)
```

注意：

使用Route组件建立映射关系，如果component属性指定了对应的组件！那么该组件的props里面将会包含路由相关信息！

如果不是页面组件，或者页面组件不是通过component设立的，那么可以使用`withRouter`来进行数据的设置

```
import React, { Component } from 'react';
import { withRouter } from 'react-router-dom';

class Notfound extends Component {
    render() {
        console.log(this)
        return (
            <div>
                404页面
            </div>
        );
    }
}

export default withRouter(Notfound);
```

2.3、编程式导航

- 编程式导航

```
this.props.history.go()/goBack()/goForward()/replace()/push()
```

- 如果在非组件里面的JS中使用编程式导航跳转
  - hash模式！**推荐**

```
import { createHashHistory } from "history"
const history = createHashHistory();
history.go()/goBack()/goForward()/replace()/push()
```

  - history模式！

```
// 顶级路由组件需要使用Router,且注入一个history实例，所有的跳转也是这个history实例

// App.js
import { Router,Redirect,Route,Switch } from "react-router-dom"
import history "./history.js"

<Router history={history}>
    xxxxx
</Router>
```

```
// history.js文件
import { createBrowserHistory } from "history"

const history = createBrowserHistory();

export default history;
```

任意代码位置：

```
import history "./history.js"
history.go()/goBack()/goForward()/replace()/push()
```

2.4、嵌套路由

- **嵌套路由的配置是写在对应的嵌套页面组件里面，而不是顶级**
- **嵌套路由的上级路由不能是严格模式！**

```
import React, { Component } from 'react';
import { NavLink, Route, Switch } from 'react-router-dom';

import { createHashHistory } from "history"

import UserIndex from "./UserIndex"
import UserClass from "./UserClass"
import UserOrder from "./UserOrder"
import UserInfo from "./UserInfo"

let history = createHashHistory();

class User extends Component {
    quit=()=>{
        // this.props.history.push('/')
        history.push('/')
    }
    render() {
        return (
            <div className="main">
                <div className="nav">
                    <ul>
                        <li><NavLink activeClassName="on" to='/user/class'>我的课
程</NavLink></li>
                        <li><NavLink activeClassName="on" to='/user/order'>我的订
单</NavLink></li>
                        <li><NavLink activeClassName="on" to='/user/info'>我的资料
</NavLink></li>
                    </ul>
                </div>
```

```
                <div className="box">
                    <div>
                        <div>欢迎您:<button onClick={this.quit}>退出</button></div>
                        <hr/>
                        {/* 主体内容 */}
                        <Switch>
                            <Route path="/user/" exact component={UserIndex}></Route>
                            <Route path="/user/order" component={UserOrder}></Route>
                            <Route path="/user/class" component={UserClass}></Route>
                            <Route path="/user/info" component={UserInfo}></Route>
                        </Switch>
                    </div>
                </div>
            </div>
        );
    }
}

// 嵌套路由:
    // 嵌套路由的配置是写在对应的嵌套页面组件里面，而不是顶级!
    // 嵌套路由的上级路由不能是严格模式!

export default User;
```

2.5、导航守卫

- 思路： 让所有`Route`组件渲染的时候都走同一个函数，在这个函数里面去做拦截!
- 实现:
    - 思路引导1

```
class MyRouter extends Component {
    render() {
        return (
            <Switch>
                <Route path="/" exact render={() => <Index/> } ></Route>
            </Switch>
        );
    }
}
```

    - 思路引导2

```
class MyRouter extends Component {
    renderCom=(val)=>{
```

```
        return val
    }
    render() {
        return (
            <Switch>
                <Route path="/" exact render={() =>this.renderCom(<Index/>)}
></Route>
                <Route path="/about"  render={() =>this.renderCom(<About/>)}
></Route>
                <Route path="/news"   render={() =>this.renderCom(<News/>)} >
</Route>
            </Switch>
        );
    }
}
```

使用render去渲染组件，组件当然也没有了props！可以使用withRouter来进行处理！

- **思路引导3**

```
// 路由的配置信息!
const routeConfig = [
    { path: "/", exact: true, component: <Index />, title: "首页" },
    {path:"/about",exact:false, component:<About/>,title:"关于我们"},
    { path: "/news", exact: true, component:<News/>,title:"新闻列表"},
    { path: "/news/:id", exact: false, component: <Info />, title: "新闻详情"
},
    { path: "/user", exact: false, component: <User />, title: "用户中
心",needLogin:true },
    { path: "/login", exact: false, component: <Login />, title: "登录" },
    { path: "/404", exact: false, component: <Notfound />, title: "404页面"
},
    { path: "/*", exact: false,redirect:"/404"}
]

class MyRouter extends Component {
    renderCom=(val)=>{
        document.title = val.title
        if(val.redirect){
            return <Redirect to={val.redirect} />
        }else{
            // 这里！！！  要做判断！登录判断，权限判断！！！
            if (val.needLogin){
                if (localStorage.getItem('userinfo')) {
                    return val.component;
                } else {
                    return <Redirect to="/login" />
                }
            }else{
```

```
                return val.component;
            }
        }
    }
    render() {
        return (
            <Switch>
                {/* <Route path="/" exact render={()
=>this.renderCom(<Index/>)} ></Route>
                <Route path="/about"  render={() =>this.renderCom(<About/>)}
></Route>
                <Route path="/news"   render={() =>this.renderCom(<News/>)} >
</Route> */}
                {routeConfig.map((val,idx)=>{
                    return <Route key={idx} path={val.path} exact={val.exact}
render={() => this.renderCom(val)} ></Route>
                })}
            </Switch>
        );
    }
}

export default MyRouter;
// 让所有的组件都走renderCom函数，在这个函数里面去做标题设置、权限判断、等等，符合条
件就返回对应的组件内容。
```

2.6、路由懒加载

- 核心插件：`react-loadable`
- 文档地址：https://www.npmjs.com/package/react-loadable
- 使用

```
yarn add react-loadable
// 或者
npm i react-loadable
```

```
import Loadable from 'react-loadable';
import Loading from './my-loading-component';   // 自定义等待组件

const LoadableComponent = Loadable({
  loader: () => import('./my-component'),
  loading: Loading,
});

/*
```

```
const  异步组件 = Loadable({
  loader:()=>import('组件')
  loading:等待组件
})
*/
```

示例

```
import Loadable from 'react-loadable';
import Loading from '../components/Loading';

// import Index from "../pages/Index"
// import About from "../pages/About"
// import News from "../pages/News"
// import Info from "../pages/Info"
// import User from "../pages/User"
// import Notfound from "../pages/Notfound"
// import Login from "../pages/Login"

const Index = Loadable({ loader: () => import("../pages/Index"), loading: Loading
})
const About = Loadable({ loader: () => import("../pages/About"), loading: Loading
})
const News = Loadable({ loader: () => import("../pages/News"), loading: Loading })
const User = Loadable({ loader: () => import("../pages/User"), loading: Loading })
const Notfound = Loadable({ loader: () => import("../pages/Notfound"), loading:
Loading })
const Login = Loadable({ loader: () => import("../pages/Login"), loading: Loading
})
const Info = Loadable({ loader: () => import("../pages/Info"), loading: Loading})
```