day06-路由

1. 路由

### 介绍

使用React构建的单页面应用，要想实现页面间的跳转, 想要随着url地址的改变切换显示不同的内容，首先想到的就是使用路由。在React中，常用的有两个包可以实现这个需求，那就是react-router和react-router-dom.

react-router: 实现了路由的核心功能

react-router-dom: 基于react-router，加入了在浏览器运行环境下的一些功能，例如：Link组件，会渲染一个a标签， BrowserRouter和HashRouter组件，前者使用pushState和popState事件构建路由，后者使用window.location.hash和hashchange事件构建路由。

实际应用中用react-router-dom即可,下面看具体使用

### 路由的使用

#### 安装路由库

cnpm i react-router-dom –save

#### 相关组件

BrowserRouter , HashRouter , MemoryRouter , NativeRouter , StaticRouter， Route, Link,NavLink,Switch,Redirect 这些是常用组件, 可以有选择的导入个别组件:

import { HashRouter,Route,Link,Switch } from 'react-router-dom';

**路由管理组件**

react-router-dom包含多种路由管理组件: BrowserRouter , HashRouter , MemoryRouter , NativeRouter , StaticRouter ;

通常，应用程序将使用其中一个路由管理组件.

这里,重点介绍BrowserRouter , HashRouter:

**BrowserRouter** 使用HTML5历史记录API（pushState，replaceState和popstate事件）的<Router>来保持您的UI与URL同步。

**HashRouter** 使用URL的哈希部分（即window.location.hash）的<Router>来保持您的UI与URL同步

**路由组件**

Switch <Switch>的所有子项应为<Route>或<Redirect>元素,渲染与位置匹配的第一个子元素<Route> 或 <Redirect> , <Switch>是独特的，因为它仅仅渲染一个路由。相反，与位置匹配的每个<Route>都会渲染.

Route 元素尝试将其路径与当前历史记录位置（通常是当前浏览器URL）进行匹配,渲染与位置匹配的组件; 如果路由没有路径，将始终匹配;

Redirect 渲染<Redirect>将导航到新位置。新位置将覆盖历史堆栈中的当前位置;

Link 最终渲染成a标签

NavLink 最终渲染成a标签, 会自动维护a标签的active类的添加/移除( 当浏览器地址栏的地址和哪个a标签的路由地址相匹配时, 该a标签会自动添加active类, 其他不匹配的a标签会自动移除active类 ).

#### 声明式导航

<Link> 和 <NavLink> 实现路由的跳转 是声明式导航

<Navlink to={ 你要跳转的页面 }>

#### 编程式导航

通过js路由对象的方式实现路由的跳转叫做编程式导航, 主要用到以下三个路由对象: history, location, match

props.history.push() //跳转路由

props.history.replace()

props.history.go() //前进 or 后退

props.history.goBack() //后退

props.history.goForward() //前进

注意: 被Route处理过之后或者用withRouter高阶组件处理过后才有这三个路由对象! 个别非路由匹配渲染的组件没有路由对象, 这时用withRouter 处理一下就可以了.

### 路由参数

#### 动态路由传参

动态路由: 路由地址中带有 :xx 这样的路由地址就是动态路由. ( 刷新页面, 参数不消失，参数会在地址栏显示 )

<Route path="/detail/:id" component={Detail}></Route>

组件内, 通过 props.match.params.xxx 接受传递过来的参数

#### 通过query传参

<Link to={{pathname:'/demo',query:{ id:22 }}}>XX</Link>

query传参, 刷新页面参数消失.

组件内, 通过 props.location.query.xxx 接受传递过来的参数

前提：必须由其他页面跳过来，参数才会被传递过来

#### 通过state传参

<Link to={{pathname:'/demo',state:{id:12,name:'dahuang'}}}>XX</Link>

state传参, 刷新页面, 参数不消失，state传的参数是加密的，比query传参好用, 但是通过手动改地址调路由无法传参.

组件内, 通过 props.location.state.xxx 就可以接受到传递过来的参数

### 嵌套路由

嵌套路由就是在子页面中再设置一层新的路由导航规则。

一级路由页面

function AppRouter() {

return (

<Router>

<div className="mainDiv">

<div className="leftNav">

<h3>一级导航</h3>

<ul>

<li> <Link to="/">博客首页</Link> </li>

<li><Link to="/video">视频教程</Link> </li>

</ul>

</div>

<div className="rightMain">

<Route path="/" exact component={Index} />

</div>

</div>

</Router>

);

}

二级路由页面

function Video(){

return (

<div>

<div className="topNav">

<ul>

<li><Link to="/video/reactpage">React教程</Link></li>

<li><Link to="/video/vue">Vue教程</Link></li>

<li><Link to="/video/flutter">Flutter教程</Link></li>

</ul>

</div>

<div className="videoContent">

<Route path="/video/reactpage/" component={Reactpage} />

<Route path="/video/vue/" component={Vue} />

<Route path="/video/flutter/" component={Flutter} />

</div>

</div>

)

}

注意: 二级路由, 三级路由的父路由是不能添加exact属性的

### 重定向与404路由

**路由重定向Redirect**

比如我们进入Index组件，然后Index组件直接重定向到Home组件。

<Redirect to="/home" />

**404路由**

我们在所有路由配置的最后添加一个无条件路由, 这样, 无路由匹配时页面跳转到404页面( NoMatch页面 )

<Switch>

<Route path="/Page" component={Page} />

<Route path="/Sub" component={Childs} />

<Route component={NoMatch} /> // 匹配未找到的路由

</Switch>

### 三种路由渲染方式

每次路由切换都触发路由对应的组件重新创建并渲染，无论是component属性还是render属性.

<Route path="/home" component = { Home }/>

当用component属性，值为匿名函数时，父组件的每次重新render（无论有没有props传递给子组件）都会造成子组件重新render（每次render都会执行constructor）

当用component属性时，值为react元素时，父组件的每次重新render都不会造成子组件每次都执行constructor

<Route path="/home" render = { () => {

return <Home/>

} }/>

当用render属性时，值为匿名函数时，父组件的每次重新render都不会造成子组件每次都执行constructor

当用render属性时，值不可以为react元素，只能是函数

<Route path="/home">

<Home/>

</Route>

最后一种方式每次（第一次除外）App执行render不会造成MyHome重新执行一次初始化生命周期，只会执行update的生命钩子，可以从父组件传递props到子组件，但是这个方式有一个缺点，子组件不能在props访问路由属性

### 路由懒加载

react项目在打包完成后发现chunk.js文件比较大，导致打开首页需要时间比较久，因此，需要进行优化。

其实仔细考虑一下不难发现，由于打包后将所有资源都打包到了一个chunk.js下，导致所有资源都一起加载了，所以进入页面会很慢。那怎么办？解决思路，拆分`chunk.js`成多个进行异步加载, 即路由懒加载。

// 导入路由对应使用的组件

import React,{Component, lazy, Suspense} from "react"

const Foo = lazy(() => import("xxxxxx"));

// 进行异步加载

<Suspense fallback={<div>加载中</div>}>

<Route path="xxxx" component={Foo}></Route>

// ....

</Suspense>

注意: Suspense组件需要包裹在所有路由组件的外部, 只需要写一个就可以了, 缺点是跳多个路由都显示相同的动画.

### 路由V6版

<Routes>

{/\* exact指定路由为精准匹配,指定后,浏览器地址栏中的url必须和这里的path完全一样 \*/}

{/\* 当一个页面有子路由时,父路由不能设置exact \*/}

<Route path="/films" element={<Films />}>

<Route path="/films/hotFilm" extra element={<HotFilm/>}></Route>

<Route path="/films/newFilm" extra element={<NewFilm/>}></Route>

<Route path='/films' exact element={<Navigate to="/films/hotFilm" />}></Route>

</Route>

<Route path="/cinema" exact element={<Cinema />}></Route>

<Route path="/news" element={<News />}></Route>

<Route path="/userCenter" element={<UserCenter />}></Route>

<Route path='/' exact element={<Navigate to="/films" />}></Route>

<Route path='\*' element={<NoMatch />}></Route>

</Routes>

1. 后台动态获取路由进行配置

有时候做一个后台管理系统，菜单并不是写死的，而是通过后台接口获得的，这时候我们要如何根据后台接口编写我们的路由。

后台获取路由配置，并编写动态路由配置.

function AppRouter() {

let routeConfig =[

{path:'/',title:'博客首页',exact:true,component:Index},

{path:'/video/',title:'视频教程',exact:false,component:Video},

]

return (

<Router>

<div className="mainDiv">

<div className="leftNav">

<h3>一级导航</h3>

<ul>

{

routeConfig.map((item,index)=>{

return (<li key={index}> <Link to={item.path}>{item.title}</Link> </li>)

})

}

</ul>

</div>

<div className="rightMain">

{

routeConfig.map((item,index)=>{

return (<Route key={index} exact={item.exact} path={item.path} component={item.component} />)

})

}

</div>

</div>

</Router>

);

}

1. 综合案例

<cdf-会员购> 底部导航(路由实现)

<cdf-会员购> 跳转详情页(动态路由实现)