**2、link**

LINK :是react-router中提供的路由切换组件，基于他可以实现点击时候路由的切换

TO [string] : 跳转到指定的路由地址

TO [object] : 可以提供一些参数配置项（和redirect类似）

{

pathame:跳转地址

serach:问号传参

state:基于这种凡是传递信息

}

replace:false 是替换history stack中当前的地址（true），还是追加一个新的地址（false）

React-router中提供的组件都要在任何一个router（hash-router）包裹的范围内使用

**原理：基于**

**2、NavLink：**

和link类似，都是为了实现路由切换跳转的，不同在于，nav-link组件在当前页面哈希地址和组件对应地址相吻合的时候，会默认给组件加一个active样式，让其有选中状态

和link类似，to和replace等属性都有，用法一样

activeClassName:把默认加的active样式类改为自己设定的，（每一个navlink都要加=””）

activeStyle:给匹配的这个nav-link设置行内样式

<NavLink

to="/faq"

activeStyle={{

fontWeight: 'bold',

color: 'red'

}}

>FAQs</NavLink>

excat&strict 控制匹配的时候是否严格匹配

isActive:匹配后的样式

<NavLink to=’/custom’>最后也会转换为A标签，如果当前页面的哈希地址和此组件中的to地址匹配了，则会给渲染后的A标签设置默认的样式类：active

NavLink不是点击谁，谁有选中的样式（但是可以路由切换），而且当前页面哈希后的地址和nav-link中的to比较，哪个匹配了，哪个才有选中的样式，所以给第一个/ 全部匹配加exact

**3、with-router:**

这个方法的意思是把一个非路由管控的组件，模拟称为路由管控的组件

<route path=’/’ component={nav}> //收路由管控的组件

Width-router(contect()(NAv))先把nav基于connect高阶一下，返回的是一个代理组件proxy，把返回的代理组件收路由管控

受路由管控组件的一些特点：

1. 只有当前页面的哈希地址（/#/…）和路由指定的地址匹配，才会把对应的组件渲染（with-router是没有地址匹配，都被模拟称为受路由管控的）
2. 路由切换的原理：凡是匹配的路由，都会把对应的组件内容重新添加到页面中，相反，不匹配的都会在页面中移除掉，下一次重新匹配上，组件需要重新渲染到页面中；每一次路由切换的时候（页面的哈希路由地址改变），都会从一级路由开始重新校验一遍
3. 所有受路由管控的组件，在组件的属性props上都默认添加了三个属性

**HISTORY**（每一次路由的切换，要不然是替换现有的）

Push ：向池子中追加一条新的信息，达到切换指定的路由地址的目的

**this**.props.history.push('/plan') //js实现路由切换

*/\*\*\*   
 \* 基于<Redirect from={'/custom'} to={'/custom/list'} />  
 \* 如果不加push，是把当前信息替换 '/custom/list'如果加了push，是产生一条新的记录  
 \* \*/*

Go ：跳转到指定的地址（传的是数字 0当前 -1上一个 -2上两个）

Go-back ⬄go(-1)回退到上一个地址

Go-forward⬄go(1)向前走一步

…

**LOCATION**  获取当前哈希路由渲染组件的一些信息

Pathname: 当前哈希路由地址 /custom/list

Search: 当前页面的问号传参值 ?lx=unsafe

**[接收的]（用qs）**

**let** {location:{search},data}=**this**.props;

State: 基于redirect/link/nav-link中的to，传递的是一个对象，对象中编写的state，就可以在location.state中获取到

**MATCH ：**获取的是当前路由匹配的一些结果（路径传参才会用到）

Params：如果当前路由匹配的是地址路径参数，则这里可以获取传递参数的值

在spa路由管控的项目中，从列表跳转到详情，总需要传递一些信息给组件，以此来展示不同的信息，传递给详情页信息的方式有好多种：

【不推荐的】

本地存储

Redux存储

点击列表中某一项的时候，把信息存储到本地或者redux中，跳转到详情页面，把信息从本地或者redux中获取到即可，这样也算实现了信息的共享

【推荐】

1. 问号传参 Search:`?id=${id}`
2. 基于state传值（弊端：一旦页面刷新，上一次传递的state值就没有了，比如“支付页”就是故意这样做）
3. url地址参数(把参数当做地址的一部分)

如：path=`/custom/detail/:id/:name` 路由冒号后面的值就是以后解析时候的属性名

//设置路由

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

//使用路由  
<Link to={{pathname:`/custom/detail/${id}`}}>

编号：{id} &nbsp;&nbsp;姓名：{name}

</Link>

//匹配路由  
let {match: {params}, data} = this.props, //路由冒号后面的的值就是以后解析时候的属性名

customID = parseFloat(params.id) || 0;

**小研究：**

1：Route/withRouter的children是高阶组件会加props：history三件套

2：react-router-dom（v4版本）基于context更新的脆弱性

3：connect等高级组件对于shouldComponentUpdate的处理

如果你对上面三点都知道，那你就不用往下看了。。。（大神请收下我的膝盖）

先回顾一下，

当时NAV组件用了connect导出，出现NavLink无法更新的状况，经过我的测试，解决方法有三，

第一种就是@吴智同学的在NAV前面加<Route>标签，

第二种是老师的<withRouter>，

第三种更简单，业务上NAV根本用不到react-redux库，不用connect就好。这个很简单，但是原理呢？我还是一脸懵逼，其实这个过程简单说来是这样的，如果没有connect（第三种），那么底层的首页等按钮更新了context，顶层HashRouter解析并开始向下解析，解析到每个Route/Switch时会拿来context判断，做出相应的行为（子组件走的更新的一套生命周期，有图更容易理解），一直走到NavLink，更新自己的状态，加上active类等（其实NavLink JSX解析出来是<NavLink><Route><Link><a>，这样加上的，具体不详说了，懂得自然懂），重点来了，那么我加上connect组件咋就不行了，原因是react-redux的connect高级组件可能会有自己的处理，他会在自己生命周期上的shouldComponentUpdate做手脚，判断自己的state/props有没有改变，没改变就返回false，这样处理其实是为了防止无谓更新的，但是把Route坑了，Route是基于context更新的，自己的children里面可怜的Route等更新呢，结果根本就没走它，connect就给拦住了，connect自己的生命周期都停了，走不到update，更别说children的了，那为啥外面加<Route>/里面connect外包上<withRouter>就可以了呢，原因就是Route/withRouter的children如果判断是高阶组件就会加props：history/match/location，这样就给connect组件传了props，那么props一改变，connect里面的shouldComponentUpdate就拦不住了，就可以顺利的更新啦，就是这么简单！