React Router

React Router 6 快速上手

1.概述

1. React Router 以三个不同的包发布到 npm 上，它们分别为：
    1. react-router: 路由的核心库，提供了很多的：组件、钩子。
    2. **react-router-dom:** 包含react-router所有内容，并添加一些专门用于 DOM 的组件，例如 `<BrowserRouter>` 等 。
    3. react-router-native: 包括react-router所有内容，并添加一些专门用于ReactNative的API，例如: `<NativeRouter>` 等。
2. 与React Router 5.x 版本相比，改变了什么？
    1. 内置组件的变化：移除 `<Switch/>` ，新增 `<Routes/>` 等。
    2. 语法的变化： `component={About}` 变为 `element={<About/>}` 等。
    3. 新增多个hook： `useParams` 、 `useNavigate` 、 `useMatch` 等。
    4. 官方明确推荐函数式组件了！！！

    ……

2.Component

1. `<BrowserRouter>`

1. 说明：`<BrowserRouter>`用于包裹整个应用。
2. 示例代码：

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

ReactDOM.render(
  <BrowserRouter>
    {/* 整体结构（通常为App组件） */}
  </BrowserRouter>,root
);
```

2. <HashRouter>

1. 说明：作用与`<BrowserRouter>`一样，但`<HashRouter>`修改的是地址栏的hash值。
2. 备注：6.x版本中`<HashRouter>`、`<BrowserRouter>`的用法与 5.x 相同。

3. <Routes/> 与 <Route/>

1. v6版本中移出了先前的`<Switch>`，引入了新的替代者：`<Routes>`。
2. `<Routes>`和`<Route>`要配合使用，且必须要用`<Routes>`包裹`<Route>`。
3. `<Route>`相当于一个 if 语句，如果其路径与当前 URL 匹配，则呈现其对应的组件。
4. `<Route caseSensitive>`属性用于指定：匹配时是否区分大小写（默认为false）。
5. 当URL发生变化时，`<Routes>`都会查看其所有子`<Route>`元素以找到最佳匹配并呈现组件 。
6. `<Route>`也可以嵌套使用，且可配合`useRoutes()`配置“路由表”，但需要通过`<Outlet>`组件来渲染其子路由。
7. 示例代码：

```
<Routes>
    /*path属性用于定义路径，element属性用于定义当前路径所对应的组件*/
    <Route path="/login" element={<Login />}></Route>

        /*用于定义嵌套路由，home是一级路由，对应的路径/home*/
```

```
    <Route path="home" element={<Home />}>
       /*test1 和 test2 是二级路由,对应的路径是/home/test1 或 /home/test2*/
      <Route path="test1" element={<Test/>}></Route>
      <Route path="test2" element={<Test2/>}></Route>
        </Route>

        //Route也可以不写element属性, 这时就是用于展示嵌套的路由 .所对应的路径是/users/xxx
    <Route path="users">
       <Route path="xxx" element={<Demo />} />
    </Route>
</Routes>
```

4. <Link>

1. 作用: 修改URL，且不发送网络请求（路由链接）。
2. 注意: 外侧需要用 `<BrowserRouter>` 或 `<HashRouter>` 包裹。
3. 示例代码：

```
import { Link } from "react-router-dom";

function Test() {
  return (
    <div>
        <Link to="/路径">按钮</Link>
    </div>
  );
}
```

5. <NavLink>

1. 作用: 与 `<Link>` 组件类似，且可实现导航的“高亮”效果。
2. 示例代码：

```
// 注意: NavLink默认类名是active, 下面是指定自定义的class

//自定义样式
<NavLink
    to="login"
    className={({ isActive }) => {
        console.log('home', isActive)
```

```
        return isActive ? 'base one' : 'base'
    }}
>login</NavLink>

/*
    默认情况下，当Home的子组件匹配成功，Home的导航也会高亮，
    当NavLink上添加了end属性后，若Home的子组件匹配成功，则Home的导航没有高亮效果。
*/
<NavLink to="home" end >home</NavLink>
```

6. <Navigate>

1. 作用：只要`<Navigate>`组件被渲染，就会修改路径，切换视图。
2. `replace`属性用于控制跳转模式（push 或 replace，默认是push）。
3. 示例代码：

```
import React,{useState} from 'react'
import {Navigate} from 'react-router-dom'

export default function Home() {
    const [sum,setSum] = useState(1)
    return (
        <div>
            <h3>我是Home的内容</h3>
            {/* 根据sum的值决定是否切换视图 */}
            {sum === 1 ? <h4>sum的值为{sum}</h4> : <Navigate to="/about" replace={true}/>}
            <button onClick={()=>setSum(2)}>点我将sum变为2</button>
        </div>
    )
}
```

7. <Outlet>

1. 当`<Route>`产生嵌套时，渲染其对应的后续子路由。
2. 示例代码：

```
//根据路由表生成对应的路由规则
const element = useRoutes([
  {
```

```
    path:'/about',
    element:<About/>
  },
  {
    path:'/home',
    element:<Home/>,
    children:[
      {
        path:'news',
        element:<News/>
      },
      {
        path:'message',
        element:<Message/>,
      }
    ]
  }
])

//Home.js
import React from 'react'
import {NavLink,Outlet} from 'react-router-dom'

export default function Home() {
    return (
        <div>
            <h2>Home组件内容</h2>
            <div>
                <ul className="nav nav-tabs">
                    <li>
                        <NavLink className="list-group-item" to="news">News</NavLink>
                    </li>
                    <li>
                        <NavLink className="list-group-item" to="message">Message</NavLink>
                    </li>
                </ul>
                {/* 指定路由组件呈现的位置 */}
                <Outlet />
            </div>
        </div>
    )
}
```

3.Hooks

1. useRoutes()

1. 作用：根据路由表，动态创建`<Routes>`和`<Route>`。
2. 示例代码：

```
//路由表配置: src/routes/index.js
import About from '../pages/About'
import Home from '../pages/Home'
import {Navigate} from 'react-router-dom'

export default [
    {
        path:'/about',
        element:<About/>
    },
    {
        path:'/home',
        element:<Home/>
    },
    {
        path:'/',
        element:<Navigate to="/about"/>
    }
]

//App.jsx
import React from 'react'
import {NavLink,useRoutes} from 'react-router-dom'
import routes from './routes'

export default function App() {
    //根据路由表生成对应的路由规则
    const element = useRoutes(routes)
    return (
        <div>
            ......
      {/* 注册路由 */}
      {element}
          ......
        </div>
    )
```

```
}
```

2. useNavigate()

1. 作用：返回一个函数用来实现编程式导航。
2. 示例代码：

```
import React from 'react'
import {useNavigate} from 'react-router-dom'

export default function Demo() {
  const navigate = useNavigate()
  const handle = () => {
    //第一种使用方式：指定具体的路径
    navigate('/login', {
      replace: false,
      state: {a:1, b:2}
    })
    //第二种使用方式：传入数值进行前进或后退，类似于5.x中的history.go()方法
    navigate(-1)
  }

  return (
    <div>
      <button onClick={handle}>按钮</button>
    </div>
  )
}
```

3. useParams()

1. 作用：回当前匹配路由的`params`参数，类似于5.x中的`match.params`。
2. 示例代码：

```
import React from 'react';
import { Routes, Route, useParams } from 'react-router-dom';
import User from './pages/User.jsx'

function ProfilePage() {
  // 获取URL中携带过来的params参数
  let { id } = useParams();
}
```

```
9
10  function App() {
11    return (
12      <Routes>
13        <Route path="users/:id" element={<User />}/>
14      </Routes>
15    );
16  }
```

4. useSearchParams()

1. 作用：用于读取和修改当前位置的 URL 中的查询字符串。
2. 返回一个包含两个值的数组，内容分别为：当前的seaech参数、更新search的函数。
3. 示例代码：

```
1  import React from 'react'
2  import {useSearchParams} from 'react-router-dom'
3
4  export default function Detail() {
5      const [search,setSearch] = useSearchParams()
6      const id = search.get('id')
7      const title = search.get('title')
8      const content = search.get('content')
9      return (
10         <ul>
11             <li>
12                 <button onClick={()⇒setSearch('id=008&title=哈哈&content=嘻嘻')}>点我更新一下收到的search参数</button>
13             </li>
14             <li>消息编号：{id}</li>
15             <li>消息标题：{title}</li>
16             <li>消息内容：{content}</li>
17         </ul>
18     )
19 }
```

5. useLocation()

1. 作用：获取当前 location 信息，对标5.x中的路由组件的`location`属性。
2. 示例代码：

```
```

```jsx
import React from 'react'
import {useLocation} from 'react-router-dom'

export default function Detail() {
    const x = useLocation()
    console.log('@',x)
  // x就是location对象:
    /*
        {
      hash: "",
      key: "ah9nv6sz",
      pathname: "/login",
      search: "?name=zs&age=18",
      state: {a: 1, b: 2}
    }
    */
    return (
        <ul>
            <li>消息编号：{id}</li>
            <li>消息标题：{title}</li>
            <li>消息内容：{content}</li>
        </ul>
    )
}
```

6. useMatch()

1. 作用：返回当前匹配信息，对标5.x中的路由组件的`match`属性。
2. 示例代码：

```jsx
<Route path="/login/:page/:pageSize" element={<Login />}/>
<NavLink to="/login/1/10">登录</NavLink>

export default function Login() {
  const match = useMatch('/login/:x/:y')
  console.log(match) //输出match对象
  //match对象内容如下:
  /*
    {
      params: {x: '1', y: '10'}
      pathname: "/LoGin/1/10"
      pathnameBase: "/LoGin/1/10"
      pattern: {
        path: '/login/:x/:y',
```

```
        caseSensitive: false,
        end: false
      }
    }
  */
  return (
    <div>
      <h1>Login</h1>
    </div>
  )
}
```

7. useInRouterContext()

作用：如果组件在 `<Router>` 的上下文中呈现，则 `useInRouterContext` 钩子返回 true，否则返回 false。

8. useNavigationType()

1. 作用：返回当前的导航类型（用户是如何来到当前页面的）。
2. 返回值：`POP` 、`PUSH` 、`REPLACE` 。
3. 备注：`POP` 是指在浏览器中直接打开了这个路由组件（刷新页面）。

9. useOutlet()

1. 作用：用来呈现当前组件中渲染的嵌套路由。
2. 示例代码：

```
const result = useOutlet()
console.log(result)
// 如果嵌套路由没有挂载,则result为null
// 如果嵌套路由已经挂载,则展示嵌套的路由对象
```

10.useResolvedPath()

1. 作用：给定一个 URL值，解析其中的：path、search、hash值。