React Router (V6)

作者：kerwin

版本：QF1.0

版权：千锋HTML5大前端教研院

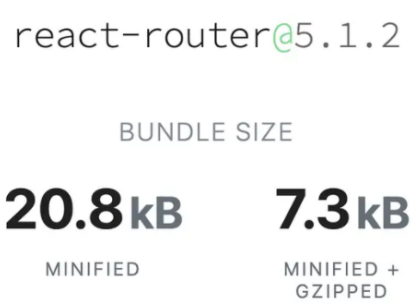
公众号: 大前端私房菜

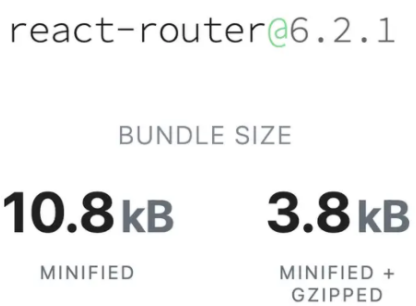
一. 简介

- react-router： **核心模块**，包含 React 路由大部分的核心功能，包括路由匹配算法和大部分核心组件和钩子。
- react-router-dom：React应用中用于路由的软件包，包括react-router的所有内容，并添加了一些特定于 DOM 的 API，包括但不限于BrowserRouter、HashRouter和Link。
- react-router-native： 用于开发React Native应用，包括react-router的所有内容，并添加了一些特定于 React Native 的 API，包括但不限于NativeRouter和Link。

二. 对比V5

1. 包大小对比





2. 特性变更

path：与当前页面对应的URL匹配。

element：新增，用于决定路由匹配时，渲染哪个组件。代替，v5的component和render。

3. 代替了
4. <*Outlet*></*Outlet*>让嵌套路由更简单
5. useNavigate代替useHistory
6. 移除了的activeClassName 和 activeStyle
7. 钩子useRoutes代替react-router-config
8. https://reactrouter.com/docs/en/v6/upgrading/v5

三. 用法详解

1. 一级路由与多级路由

```
<Routes>
    {/*<Route path="/" element={<Film/>}/>*/}
    <Route index element={<Film/>}/>
    <Route path="/film" element={<Film/>}/>
    <Route path="/cinema" element={<Cinema/>}/>
    <Route path="/center" element={<Center/>}/>
</Routes>
```

> index用于嵌套路由，仅匹配父路径时，设置渲染的组件。
>
> 解决当嵌套路由有多个子路由但本身无法确认默认渲染哪个子路由的时候，可以增加index属性来指定默认路由。index路由和其他路由不同的地方是它没有path属性，他和父路由共享同一个路径。

2. 路由重定向

(1) 官方推荐方案 1: 使用 Navigate 组件替代

```
<Routes>
    {/* <Route index element={<Film/>}/> */}
    <Route path="/film" element={<Film/>}/>
    <Route path="/cinema" element={<Cinema/>}/>
    <Route path="/center" element={<Center/>}/>
    <Route path="*" element={<Navigate to="/film"/>}/>
</Routes>
```

(2) 官方推荐方案 2: 自定义 Redirect 组件

```
<Route path="*" element={<Redirect to="/film"/>}/>

function Redirect({to}){
    const navigate  =useNavigate()
    useEffect(() => {
        navigate(to,{replace:true})
    })
    return null
}
```

(3) 404如何实现?

```
<Route path="/" element={<Redirect to="/film"/>}/>
<Route path="*" element={<NotFound/>}/>
```

3. 嵌套路由

```
<Route path="/film" element={<Film/>}>
    {/* <Route index  element={<Nowplaying/>}/> */}
    <Route path="" element={<Redirect to="/film/nowplaying"/>}/>
    <Route path="nowplaying" element={<Nowplaying/>}/>
    <Route path="comingsoon" element={<Comingsoon/>}/>
</Route>

Film组件 <Outlet></Outlet>
```

4. 声明式导航与编程式导航

```
 <ul>
    <li><NavLink to="nowplaying" className={({ isActive }) => isActive ?
"kerwinactive" : ""} >正在热映</NavLink></li>
    <li><NavLink to="comingsoon" className={({ isActive }) => isActive ?
"kerwinactive" : ""}>即将上映</NavLink></li>
 </ul>
```

```
//url传参
const navigate = useNavigate()
navigate(`/detail?id=${id}`)

//获取url参数
import { useSearchParams } from 'react-router-dom'
const [searchParams, setSearchParams] = useSearchParams()
// 获取参数
searchParams.get('id')
// 判断参数是否存在
searchParams.has('id')
// 同时页面内也可以用set方法来改变路由
setSearchParams({"id":2})
```

5.动态路由

```
//跳转页面,路由传参
navigate(`/detail/${id}`)

//配置动态路由
<Route path="/detail/:id" element={<Detail/>}/>

//获取动态路由参数

const {id} = useParams()
```

6.路由拦截

```
<Route path="/center" element={
    <AuthComponent>
        <Center></Center>
    </AuthComponent>
}/>

function AuthComponent({children}){
    return localStorage.getItem("token")?
    children:<Redirect to="/login"/>
}
```

7.路由模式

```
import {HashRouter} from 'react-router-dom'
import {BrowserRouter} from 'react-router-dom'
```

8.withRouter / 类组件跳转方法

```
/*
 * @作者: kerwin
 * @公众号: 大前端私房菜
 */
import {
    useLocation,
    useNavigate,
    useParams
  } from "react-router-dom";

  function withRouter(Component) {
    function ComponentWithRouterProp(props) {
      let location = useLocation();
      let push = useNavigate();
```

```
      let params = useParams();
      return (
        <Component
          {...props}
          history={{ location, push, params }}
        />
      );
    }

    return ComponentWithRouterProp;
  }

  export default withRouter
```

9. 路由懒加载

```
const LazyLoad = (path) => {
    const Comp = React.lazy(() => import(`../views/${path}`))
    return (
        <React.Suspense fallback={<>加载中...</>}>
            <Comp />
        </React.Suspense>
    )
}
```

```
export default function MRouter() {
    return (
        <Routes>
            {/* <Route index element={<Film/>}/> */}
            <Route path="/film" element={LazyLoad("Film")}>
                {/* <Route index  element={<Nowplaying/>}/> */}
                <Route path="" element={<Redirect to="/film/nowplaying"/>}/>
                <Route path="nowplaying" element={LazyLoad("film/Nowplaying")}/>
                <Route path="comingsoon" element={LazyLoad("film/Comingsoon")}/>
            </Route>
            <Route path="/cinema" element={LazyLoad("Cinema")}/>
            <Route path="/login" element={LazyLoad("Login")}/>
            <Route path="/center" element={<AuthComponent>
                {LazyLoad("Center")}
            </AuthComponent>}/>
            <Route path="/detail/:id" element={LazyLoad("Detail")}/>
            <Route path="/" element={<Redirect to="/film"/>}/>
            <Route path="*" element={LazyLoad("NotFound")}/>
        </Routes>
    )
}
```

10. useRoutes钩子配置路由

```
export default function MRouter() {
    const element = useRoutes([
        {path:"/film",element:LazyLoad("Film"),children:[
            {
                path:"",
                element:<Redirect to="/film/nowplaying"/>
            },
            {
                path:"nowplaying",
                element:LazyLoad("film/Nowplaying")
            },
            {
                path:"comingsoon",
                element:LazyLoad("film/Comingsoon")
            }
        ]},
        {
            path:"/cinema",element:LazyLoad("Cinema")
        },
        {
            path:"/login",element:LazyLoad("Login")
        },
        {
            path:"/center",element:<AuthComponent>
            {LazyLoad("Center")}
            </AuthComponent>
        },
        {
            path:"/detail/:id",element:LazyLoad("Detail")
        },
        {
            path:"/",element:<Redirect to="/film"/>
        },
        {
            path:"*",element:LazyLoad("NotFound")
        },
    ])

    return element
}
```