目录

[4.x 2](#_Toc495830892)

[模板 3](#_Toc495830893)

[普通版 3](#_Toc495830894)

[配置文件版 3](#_Toc495830895)

[匹配原理： : () \* \*\* 5](#_Toc495830896)

[api 5](#_Toc495830897)

[Router 5](#_Toc495830898)

[children(必须) 5](#_Toc495830899)

[routes 6](#_Toc495830900)

[history 6](#_Toc495830901)

[createElement(Component,props) 6](#_Toc495830902)

[stringifyQuery(queryObject) 6](#_Toc495830903)

[parseQueryString(queryString) 6](#_Toc495830904)

[onError(error) 6](#_Toc495830905)

[onUpdate() 7](#_Toc495830906)

[Link 7](#_Toc495830907)

[to 7](#_Toc495830908)

[query 7](#_Toc495830909)

[hash 7](#_Toc495830910)

[state 7](#_Toc495830911)

[activeClassName 7](#_Toc495830912)

[activeStyle 7](#_Toc495830913)

[onClick(e) 8](#_Toc495830914)

[在link中加入query 8](#_Toc495830915)

[IndexLink 8](#_Toc495830916)

[RoutingContext 8](#_Toc495830917)

[Route 8](#_Toc495830918)

[path 8](#_Toc495830919)

[component 9](#_Toc495830920)

[components 9](#_Toc495830921)

[getComponent(loction,callback) 9](#_Toc495830922)

[getComponents(location,callback) 9](#_Toc495830923)

[children 9](#_Toc495830924)

[onEnter(nextState, replaceState, callback?) 10](#_Toc495830925)

[onLeave() 10](#_Toc495830926)

[PlainRoute 10](#_Toc495830927)

[childRoutes 10](#_Toc495830928)

[getChildRoutes(location,callback) 10](#_Toc495830929)

[indexRoute 10](#_Toc495830930)

[getIndexRoute(location,callback) 10](#_Toc495830931)

[Redirect 10](#_Toc495830932)

[from 10](#_Toc495830933)

[to 10](#_Toc495830934)

[query 10](#_Toc495830935)

[IndexRoute 11](#_Toc495830936)

[IndexRedirect 11](#_Toc495830937)

[Route Components 11](#_Toc495830938)

[history 11](#_Toc495830939)

[location 11](#_Toc495830940)

[params 11](#_Toc495830941)

[route 11](#_Toc495830942)

[routeParams 11](#_Toc495830943)

[children 12](#_Toc495830944)

[生命周期mixin 12](#_Toc495830945)

[routerWillLeave(nextLocation) 12](#_Toc495830946)

[History Mixin 12](#_Toc495830947)

[putState(state,pathname,query 13](#_Toc495830948)

[replaceState(state,pathname,query)) 13](#_Toc495830949)

[go(n) 13](#_Toc495830950)

[goBack() 13](#_Toc495830951)

[goForward() 13](#_Toc495830952)

[createPath(pathname,query) 13](#_Toc495830953)

[createHref(pathname,query) 13](#_Toc495830954)

[isActive(pathname,query,indexOnly) 13](#_Toc495830955)

[RouteContext Mixin 13](#_Toc495830956)

[Utilities 13](#_Toc495830957)

[useRoutes(createHistory 13](#_Toc495830958)

[match(location,cb)) 13](#_Toc495830959)

[createRoutes(routes) 13](#_Toc495830960)

# 4.x

You should not use <Route component> and <Route children> in the same route

route不能在嵌套

react-router4没有indexRoute了(v4本质上变成了一个组件)

# 模板

## 普通版

容器

App组件 render({this.props.children})

路由

var MyRouter=<Router>

                <Route path='/' component={app}>

                <IndexRoute component={index}/>

                <Route path='one' component={one}/>

                <Route path='detail' component={detail}>

                    <Route path='/message/:id' component={Message}/>

                    <Redirect from='message/:id' to='/message/:id'/>

                </Route>

                </Route>

            </Router>

render(MyRouter,document.body)

## 配置文件版

routeConfig={

    path:'/',

    component:app,

    indexRoute:{path:'/',component:index},

    childRoutes:[

        {path:'one',component:one},

        {path:'detail',

            component:detail,

            childRoutes:[

                { path: '/messages/:id', component: Message },

                {   path: 'messages/:id',

                    onEnter: function (nextState, replaceState) {

                    replaceState(null, '/messages/' + nextState.params.id)

                }

          }

            ]}

    ]

}

render(<Router routes={routeConfig} />, document.body)

或者每个route再单独拿出来

const one = {

    path: 'one',

    getComponent(nextState, cb) {

        require.ensure([], (require) => {

            return cb(null, require('./js/components/about'))

        })

    }

}

# 匹配原理： : () \* \*\*



# api

## Router

### children(必须)

一个或多个的 Route 或 PlainRoute。当 history 改变时， <Router> 会匹配出 Route 的一个分支，并且渲染这个分支中配置的组件，渲染时保持父 route 组件嵌套子 route 组件。

### routes

children 的别名。

### history

### createElement(Component,props)

当 route 准备渲染 route 组件的一个分支时，就会用这个函数来创建 element。当你使用某种形式的数据进行抽象时，你可以想要获取创建 element 的控制权，例如在这里设置组件监听 store 的变化，或者使用 props 为每个组件传入一些应用模块。

<Router createElement={createElement} />

// 默认行为

function createElement(Component, props) {

// 确保传入了所有的 props！

return <Component {...props}/>

}

// 你可能会使用什么，如 Relay

function createElement(Component, props) {

// 确保传入了所有的 props！

return <RelayContainer Component={Component} routerProps={props}/>

}

### stringifyQuery(queryObject)

把 Link 或调用 transitionTo 函数的对象转化成 URL query 字符串

### parseQueryString(queryString)

把 query 字符串转化成对象，并传递给 route 组件 props 的函数。

### onError(error)

通常，它们会来自那些异步的特性，如 route.getComponents，route.getIndexRoute，和 route.getChildRoutes。

### onUpdate()

当 URL 改变时，需要更新路由的 state 时会被调用。

## Link

<Link> 以适当的 href 去渲染一个可访问的锚标签。

### to

跳转链接的路径，如 /users/123。(绝对路径)

### query

已经转化成字符串的键值对的对象。

### hash

URL 的 hash 值，如 #a-hash。

注意：React Router 目前还不能管理滚动条的位置，并且不会自动滚动到 hash 对应的元素上。如果需要管理滚动条位置，可以使用 scroll-behavior 这个库。

### state

保存在 location 中的 state。

### activeClassName

### activeStyle

### onClick(e)

自定义点击事件的处理方法。如处理 <a> 标签一样 - 调用 e.preventDefault() 来防止过度的点击，同时 e.stopPropagation() 可以阻止冒泡的事件。

### 在link中加入query

const Page = (props) =>

<div>

<h1>{props.location.query.message || 'Hello'}<Links /></h1>

</div>

const Links = () =>

<nav>

<Link to={{ pathname: "/", query: {message: "lining"} }}>lining</Link>

</nav>

## IndexLink

## RoutingContext

在 context 中给定路由的 state、设置 history 对象和当前的 location，<RoutingContext> 就会去渲染组件树。

## Route

### path

URL 中的路径。

它会组合父 route 的路径，除非它是从 / 开始的， 将它变成一个绝对路径。

在动态路由中，绝对路径可能不适用于 route 配置中。

如果它是 undefined，路由会去匹配子 route。

### component

### components

### getComponent(loction,callback)

### getComponents(location,callback)

### children

Route 可以被嵌套，this.props.children 包含了从子 route 组件创建的元素。

### onEnter(nextState, replaceState, callback?)

### onLeave()

## PlainRoute

### childRoutes

### getChildRoutes(location,callback)

## indexRoute

### getIndexRoute(location,callback)

## Redirect

在 route 层 <Redirect> 可以被放在任何地方，尽管正常的优先 规则仍适用。如果你喜欢将下一个重定向到它各自的 route 上，from 的路径会匹配到一个与 path 一样的正常路径。

### from

你想由哪个路径进行重定向，包括动态段。

### to

你想重定向的路径。

### query

默认情况下，query 的参数只会经过，但如果你需要你可以指定它们。

## IndexRoute

和route一样

## IndexRedirect

和redirect一样,除了from

## Route Components

路由包裹后的组件。路由会在渲染时将以下属性注入组件中

### history

Router 的 history history。

对于跳转很有用的 this.props.history.pushState(state, path, query)

### location

当前的 location

可以得到query {props.location.query.message || 'Hello'}

### params

URL 的动态段。

### route

渲染组件的 route。

### routeParams

this.props.params 是直接在组件中指定 route 的一个子集。例如，如果 route 的路径是 users/:userId 而 URL 是 /users/123/portfolios/345，那么 this.props.routeParams 会是 {userId: '123'}，并且 this.props.params 会是 {userId: '123', portfolioId: 345}。

### children

匹配到子 route 的元素将被渲染。如果 route 有已命名的组件，那么此属性会是 undefined，并且可用的组件会被直接替换到 this.props 上。

## 生命周期mixin

### routerWillLeave(nextLocation)

nextLocation - 下一个 location

## History Mixin

你的 route components 不需要这个 mixin，它在 this.props.history 中已经是可用的了。这是为了组件更深次的渲染树中需要访问路由的 history 对象。(就是路由组件的字组件)

### putState(state,pathname,query

### replaceState(state,pathname,query))

### go(n)

### goBack()

### goForward()

### createPath(pathname,query)

### createHref(pathname,query)

### isActive(pathname,query,indexOnly)

### RouteContext Mixin

### Utilities

### useRoutes(createHistory

### match(location,cb))

### createRoutes(routes)