

路由

React路由

现代前端应用大多数时SPA（单页应用程序），也就是只有一个HTML页面的应用程序，因为他的用户体验更好、对服务器的压力更小。为了有效地使用单个页面来管理原来多个页面的功能，前端路由应运而生。

- 前端路由功能：让用户从一个视图（页面）导航到另一个视图（页面）
- 前端路由是一套映射规则，在React中，是URL路径与组件的对应关系
- 使用React路由简单来说，就是配置路径和组件（配对）

React路由基本使用

1. 安装： `yarn add react-router-dom`

2. 导入路由的三个核心组件：

BrowserRouter/Route/Link

- `import {BrowserRouter as Router, Route, Link} from 'react-router-dom'`

3. 使用Router组件包裹整个应用

4. 使用Link组件作为导航菜单（路由入口）

- `<Link to="/first">页面一</Link>`

5. 使用Route组件配置路由规则和要展示的组件（路由出口）

- path表示路径，与Link中的to属性的内容对应
- component表示要展示的组件
- `<Route path="/first" component={First}>`
`</Route>`
- **注意react-router-domV6版本之后使用方法有所改动**

常用组件声明

- Router组件：包裹整个应用，以恶搞React应用只需要使用一次
- 两种常用的Router：
 - HashRouter（使用URL的哈希值实现
(localhost:3000/#/first)）在Vue中兼容性更好
 - BrowserRouter（使用H5中的history API实现
(localhost:3000/first)）
- Link组件：用于指定导航链接
 - 最终会被编译为a标签；to属性被编译为href，即浏览器地址栏中的pathname
 - 可以通过location.pathname来获取to中的值
- Route组件：指定路由展示组件相关信息

- path属性：路由规则
- component属性：展示的组件
- Route组件写在哪，组件就会被渲染在哪

路由执行过程

1. 点击Link组件，修改了浏览器地址中的url
2. React路由监听到地址栏url变化
3. React路由内部遍历所有Route组件，使用路由规则（path）与pathname进行匹配
4. 当路由规则与pathname匹配时，展示该Route组件的内容

编程式导航

- 编程式导航：通过JS代码实现页面跳转

```
1 | this.props.history.push('/home')
```

- history是React路由提供的，用于获取浏览器历史记录的相关信息
- push (path)：跳转到某个页面，参数path表示要跳转的路径
- **注意react-route-domV6版本不支持此方法，应使用useNavigate()API**
 - `const navigate = useNavigate();navigate('/home')`

- go(n): 前进或后退到某个页面, 参数n表示前进或后退页面的数量 (-1表示后退一页)

默认路由

- 进入页面时默认的展示页面
- 默认路由: 进入页面时就会默认匹配的路由
- 默认路由的path: /
 - `<Route path="/" component={Home} />`

匹配模式

模糊匹配模式

- 问题: 默认路由在路由切换时仍然会被显示(**V6没有这个问题**)
- 原因: 默认情况下React路由是模糊匹配模式
- 模糊匹配规则: 之哟啊pathname以path开头就会被匹配成功

精确匹配

- 给Route组件添加exact属性, 就能让其变为精确匹配模式
- 精确匹配: 只有当path和pathname 完全匹配时才会展示该路由

组件的生命周期

学习组件的生命周期有助于理解组件的运行方式、从而完成更复杂的组件功能、分析组件错误原因等等

组件的生命周期指：组件从被创建到挂载在页面中运行，再到组件不用时卸载的过程。

钩子函数：生命周期的每个阶段总伴随着一些方法调用，这些方法就是生命周期的钩子函数，为开发人员在不同阶段操作组件提供了时机。

只有类组件才有生命周期

生命周期的三个阶段

1. 创建时
2. 更新时
3. 卸载时

创建时（挂在阶段）

- 执行时机：组件创建时（页面加载时）
- 钩子函数执行顺序：
 1. constructor()
 2. render()
 3. componentDidMount()

钩子函数	触发时机	作用
constructor	创建组件时，最先执行	1. 初始化state 2. 为事件处理程序绑定this
render	每次组件渲染都会触发	渲染UI（注意：不能调用setState()）
componentDidMount	组件挂载（完成DOM渲染）后	1. 发送网络请求 2. DOM操作

不能在render中调用setState的原因是：调用setState会导致数据更新以及UI更新（渲染），即setState方法将会调用render方法，因此如果在render中调用setState会导致递归效用

componentDidMount会紧跟render方法触发，由于DOM操作需要DOM结构已经渲染，因此DOM操作应被放置于该钩子函数内。

更新阶段

- 更新阶段的执行时机包括：
 - New props，组件接收到新属性
 - setState()，调用该方法时

3. forceUpdate(), 调用该方法时

其中forceUpdate用于使组件强制更新，即使没有数值上的改变。

- 钩子函数执行顺序：
 1. shouldComponentUpdate
 2. render()
 3. componentDidUpdate()

钩子函数	触发时机	作用
shouldComponentUpdate	更新阶段的钩子函数，组件重新渲染前执行（即在render前执行）	通过该函数的返回值来决定组件是否重新渲染。
render	每次组件渲染都会触发	渲染UI（与挂载阶段是同一个）
componentDidUpdate	组件更新（完成DOM渲染）后	1. 发送网络请求 2. DOM操作

需要注意的是在componentDidUpdate中调用setState()必须放在一个if条件中，原因与在render中调用setState相同，render执行完后会立即执行componentDidUpdate导致递归调用。通常会比较更新前后的props是否相同，来决定是否重新渲染组件。可以使用componentDidUpdate(prevProps)得到上一次的props，通过this.props获取当前props


```
1 class App extends React.Component {
2
3     constructor(props) {
4         super(props)
5
6         this.state = {
7             count: 0
8         }
9         console.warn('生命周期钩子函数:
10 constructor')
11     }
12     componentDidMount(){
13         console.warn('生命周期钩子函数:
14 componentDidMount')
15     }
16     handleClick = () =>{
17         this.setState({
18             count: this.state.count + 1
19         })
20     }
21
22     render() {
23         return (
24             <div>
25                 <Counter count=
26 {this.state.count} />
27                 <button onClick=
28 {this.handleClick}>打豆豆</button>
```

```
27         </div>
28     )
29 }
30 }
31
32 class Counter extends React.Component {
33     render() {
34         console.warn('--子组件--生命周期钩
子函数: render')
35         return <h1 id='title'>统计豆豆被打
的次数: {this.props.count}</h1>
36     }
37
38
39
40     componentDidUpdate(prevProps) {
41         console.warn('--子组件--生命周期钩
子函数: componentDidUpdate')
42
43         console.log('上一次的props: ',
prevProps, '，当前的props: ', this.props)
44         if(prevProps.count !==
this.props.count) {
45             this.setState({})
46             // 发送ajax请求的代码
47         }
48     }
49 }
50
```

```
51 ReactDOM.render(<App />,
    document.getElementById('root'))
```

卸载时（卸载阶段）

- 执行时机：组件从页面中消失
- 钩子函数执行顺序：
 - `componentWillUnmount`

钩子函数	触发时机	作用
<code>componentWillUnmount</code>	组件卸载（从页面中消失）	执行清理工作（比如：清理定时器等）

```
1 class App extends React.Component {
2
3     constructor(props) {
4         super(props)
5
6         this.state = {
7             count: 0
8         }
9         console.warn('生命周期钩子函数：
10         constructor')
11     }
```

```
12     componentDidMount(){
13         console.warn('生命周期钩子函数：
componentDidMount')
14     }
15
16     handleClick = () =>{
17         this.setState({
18             count: this.state.count + 1
19         })
20     }
21
22     render() {
23         return (
24             <div>
25                 {this.state.count > 3 ?
(
26                     <p>豆豆被打死了~</p>
27                 ) : (
28                     <Counter count=
{this.state.count} />
29                 )}
30                 <button onClick=
{this.handleClick}>打豆豆</button>
31             </div>
32         )
33     }
34 }
35
36 class Counter extends React.Component {
37
```

```
38     componentDidMount() {
39         // 开启定时器
40         this.timerId = setInterval(() =>
41         {
42             console.log("定时器正在执行~")
43             }. 500)
44     }
45     render() {
46         console.warn('--子组件--生命周期钩
子函数: render')
47         return <h1 id='title'>统计豆豆被打
的次数: {this.props.count}</h1>
48     }
49
50     componentWillUnmount(){
51         console.warn('--子组件--生命周期钩
子函数: componentWillUnmount')
52         // 清理定时器
53         clearInterval(this.timerId)
54     }
55
56     componentDidUpdate(prevProps) {
57         console.warn('--子组件--生命周期钩
子函数: componentDidUpdate')
58
59         console.log('上一次的props: ',
prevProps, ', 当前的props: ', this.props)
60         if(prevProps.count !==
this.props.count) {
```

```
61         this.setState({})
62         // 发送ajax请求的代码
63     }
64 }
65 }
66
67 ReactDOM.render(<App />,
    document.getElementById('root'))
```

其他钩子函数

- 旧版本遗留，先已弃用的钩子函数：
 - `componentWillMount()`
 - `ComponentWillReceiveProps()`
 - `ComponentWillUpdate()`
- 新版完整生命周期钩子函数：
 - 创建时：
 - `constructor`
 - `getDerivedStateFromProps`(不常用)
 - `render`
 - `React更新DOM和refs`
 - `componentDidMount`
 - 更新时
 - `getDerivedStateFromProps`(不常用)
 - `shouldComponentUpdate`(详见组件性能优化)
 - `render`
 - `getSnapshotBeforeUpdate`(不常用)

- React更新DOM和refs
- componentDidUpdate
- 卸载时
 - componentWillUnmount