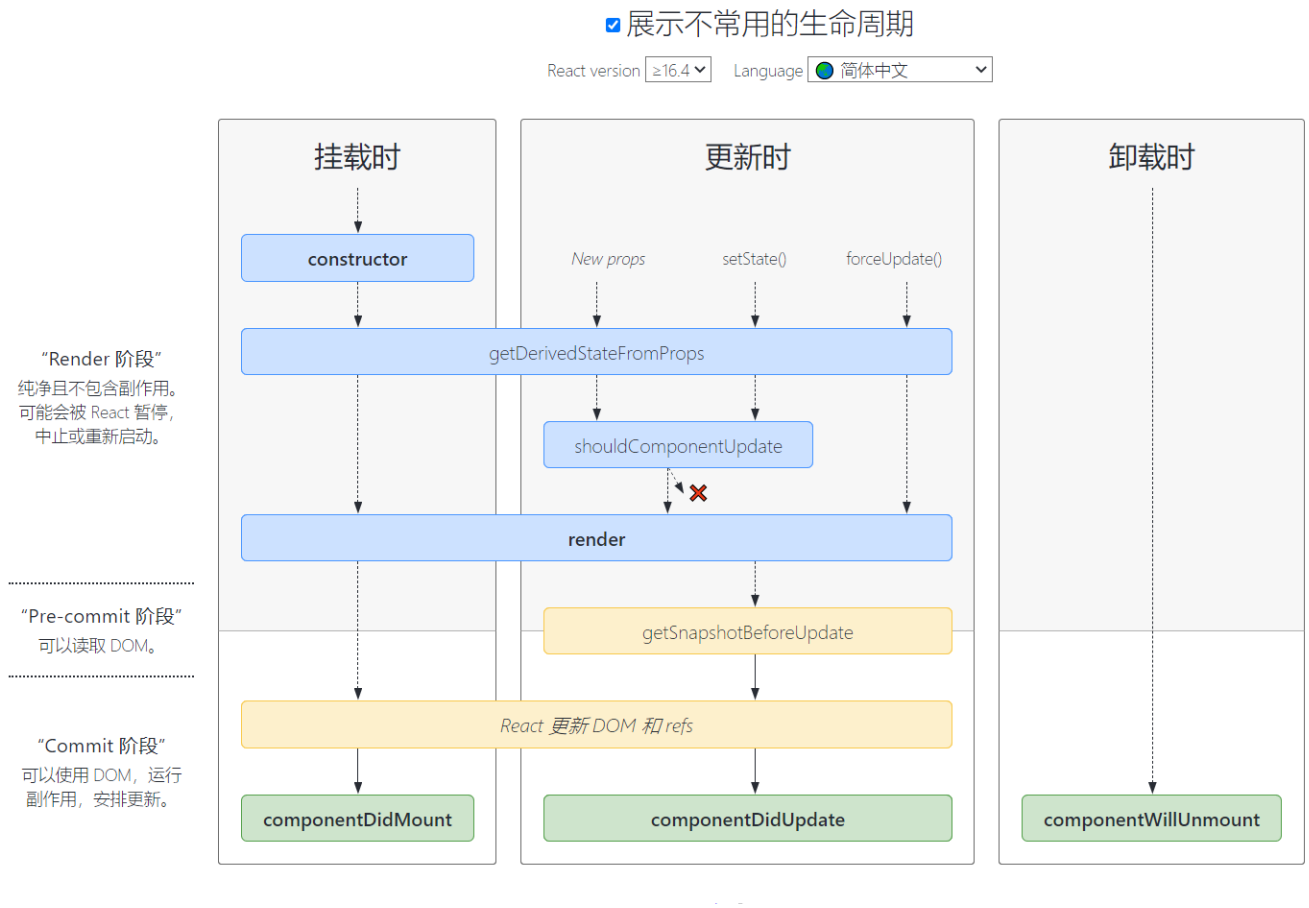
React进阶

1. 生命周期



1. 挂载时

> 由ReactDOM.render()触发

1. constructor(): 构造器函数，加载的时候调用一次，可以初始化state
2. static getDerivedStateFromProps(props, state): 组件每次被render的时候，包括在组件构建之后(虚拟dom之后，实际dom挂载之前)，每次获取新的props或state之后；每次接收新的props之后都会返回一个对象作为新的state，返回null则说明不需要更新state；
3. render(): 创建虚拟dom，进行diff算法，更新dom树
4. componentDidMount(): 组件渲染之后调用，只调用一次，一般在这个钩子中做一些初始化的事，例如：开启定时器、发送网络请求、订阅消息

2. 更新时

> 由组件内部this.setSate()或父组件重新render()时触发

1. static getDerivedStateFromProps(props, state): 同上
2. shouldComponentUpdate(nextProps, nextState): 控制组件更新的"阀门", 组件接收到新的props或者state时调用，return true就会更新dom（使用diff算法更新），return false能阻止更新（不调用render）
3. render(): 同上
4. getSnapshotBeforeUpdate(preProps, preState): 触发时间: update发生的时候，在render之后，在组件dom渲染之前；返回一个值，作为componentDidUpdate的第三个参数；
5. componentDidUpdate(preProps, preState, snapshotValue): 组件更新完成后调用

3. 卸载时

> 由ReactDOM.unmountComponentAtNode()触发

1. componentWillUnmount(): 组件将要卸载时调用，一般在这个钩子中做一些收尾的事，例如：关闭定时器、取消订阅消息

2. 路由

1. 前端路由原理

1.1 前端路由的诞生缘由

前端路由的出现要从 ajax 开始，它是浏览器用来实现异步加载的一种技术方案。

- **多页面应用**: 网页都是通过直接返回 HTML，用户的每次更新操作都需要重新刷新页面。极其影响交互体验，随着网络的发展，迫切需要一种方案来改善这种情况。
- **Ajax(Asynchronous JavaScript And XML)**: 微软首先提出 iframe 标 签，iframe 带来了异步加载和请求元素的概念，随后微软的 Outloook Web App 团队提出 Ajax 的基本概念（XMLHttpRequest 的前身）， 并在 IE5 通过 ActiveX 来实现了这项技术。在微软实现这个概念后，其他 浏览器比如 Mozilia，Safari，Opera 相继以 XMLHttpRequest 来实现 Ajax。 有了 Ajax 后，用户交互就不用每次都刷新页面， 体验带来了极大的提升。
- **SPA(Single Page Web Application)**: 单页应用不仅仅是在页面交互是无刷新的，连页面跳转都是无刷新的，为 了实现单页应用，所以就有了前端路由。单页应用的概念是伴随着 MVVM 出现的。最早由微软提出，然后他们在浏览器端用 `Knockoutjs` 实现。但这项技术的强大之处并未当时的开发者体会到，可能是因为 `Knockoutjs` 实现过于复杂，导致没有大面积的扩散。同样，这次接力的选手依然是 Google。Google 通过 Angularjs 将 MVVM 及单页应用发扬光大，让前端 开发者能够开发出更加大型的应用，职能变得更大了随后都是就是前端圈 开始得到了爆发式的发展，陆续出现了很多优秀的框架。

1.2 什么是前端路由

前端路由就是把不同路由对应不同的内容或页面的任务交给前端来做，之前是通过服务端根据 url 不同返回不同的页面来实现。

前端路由的实现其实很简单。本质上就是检测 url 的变化，截获 url 地址，然后解析来匹配路由规则。

<Route path="/test" component={Test}>，当浏览器的path变为/test时，当前路由组件就会变为Test组件

1.3 前端路由的基石-history

利用 H5 的 history.pushState 和 history.replaceState ，这两个 history 新增 的 api，为前端操控浏览器历史栈提供了可能性 这两个 Api 都会操作浏览器的历史栈，而不会引起页面的刷新。 不同的是，pushState 会增加一条新的历史记录，而 replaceState 则会替换 当前的历史记录。

2. React路由的基本使用

2.1 react-router和react-router-dom区别

- react-router: 实现了路由的核心功能。
- react-router-dom: 基于react-router，加入了在浏览器运行环境下的一些功能。

react-router-dom 可以看做是 react-router 的加强版，所以使用的时候使用 react-router-dom即可

2.2 react-router-dom相关API

2.2.1. 内置组件

- <BrowserRouter>
- <HashRouter>
- <Route>
- <Redirect>
- <Link>
- <NavLink>
- <Switch>

2.2.2. 其他

- history对象
- match对象
- withRouter函数

2.3 react路由的基本使用

- 最外侧使用<BrowserRouter>或<HashRouter>标签包裹
- React中路由链接实现切换组件--使用<Link>或<NavLink>编写路由链接

- 使用<Route>注册路由

3. React路由组件和一般组件

1. 写法不同
   一般组件：<Demo/>
   路由组件：<Route path="/demo" component={Demo}/>
2. 存放位置不同
   一般组件：components
   路由组件：pages
3. 接收到的props不同
   一般组件：写组件标签时传递了什么，就能收到什么
   路由组件：接收到三个固定的属性

```
history:
    go: ƒ go(n)
    goBack: ƒ goBack()
    goForward: ƒ goForward()
    push: ƒ push(path, state)
    replace: ƒ replace(path, state)
location:
    pathname: "/demo"
    search: ""
    state: undefined
match:
    params: {}
    path: "/demo"
    url: "/demo"
```

4. NavLike的使用

<NavLink>是<Link>的一个特定版本，会在匹配上当前的url的时候给已经渲染的元素添加参数，组件的属性有：

- activeClassName(string)：设置选中class，默认值为active
- activeStyle(object)：当元素被选中时，为此元素添加样式
- exact(bool)：为true时，只有当导致和完全匹配class和style才会应用
- strict(bool)：为true时，在确定为位置是否与当前URL匹配时，将考虑位置pathname后的斜线

- isActive(func)判断链接是否激活的额外逻辑的功能

5. Swith的使用

用于渲染与路径匹配的第一个子 <Route> 或 <Redirect>。Switch可以提高路由匹配效率(单一匹配)。

6. Redirect的使用

使用 <Redirect> 会重定向到一个新的位置。新的位置将覆盖历史堆栈中的当前条目。

7. 嵌套路由的使用

- 注册子路由时要写上父路由的path值
- 路由的匹配是按照注册路由的顺序进行的

8. 向路由组件传递参数

8.1 params参数

路由链接(携带参数)：<Link to='/demo/test/tom/18'}>详情</Link>
注册路由(声明接收)：<Route path="/demo/test/:name/:age" component={Test}/>
接收参数：this.props.match.params

8.2 search参数

路由链接(携带参数)：<Link to='/demo/test?name=tom&age=18'}>详情</Link>
注册路由(无需声明，正常注册即可)：<Route path="/demo/test" component={Test}/>
接收参数：this.props.location.search
备注：获取到的search是urlencoded编码字符串，需要借助querystring解析

8.3 state参数

路由链接(携带参数)：<Link to={{pathname:'/demo/test',state:{name:'tom',age:18}}}>详情</Link>
注册路由(无需声明，正常注册即可)：<Route path="/demo/test" component={Test}/>
接收参数：this.props.location.state
备注：刷新也可以保留住参数

9. 编程式路由导航

借助this.prosp.history对象上的API对操作路由跳转、前进、后退

- this.prosp.history.push()
- this.prosp.history.replace()
- this.prosp.history.goBack()

- this.prosp.history.goForward()
- this.prosp.history.go()

10. withRouter的使用

- withRouter可以加工一般组件，让一般组件具备路由组件所特有的API
- withRouter的返回值是一个新组件

11. useHistory的使用

从React Router v5.1.0开始，新增了useHistory钩子（hook），在React 16.8.0及以上的函数式组件中可以使用

3. ReduX

4. 其他

4.1 Hooks

参考链接:

React官方文档
尚硅谷2021版React技术全家桶全套完整版 – bilibili
React.Component – 官方文档
React生命周期图谱 – 官方文档
react 生命周期执行顺序，render执行条件 – 简书

React Router 官方中文文档
React Router 中文文档
React Router 中文文档（一）– segmentfault
React Router 中文文档（二）– segmentfault
什么是前端路由? – CSDN
React Router原理 – 简书