## React 学习

一jsx语法规则：

1. 定义虚拟DOM时，不要写引号
2. 标签中混入js表达式时要用{}
3. 样式的类名指定不要用class，要用className
4. 内联样式，要用style={{}}的形式去写
5. 只有一个跟标签
6. 标签必须闭合
7. 标签首字母（若小写字母开头，则将该标签转为html中同名元素，若html中无该标签对应的同名元素，则报错，若大写，react就去渲染对应的组件，若组件没定义，则报错）
8. 函数式组件
9. 首字母必须大写
10. 必须有return返回值
11. 渲染组件到页面时必须是闭合的标签（<Demo/>）
12. 函数组件的this是undefined ,因为通过babel编译后，开启了严格模式
13. ReactDom.render()
14. react解析组件标签
15. 发现组件时使用函数定义的，随后调用该函数，将返回的虚拟DOM转为真实的DOM，随后呈现在页面中
16. 类

// 创建一个Person 类

class Person {

// 构造器方法

constructor(name, age) {

// this指的是类的实例对象

this.name = name

this.age = age

}

// 一般方法

speak () {

//speak ()方法放在Person 类的原型对象上，供实例使用

// 通过Person 实例调用speak()时，speak()中的this就是Person 实例

Consoloe.log(this.name,this.age)

}

}

// 创建一个Person 的实例对象

const p1 = new Person(‘tom’,18)

const p2 = new Person(‘jerry,19)

p1.speak ()

P2.speak ()

1. 类的继承

//创建一个Student类，继承Person 类

class Student extends Person {

constructor(name, age,grade) {

//super代表父类的构造函数，只能放在this之前

super(name,age)

this.grade = grade

}

speak () {

Consoloe.log(this.name,this.age, this.grade )

}

Study() {

Consoloe.log(55555 )

}

const s1 = new Student (‘小泽’, 14,’高一’)

s1.speak () // 获取父类的方法

1. 类式组件
2. 类名就是组件名
3. 继承React.Component
4. 必须要有render,必须要有return返回值
5. 渲染组件到页面React.React()
6. render放在类的原型对象上，供实列使用

class Cap extends React.Component {

//render放在Cap 的原型对象上，供实列使用

//render中的this指的是Cap组件实列对象

render () {

}

}

ReactDOM.render(<Cap/>,document.getElementById(‘test’)

ReactDOM.render执行之后，发生了什么？

1. React解析组件标签，找到了Cap组件
2. 发现组件是使用类定义的，随后new出来该类Cap的实列，并通过该实列调用的原型上的方法
3. 将render返回的虚拟DOM转为真实DOM
4. 组件有state就是复杂组件(类组件)，否则就是简单组件（函数组件）
5. 组件实列的三大核心属性（state、props、refs）
6. State状态（是组件类实列上的）
7. state是组件对象最重要的属性，值是对象
8. 组件被称为‘状态机’，通过更新组件的state来更新对应的页面显示
9. 组件中的render方法中的this为组件实列对象
10. 组件自定义的方法中this为undefined。可以通过强制绑定this，通过bind()，或者箭头函数
11. Props
12. 每个组件对象都会有props（properties的简写）属性
13. 组件标签的所有属于都保存在props
14. 通过标签属性从组件向外组件内传递变化的数据
15. 组件内部不能修改props数据
16. Refs
17. 什么是高级函数
18. 若A函数，接收的参数是一个函数，那么A就可以称之为高阶函数
19. 若A函数，调用的返回值依然是一个函数，那么A就可以称之为高阶函数
20. 组件生命周期
21. Constructor 最先调用的是构造器
22. ConponentWillMount 组件将要挂在的钩子
23. Render 初始化渲染、状态（state）更新之后（1+n次，能代表state改变的次数）
24. componentDidMount 组件挂载完毕后调用 （只调一次）
25. ShouldComponentUpdate 控制组件更新的‘阀门’，返回true继续，返回false暂停
26. ComponentWillUpdate 组件将要更新的钩子
27. ComponentDidUpdate 组件更新完毕的钩子
28. ComponentWillUnmount 组件将要卸载时调用
29. ForceUpdate()强制更新（流程： ForceUpdate()—》ComponentWillUpdate—》Render—》ComponentDidUpdate—》ComponentWillUnmount）
30. 父组件render （）
31. ComponentWillReceiveProps() 组件将要接收新的props的钩子
32. 生命周期的三个阶段(旧的)
33. 初始化阶段：右ReactDom.render()触发---初次渲染
34. constructor()
35. componentWillMount()
36. Render()
37. componentDidMount()
38. 更新阶段：由组件内部this.setState()或父组件重新render触发
39. shouldComponentUpdate()
40. componentWillUpdate()
41. Render()
42. componentDidUpdate()
43. 卸载组件：由ReactDom.unmountComponentAtNode()触发
44. componentWillUnmount()
45. 生命周期(新版本)
46. componentWillMount() 、componentWillReceiveProps()、componentWillUpadate() 这三个在新的react生命周期中要加UNSAFE前缀(UNSAFE\_componentWillUpadate())
47. 生命周期(新版本)

1、constructor()

2. getDerivedStateFromProps() 从props得到一个衍生的状态

使用场景：若state的值在任何时候都取决于props

语法：static getDerivedStateFromProps(props, state) {

return props

}

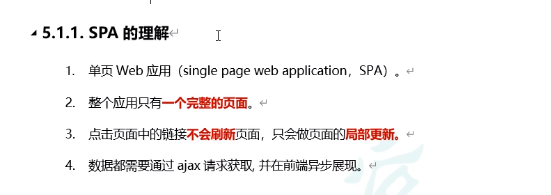
1. getSnapshotBeforeUpdate() 在最近一次渲染输出（提交到DOM节点）之前调用

语法：getSnapshotBeforeUpdate(props, state) {

return null

}

1. 路由（一个路由就是一个映射关系，key为路径，value可能是function或component）
2. 前端路由主要是靠BOM（Window）的history属性
3. React-router-dom
4. SPA(单页面应用)



1. 路由组件与一般组件的区别
2. 写法不同
3. 一般组件：<Demo/>
4. 路由组件：<Route path=’/demo’ component={Demo}>
5. 存放位置不同
6. 一般组件：存放在components
7. 路由组件：存放在pages

（3）接收到的props不同

A、一般组件：写组件标签时，传递了什么就接收什么

B、路由组件：固定的三个属性：（history、location、match）

History：

go

goBack

goForward

push

replace

Location:

Hash

Key

Pathname

Search

State

Match:

isExact

Params:{}

Path

Url

1. Switch: 可以提高路由匹配效率
2. 解决多级路劲刷新页面样式丢失的问题
3. 在index.html中引入样式的时不写 ./ 写/
4. 在index.html 中引入样式时不写 ./ 写%PUBLIC\_URL%
5. 使用HashRouter
6. 路由的严格匹配与模糊匹配
7. 默认使用的是模糊匹配
8. 开启严格匹配（exact={true}）
9. 严格匹配不要随便开启
10. Redirect 重定向
11. 一般写在所有路由最下方，当所有路由都无法匹配时，跳转到Redirect 指定的路由
12. 具体编码：

<Switch>

<Router path=’’/>

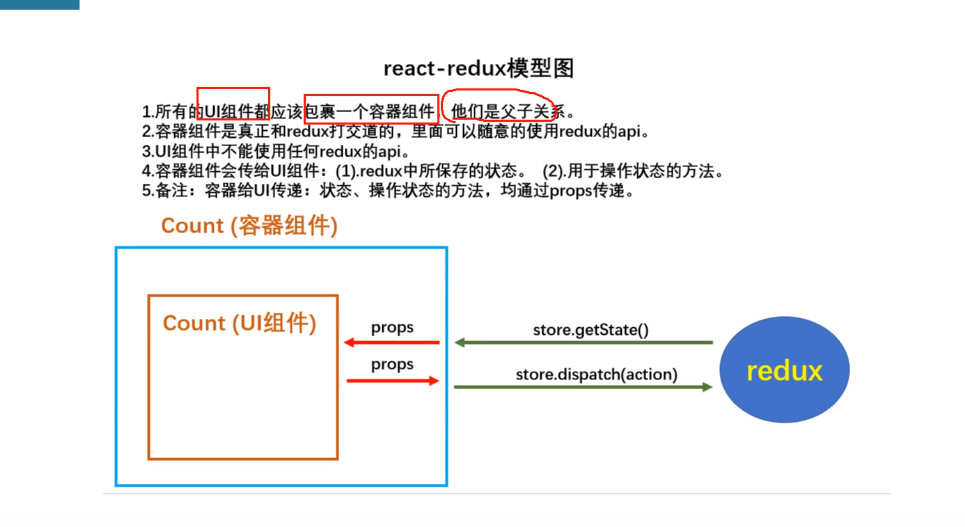
<Redirect to=’’>

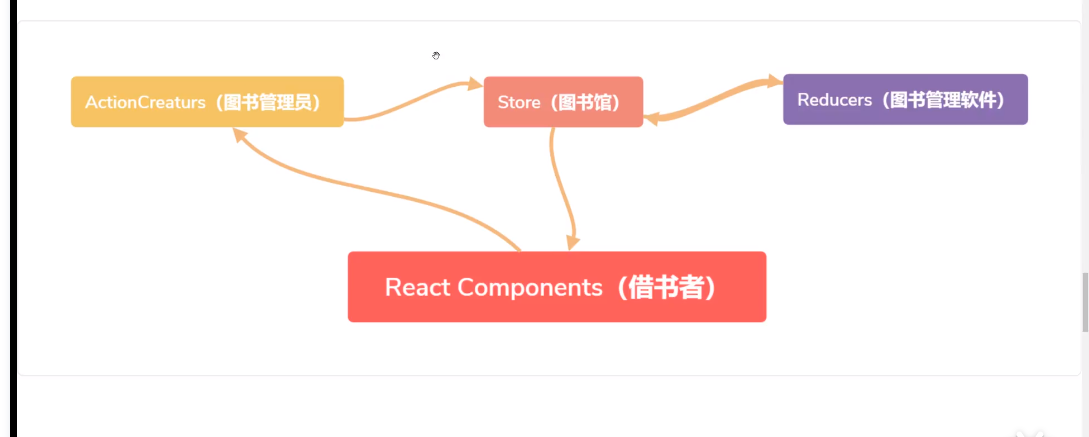
</Switch>

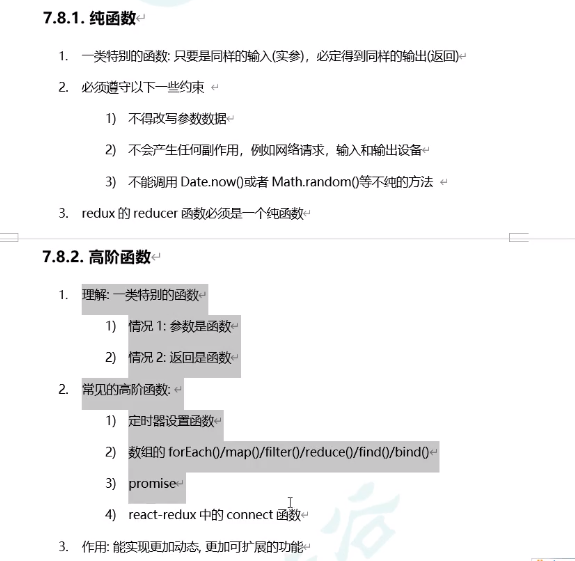
1. 嵌套路由
2. 注册子路由时要写上福路由的path值
3. 路由的匹配是按照注册路由的顺序进行的
4. 向路由组件传递参数
5. params 参数
6. 路由链接（携带参数）
7. 注册路由（声明接收）
8. 接收参数 ：const list = this.props.match.params
9. search 参数
10. 路由链接（携带参数）
11. 注册路由（无需声明）
12. 接收参数：const list = this.props.location.search
13. 注意：接收到的search是urlencoded编码字符串，需要借助querystring解析
14. state 参数
15. 路由链接（携带参数）
16. 注册路由（无需声明）
17. 接收路由const list = this.props.location.state
18. 路由跳转push与repalce
19. push与repalce的区别：push会缓存路由，repalce不会
20. withRouter 让一般组件具备路由组件所特有的api

（1）withRouter 返回的值是一个新组建

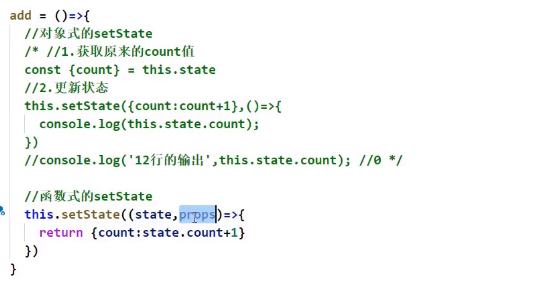
1. BrowserRouter与HashRouter
2. 底层原理不一样：
3. BrowserRouter 是使用H5的history API，不兼容IE9以下的版本
4. HashRouter 使用的是URL的哈希值
5. Ur表现形式不一样
6. BrowserRouter 的路劲中没有#
7. HashRouter 的路劲中有#
8. 刷新后对路由的state参数的影响
9. BrowserRouter 没有任何影响
10. HashRouter 刷新会导致state参数丢失
11. 备注：HashRouter可以用来解决一些路劲错误相关的问题
12. Redux
13. redux的三大核心概念：
14. Action 动作对象 包括两个属性
15. type 标识属性，值为字符串，唯一，必要属性
16. Data 数据属性，值类型任意，可选属性
17. reducer
18. 用于初始化状态、加工状态
19. 加工时，根据旧的state和action，产生新的state的纯函数
20. Store
21. 将state、action、reducer 联系在一起的对象
22. react-redux







1. setState
2. setState更新状态的两种写法：
3. 、setState(stateChange, [callBack]) ----- 对象式的setsState
4. stateChange 为状态改变对象（该对象可以体现出状态的更改）
5. Callback是可选的回调函数，它在状态更新完毕、界面也更新后（render调用后）才被调用
6. setState（updater， [callback]） --- 函数式的setState
7. Updater 为返回stateChange 对象的函数
8. Updater 可接收state和props
9. Callback 是可选的回调函数，它在状态更新、界面更新后（render调用后）才被调用
10. 总结
11. 对象式的setState是函数式setState的简写方式（语法糖）
12. 使用原则：
13. 如果新状态不依赖与原状态 -- 使用对象方式
14. 如果新状态依赖与原状态 -- 使用函数方式
15. 如果需要在setState（）执行后获取最新的状态数据，要在第二个callback函数中读取



1. lazyLoad 路由组件

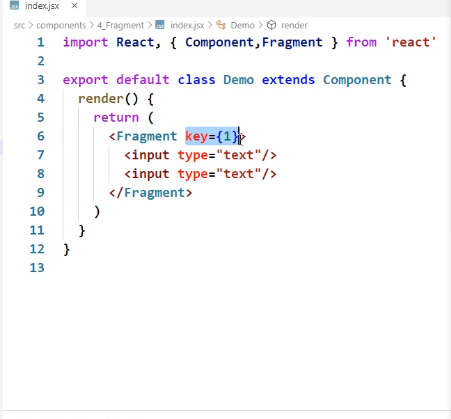


1. Hooks





1. Fragment 可以不用必须有一个真实的DOM根标签



1. context

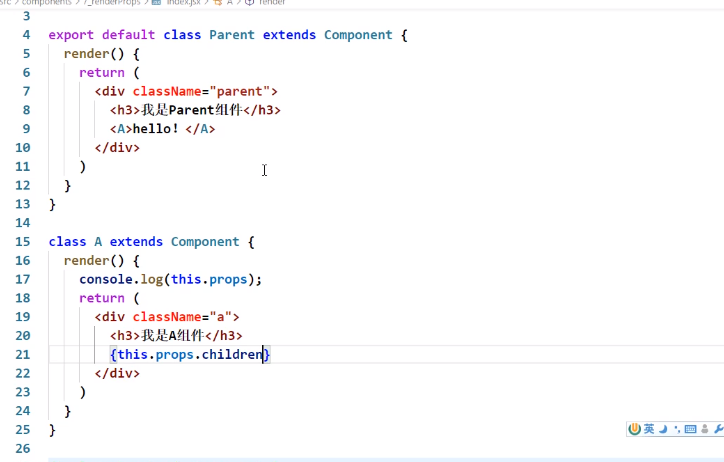


二十、组件优化 （pureComponent）

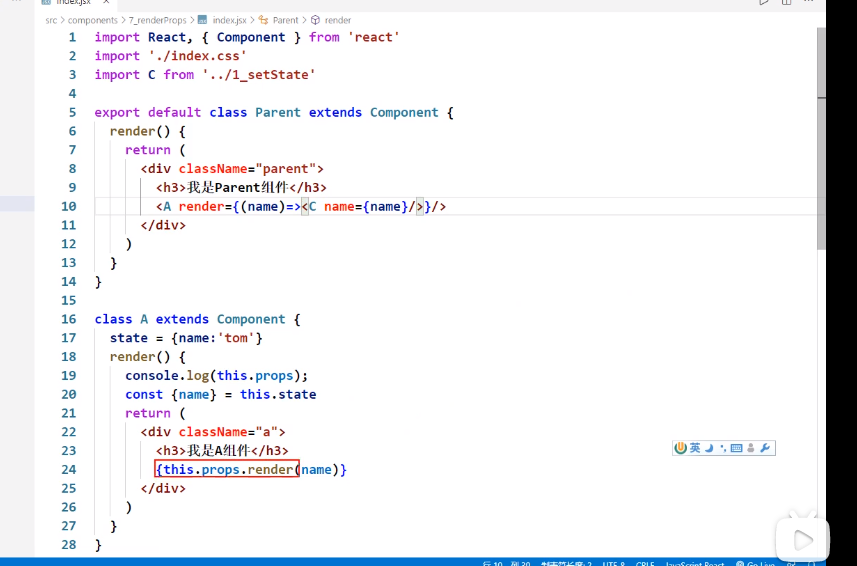


二十、renderProps

<A>hello!</A> A组件这样传参，在子组件通过this.props.children方法获取



<A render = {(name)=><C name = {name}/>}/> 向C组件传递参数



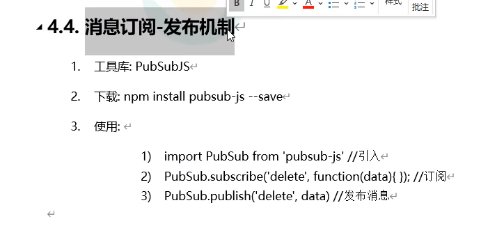
二十一、错误边界



二十二、组件通信方式总结



二十三、消息订阅-发布机制



二十四、fetch 发送请求

1. fetch与xhr（XMLHttpRequest）是平级的
2. Jq与axios是对xhr的封装
3. Fetch是内置的，是Windows内置对象

