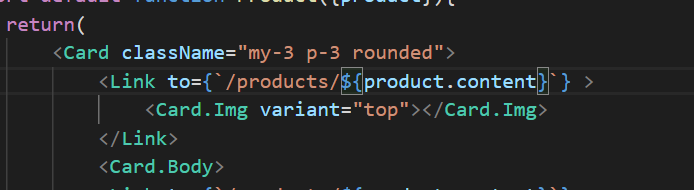
1. 虚拟DOM需要使用容器包括起来（不一定要在标签里面），render函数中第二个参数要指定容器标签，不能直接放在body里面
2. 一个组件只有一个根标签（和容器标签不是同一个）
3. React组件名首字母都大写，写的html标签都会被babel转换为html标签（都是首字母小写），即首字母小写的标签会被当作html标签，首字母大写的标签会被当成react标签或自定义标签
4. 插值表达式只能是表达式，不能是循环。表达式即存在返回值的函数调用和变量使用等等
5. 使用propstype库限制属性类型
6. 类中加static作为类的静态属性，类似类名.props，this.state是实例对象属性
7. Constructor中传入props，super(props)的作用是把父类的props和其它属性、方法传递给子类，并且能访问到
8. React内联函数是在节点属性中定义的函数，内联函数形式的ref在重新render后会调用两次
9. jsx写注释用大括号，里面写js注释，原理是大括号内的是js表达式，表达式的内容是js注释
10. ref和ReactDOM.findDOMNode：前者获取组件实例和dom节点，后者获取dom节点和组件对应的dom
11. 受控组件与非受控组件
12. Web事件都会接收一个event对象
13. React组件的事件有内联式函数和类属性式函数，内联式本身作为一个函数监听
14. React组件的内置方法定义时不需要使用function声明或箭头函数声明，相当于重写方法
15. React can’t perform a state updata。。。。。。，组件被卸载后仍然有改变组件状态的回调在执行
16. React生命周期函数等内置函数不需要function或箭头函数声明，相当于重写方法，不写也有默认值
17. React新生命周期更改了willMount、willUpdate、willReceiveProps的名字，并使用新的生命周期函数代替 。
18. 新生命周期中上面的是静态方法，接收参数作为state的值且为固定值（常量），下面的是返回参数传递给Didmount
19. React项目打包后交给后端开发？public文件夹下的是静态资源，index.html最主要，里面的div作为容器
20. Index.html是展示的页面，index.js 是入口文件，webpack自动找到index.html文件中的root标记的div标签
21. React前后端联合开发，react底层原理（webpack），webpack打包项目后部署到后端，webpack把浏览器无法解析的文件打包成浏览器可以解析的文件并输出到指定位置
22. React项目已配置好服务器、自动执行index.js文件、自动寻找HTML文件的root节点并把组件渲染在节点中
23. React导入时{Component}导入Component，不是解构赋值，而是react导出了Component
24. React项目的每个组件可以放在一个文件夹下，包括样式文件和资源文件，也可以把每个组件都定义为index.js（放在了不同的文件夹下），导入时会默认导入index.js文件
25. React样式如果使用css的方式，当几个组件导入同一个组件中时样式的定义会产生冲突，解决方案：样式文件模块化即：index.module.css，或使用less方式定义
26. Css文件导入直接import ‘index.css’，导入即可使用
27. Vscode中，蓝色标签、小写字母开头是html标签
28. 使用main标签放置网页主要内容，同时main也可以单独设置样式
29. rem:浏览器都有默认字体大小（font-size值），不设置fontsize则默认标签的fontsize值为16px，所以有1em=16px，rem表示某标签相对于根元素的fontsize值的大小，em表示某标签相对于父元素fontsize值的大小
30. JavaScript数组map方法：接受一个回调函数，回调函数的参数是数组的每个元素，返回值是一个新的数组，元素是经过回调函数处理后的值。React中使用数组的map方法渲染页面原理：返回了一个jsx对象的数组，所以能显示页面
31. Row、col参数的作用，bootstrap使用，JavaScript、react已没问题
32. React函数组件接收参数通过props或{参数名}
33. 标签属性使用模板字符串不起作用加上{}
34. 使用数组map方法渲染内容需要加上key属性
35. 函数式组件设置默认props：组件名. defaultProps={}，默认props都是从父组件接受的props，设置一个初始值，当父组件没有传递值时使用默认值。如果是组件本身的props可直接设置初始值
36. 实现动态添加内容的列表之类的组件不一定要用flatlist组件，可以用数组的map方法实现渲染每一个元素
37. react-router-dom：BrowserRouter包裹，route设置路由页面，path属性设置路由。Link标签设置跳转，to属性设置跳转路由。path设置为‘/’表示当前显示该组件，下面的ProductScreen组件也显示在这个位置，只是在路由没有转换到对应路由时不会显示。Path设置为‘products/：content’：，冒号后面的是可变动的内容， 如：products对应固定的，后面的模板字符串中content属性名对应:content，动态可变
38. 数组的find方法，参数为一个函数，遍历数组元素并把元素作为函数参数，找出符合条件的元素，返回新数组
39. Reactpro后端：更改项目结构（新建后端文件夹和前端文件夹），根目录下添加yarn文件（方便启动服务），设置yarn的脚本，编写服务端代码。注意前端导出导入和后端导出导入的区别
40. Link和node中的路径都可以用:，冒号的作用是使用参数，实际使用的参数和路径中定义的参数名不一定相同，只是会把传入的参数值作为路径名
41. React跨域，axios基本使用，async await，hooks，
42. ！！！！！\*\*\*\*\*React函数组件接受的props为内置属性，其中的match属性保存路由相关的信息（match相当于请求req），match中的路由信息来源于Link标签跳转的路由值，例如使用route设置路由时设置的参数（：参数名）会保存在match的params属性中（此时组件类似一个服务器，注意：url中的带冒号的参数都作为请求参数），例如reactpro项目中的id（match是使用router后传递给组件props的）[React-Router传递给组件的props中有什么 - 简书 (jianshu.com)](https://www.jianshu.com/p/f37702ca4e33)
43. Node返回对象？axios使用？
44. 原生重新绘制：替换dom树或替换dom节点。React重新绘制：替换虚拟dom中的不同节点（此节点和dom节点不是一个概念）。
45. React的虚拟dom就是一个jsarray对象，操作js对象的开销远小于操作dom节点的开销，所以性能提升了。
46. React的jsx语法中的html标签不是dom节点！！！只是用来生成虚拟dom的。
47. React ref获取组件实例，如果是react组件获取的是实例，如果是html标签获取的是dom节点，对性能消耗较大，尽量少用
48. Node使用一定格式返回的数据确实是这种格式，在浏览器中获取时是这种格式，比如res.json返回json格式数据。只是用axios或其它API获取数据时对数据格式进行了处理，不一定是json格式的数据
49. React函数式组件父组件向子组件传值，可以在props对象中解构，也可在参数中结构（用{参数名}方式）
50. React+react-redux：provider作为包裹app，传入store。Store传入reducer和initialstate。数据并没有固定存在哪个地方，但数据是reducer返回的，action单独创建，向action函数中传入dispatch函数，通过调用action函数间接调用dispatch函数向reducer传递数据，reducer数据更改，store更改
51. React函数式组件使用hook就是使用一些函数如useState、useEffect等，useEffect是挂载后调用，和componentDidMounted一样
52. 注意！！！！！：函数式组件用useEffect模拟hook，刷新UI只有当state和props变化时
53. 函数式组件：变量更新不会刷新UI
54. Redux：store中state变化时会触发state监听函数（useSelector和mapStateToProps），
55. **函数式组件生命周期相关内容：**useState的目的是把变量当作类组件的state，更新值时重新渲染。UseEffect的目的是模拟生命周期，每次渲染UI时根据第二个参数是否变化决定是否执行。
56. 类组件props不可变（props变化重新渲染UI指的是当父组件传过来的props变化时）。类组件和函数组件中定义的变量变化时都不会重新渲染UI！！！！！
57. 函数式组件没有生命周期和props，用hook模拟生命周期和state，用redux和react-redux模拟props，redux的useSelector和mapStateToProps把redux的state转换为组件的props，props变化时也会刷新UI。Reactpro项目HomeScreen文件的刷新就是props变化触发的
58. Redux的action获取的数据也可能是UI层传来的，通过定义action函数的参数
59. Redux的action向store发送dispatch时不会指定向哪个reducer发送，会向总的reducer发送action（combineReducers函数结合的），哪个reducer的type对应了就执行哪个语句
60. 箭头函数返回一个jsx对象用小括号
61. React的事务就是一个包裹react封装的事件的对象，有initialize和close两个钩子函数，react封装的事件触发时，会调用钩子函数，开始时把batchingStrategy设为true，所以setState异步，而使用settimeout或addEventListener设置函数时，使用setState是同步
62. React/vue列表渲染使用key的时候会绑定渲染元素。没有key的情况下就是一些数据和组件的组合，有key的情况下就是把这些数据和组件组成了一个对象。没有key时：更改元素内容或顺序，就是把元素内容换成新的顺序的内容，组件本身并没有动。有key时：顺序改变会把DOM对象顺序改变。（增删情况再分析）
63. React使用bind绑定函数的时候，最好在构造函数中绑定，返回新的函数，作为参数传给组件。因为bind是返回改变this后的新函数，call和apply是绑定然后执行
64. React事件机制：react合成事件绑定到document(文档对象)上，并不是每个节点上，且事件参数是封装后的(和DOM原生事件参数不同)。事件触发时，事件冒泡至document上时才会触发回调，顺序是内层先外层后。DOM原生事件还是再节点上，冒泡至响应节点即执行
65. DOM事件流：三个阶段：事件捕获，事件接收(触发事件的节点接收到传过来的事件)，事件冒泡
66. setState多次使用且操作相同时，传入对象会被覆盖，传入函数不会被覆盖，解释：传入对象时，会生成一个回调函数，如果多次的操作相同，则生成一个回调，即只改变一次state；传入函数时，每个函数都会被加入任务队列，每个都会执行，即多次改变
67. setState异步同步要看它最近一层的函数是react合成事件所属的还是原生JavaScript函数。传入对象和传入函数的区别和异步同步结合的代码
68. HashRouter原理：通过原生hashChange事件控制DOM操作界面组件变化。每次改变URL不会向服务端发请求，会触发hashChange事件。
69. BrowserRouter原理：封装history对象，刷新页面会存储其他页面传递的参数
70. useRef：作用一是绑定一个DOM元素，和直接给组件赋ref属性值一样。返回的是一个只有current属性的对象，**current属性的类型和传入的参数类型一样（其实就是传入的参数）。绑定一个组件时直接给组件赋ref属性。**
71. **useRef的第二个作用：传入的参数是current属性的值，可以用来保存上一个值，useRef返回的对象不会在每次刷新DOM后更新，即除非手动更新（通过.current的方式更新），否则不会更新**
72. **useEffect返回的函数在下一次useEffect执行之前执行**
73. **useEffect是在DOM挂载后执行！返回的函数也是**
74. **JSX中不仅可以注释、执行map渲染，也可以进行输出，只要是JS语句都可以**
75. **React父组件re-render，子组件re-render情况总结：父子组件都是类组件时，父组件不管有没有给子组件传值，父组件re-render都会导致子组件re-render（切记：子组件re-render条件是父组件re-render或自己调用setState）。父子组件是函数组件时，父组件向子组件传值时，父组件re-render时不管传递的值有没有变化都会导致子组件re-render，不想让子组件re-render就用useCallback和memo；父组件没有向子组件传值时，父组件re-render不会导致子组件re-render**
76. **项目结构问题：页面组件可以用page文件夹管理，下面放多个文件夹，index入口文件，该页面相关组件也放进去。Component文件夹放一些和页面无关的组件，无关并不是说非UI组件，而是不和主要UI相关，如Header和Footer，Toast等较独立的组件和公共的组件**
77. **react函数组件：父组件re-render，并且给子组件传递了值，子组件re-render的原因可能是：re-render时重新给子组件传值调用，导致子组件re-render，使用useCallback的原理可能是缓存起来，父组件re-render时不重新调用子组件**