**React Hook**

React hook是react 16.8新增特性，可以在不编写class的情况下使用状态state,组件尽量写成纯函数，如果需要外部功能或者副作用就用钩子（react hook）钩进来。

自定义的hook是一个函数，命名以use开头，如果不用use开头react就无法判断某个函数是否包含对其内部hook的调用，也无法自动检查你的hook是否违反了规则

**1、优缺点**

优点：代码可读性更强，方便阅读和维护

组件树层级变浅，更容易复用代码，代码量少

不需要考虑this指向问题

缺点：状态不同步（可以理解为闭包）：每个函数都有一份独立的作用域，函数的变量是保存在运行时的作用域里，异步操作时，经常会碰到异步回调的变量饮用是之前的

**hook常用的钩子（自定义钩子还在学习中。。。）**

3、**useState（）**状态钩子---给函数引入状态



Example组件是一个函数，内部使用了useState（）钩子引入了状态，声明了一个变量number初始值是0，返回的是一个数组，第一个是变量即count，第二个是状态的变量名setCount

下面附上相同案例的class写法

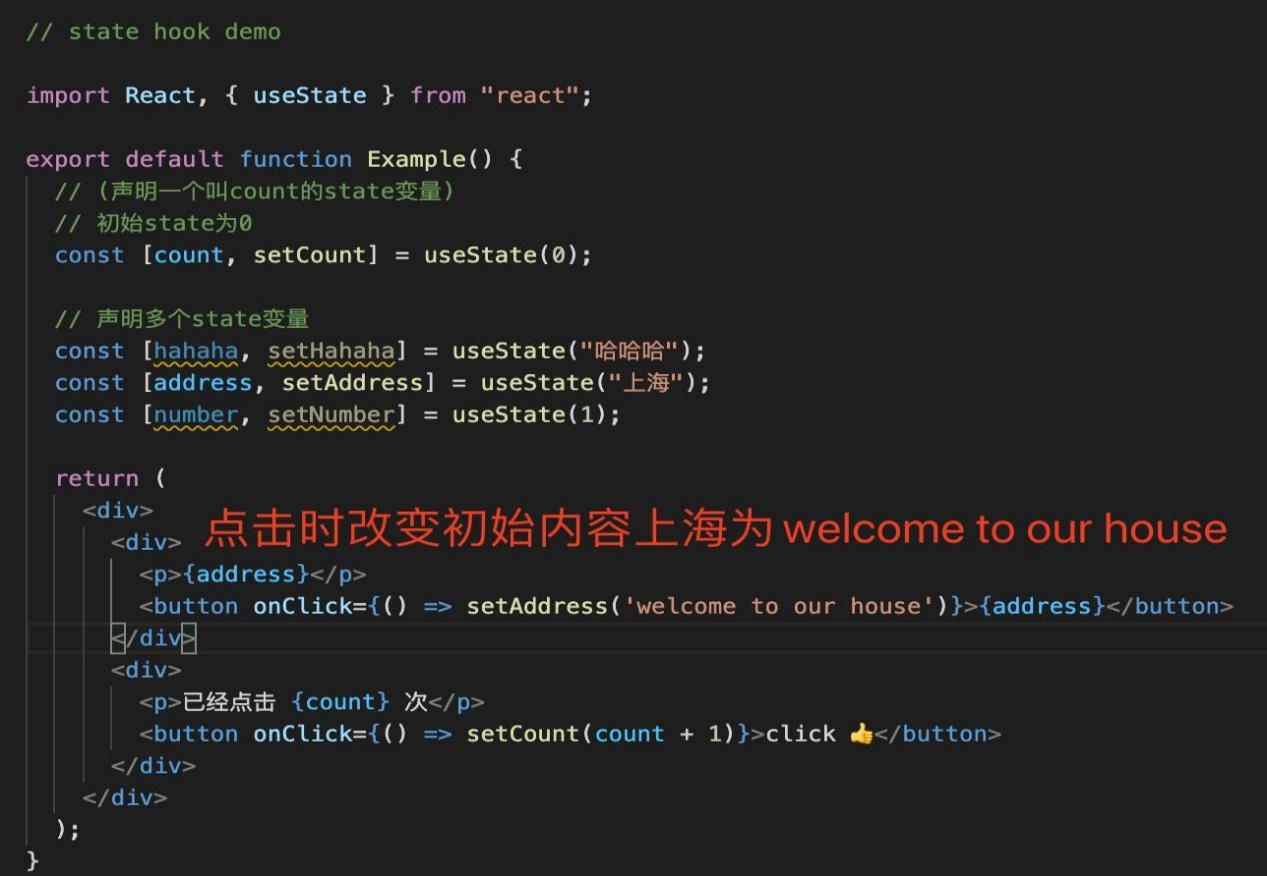


两种写法最直观的感受就是函数组件的代码要少一些，没有this，在class中显示当前的number要读取this.state.number，在函数组件里可以直接用number,不需要考虑this（更新也是同理）.

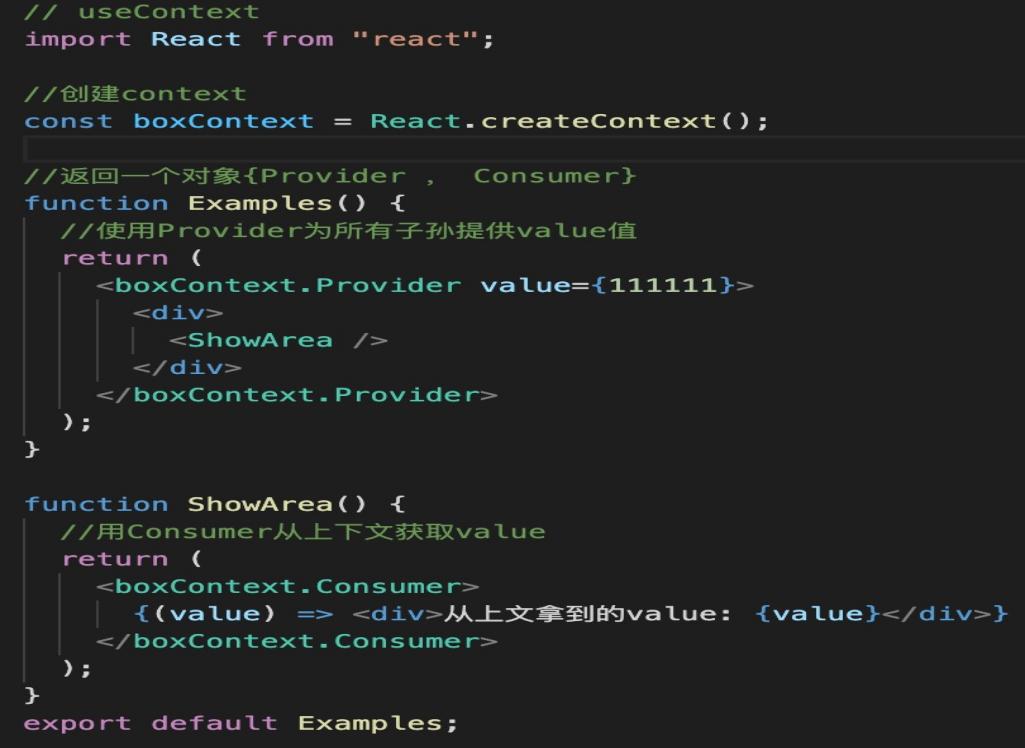
1）useState作用

useState在函数组件里调用它来给组件添加一些内部的state，它唯一的参数就是初始的state,在上面例子中初始值是0（初始的state参数只有在第一次渲染的时候会用到）,react会在重复渲染时保留这个state，useState会返回一个数组：[当前state，更新state的函数]。

在一个组件中声明多个state变量，可以给不同的state变量取不同的名字，并单独去更新他们



4、**useContext（）**---共享钩子状态，组件之间共享状态，可以解决props传递数据的问题



5、**useReducer（）** ---action钩子

useState的替代方案，接受一个形如（state, action）=> newState的reducer,返回一个数组[当前的state，发送action的dispatch方法]

const [state, dispatch] = useReducer(reducer, initialArg,init);

1）reducer适用场景

state是一个数组或者对象

state逻辑复杂

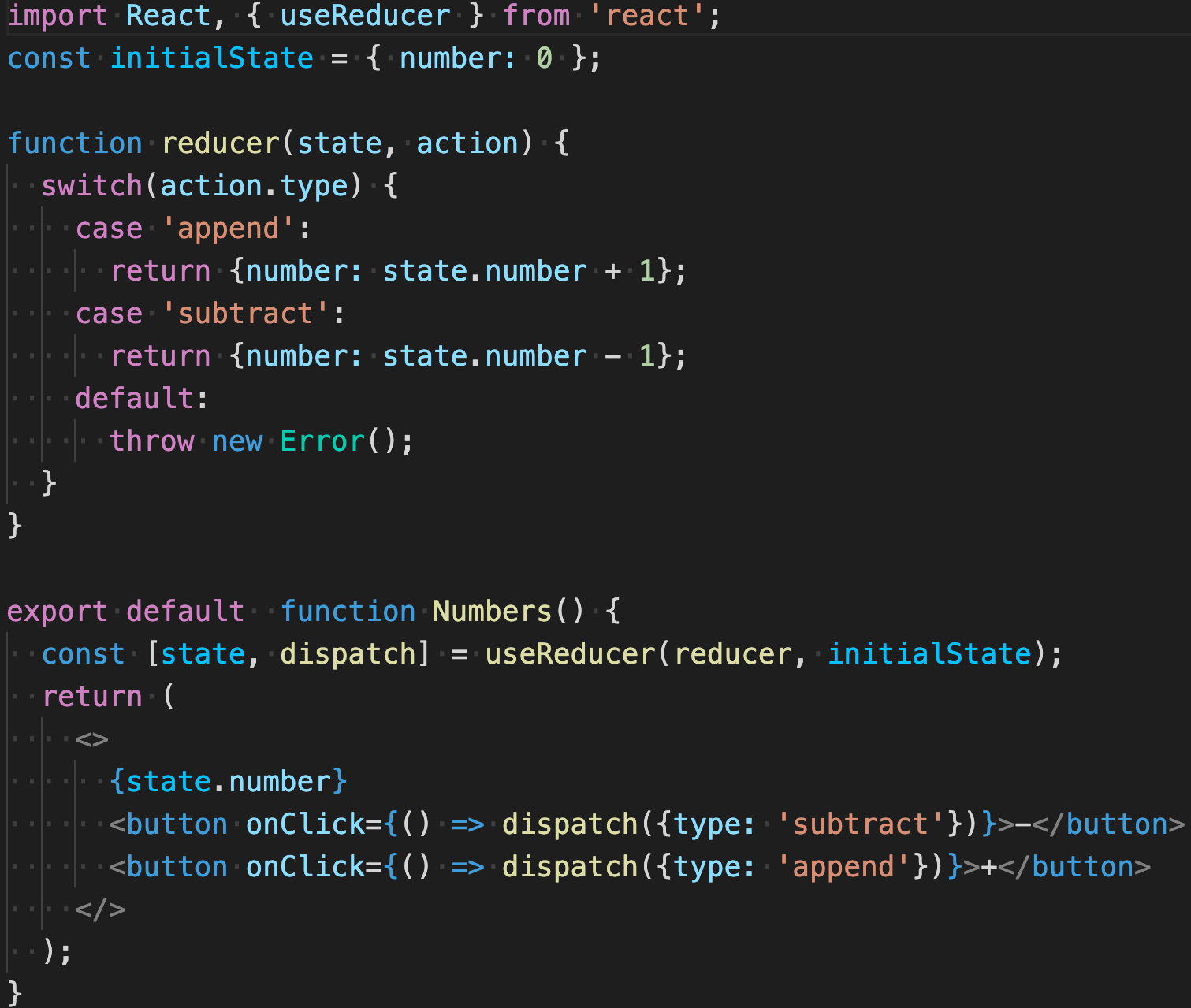
需要在深层子组件里修改状态

2）优势

当状态更新逻辑复杂时reducer比setState更加擅长描述“如何更新状态”

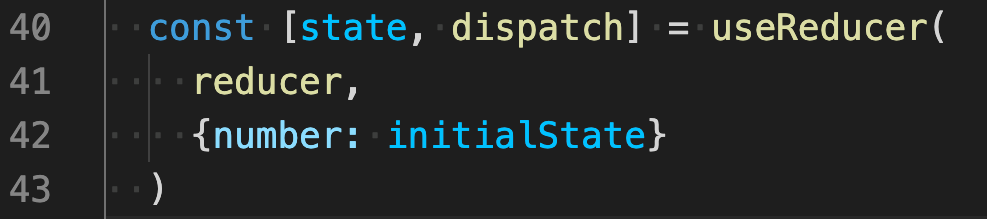
通过传递useReducer的dispatch，可以减少状态值的传递（useReducer总是返回相同的dispatch函数，这是彻底解耦的标志：状态更新逻辑可以任意变化，而发起actions的渠道始终不变）

3）用reducer实现之前的计数例子



state是一个number类型的数值，reducer根据action的类型（加/减）去修改对应的state，并返回最终的state。react 会确保 dispatch 函数的标识是稳定的，并且不会在组件重新渲染时改变。这就是为什么可以安全地从 useEffect 或 useCallback 的依赖列表中省略 dispatch

4）指定初始state的两种方法

A：将state作为第二个参数传入useReducer是比较简单的方法

注意

React 不使用 state = initialState 这一由 Redux 推广开来的参数约定。有时候初始值依赖于 props，因此需要在调用 Hook 时指定。如果你特别喜欢上述的参数约定，可以通过调用 useReducer(reducer, undefined, reducer) 来模拟 Redux 的行为，但我们不鼓励你这么做。

B：惰性初始化

需要将init函数作为useReducer的第三个参数传入，这样初始state将被设置为init(initialArg)---这么做可以将用于计算 state 的逻辑提取到 reducer 外部,为将来对重置state和action做处理提供了便利



5）跳过dispatch

如果 Reducer Hook 的返回值与当前 state 相同，React 将跳过子组件的渲染及副作用的执行。（React使用 Object.is 比较算法 来比较 state。）

注意：React 可能仍需要在跳过渲染前再次渲染该组件。不过由于 React 不会对组件树的“深层”节点进行不必要的渲染，所以大可不必担心。如果你在渲染期间执行了高开销的计算，则可以使用 useMemo 来进行优化。

6、**useEffect（）**就是一个Effect Hook,给函数增加了操作副作用的能力，常用于向服务器请求数据（可以理解为class中componentDidMount,componentDidUpdate和componentwillUnmount三个函数的组合）



1)不需要清除的effect-react更新DOM之后做一些例如：发送请求，修改DOM等

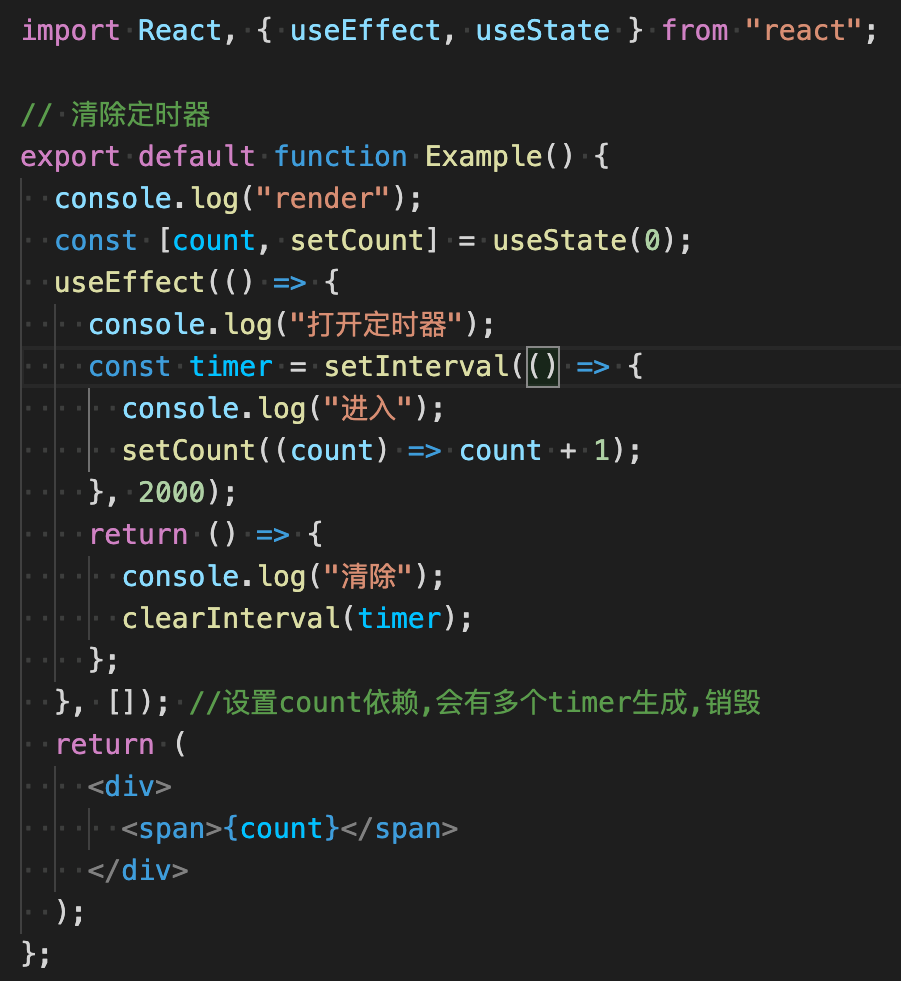
和上面相同案例class写法



在class组件中react操作dom之后，立即更新了document的title,这个时候我们希望组件在加载和更新的时候做相同的操作，所以我们要在componentDidMount和componentDidUpdate中些重复的代码，做同样的操作

但是在函数组件中当我们调用 useEffect 时，就是在告诉 react 在完成对 DOM 的更改后运行你的“副作用”函数。由于副作用函数是在组件内部声明的，所以它们可以访问到组件的 props 和 state。默认情况下，react 会在每次渲染后调用副作用函数 —— **包括**第一次渲染的时候，不需要像class组件那样去考虑挂载还是更新

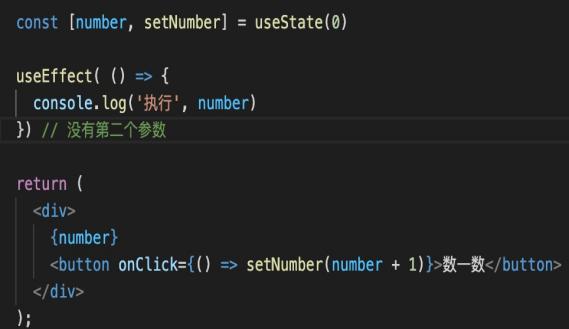
2）需要清除的effect-像定时器，订阅外部数据源等，防止内存泄漏



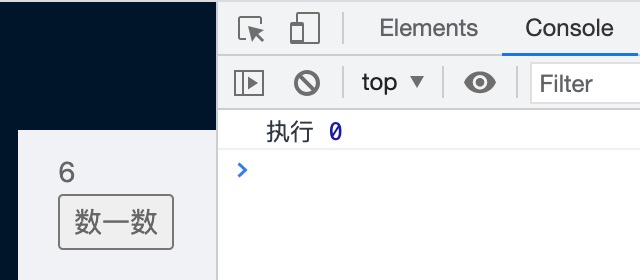
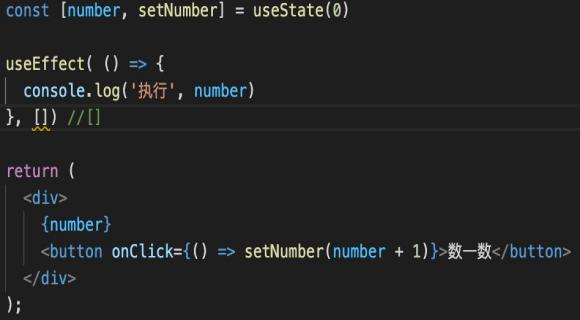
3)useEffect接受两个参数（函数（异步操作），数组（依赖项）），useEffect会在组件每次render之后调用，可以通过传参来决定是否调用

4)第二个参数传参的四种情况

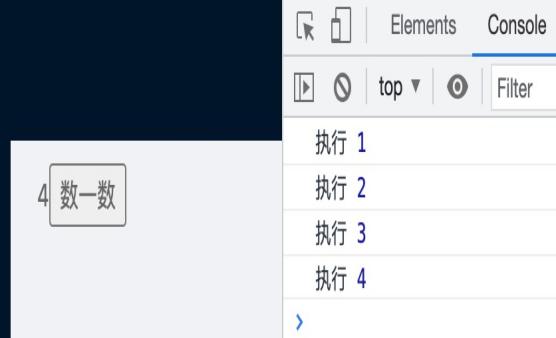
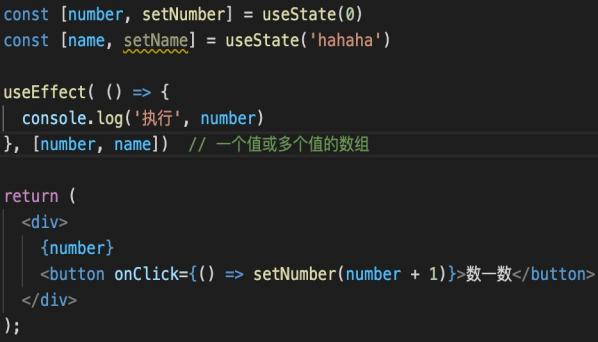
A：不传 每次渲染之后都会运行useEffect,获取数据并更新状态



B：空数组 仅初始化调用一次

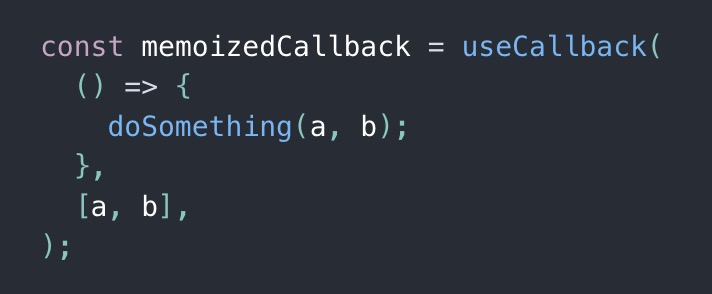


C：一个值或者多个值的数组 只有一个值变化即执行，多个值的数组有一个变化即变化



7、useCallback（）-用于得到一个固定引用值的函数

useCallback有两个参数（函数， 数组），返回一个 [memoized](https://en.wikipedia.org/wiki/Memoization" \t "/Users/zhangyuyan/Documents\\x/_blank) 回调函数

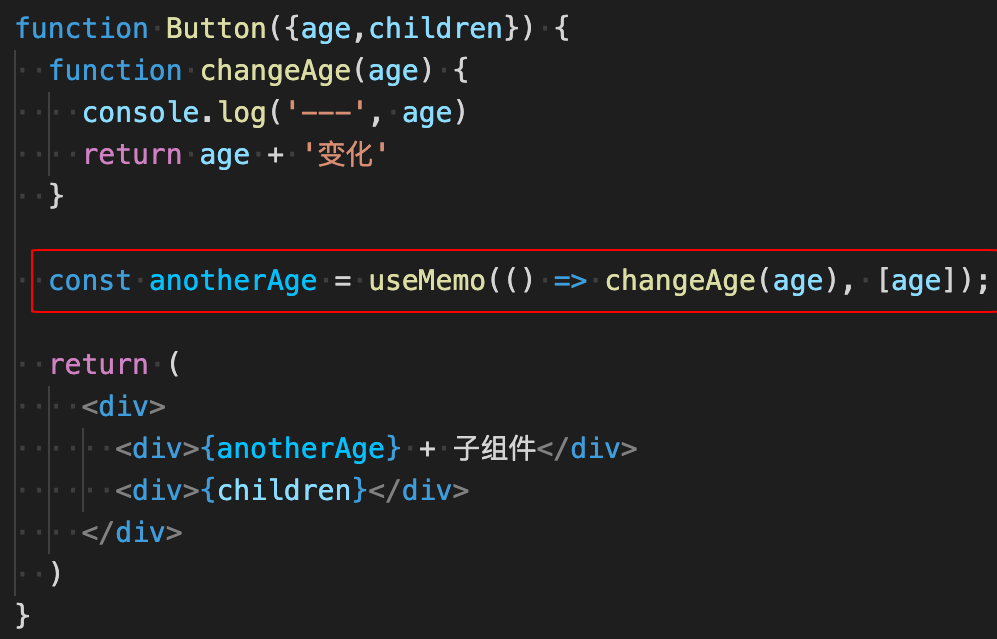


8、useMemo--主要用来解决hook产生的无用渲染的性能问题（一个函数是否需要多次执行）

const memoizedvalue = useMemo(() => computeExpensiveValue(a,b), [a,b]),返回一个memoized值



这个例子我们在点击父组件任意一个按钮的时候，子组件都会发生变化（可以在changename里打印查看）我只点击address按钮，只想修改他自己，age没有变化，但是控制台打印出执行了changeAge方法。 这意味着性能的损耗，下面用useMemo来优化



9、useRef()

const refContainer = useRef(initialValue);

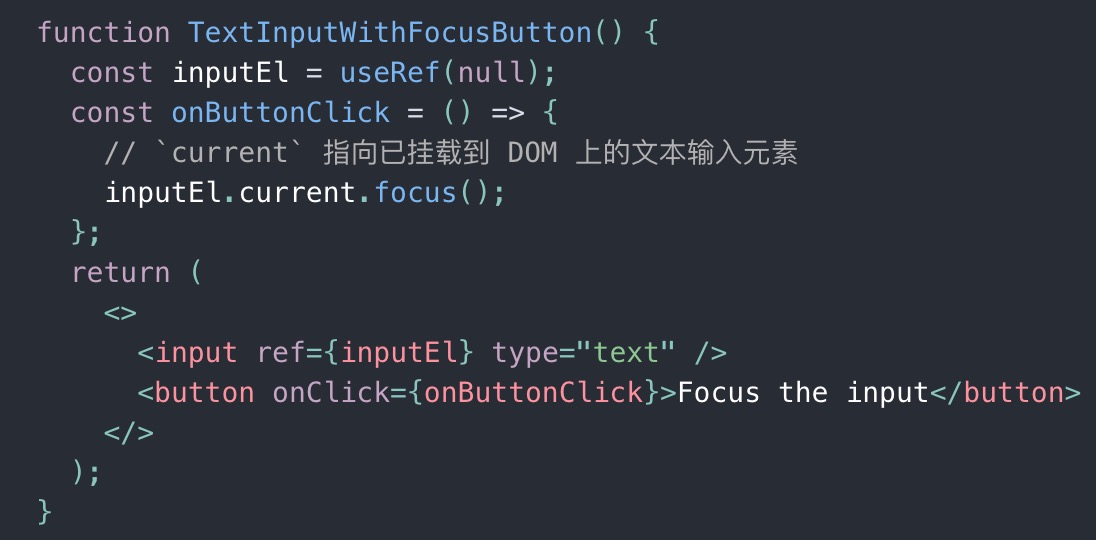
返回一个可变的 ref 对象，该对象只有一个 current 属性，初始值为传入的参数( initialValue )。

返回的 ref 对象在组件的整个生命周期内保持不变

当更新 current 值时并不会重新render ，这是与 useState 不同的地方

更新 useRef 是 side effect (副作用)，所以一般写在 useEffect 或 event handler 里

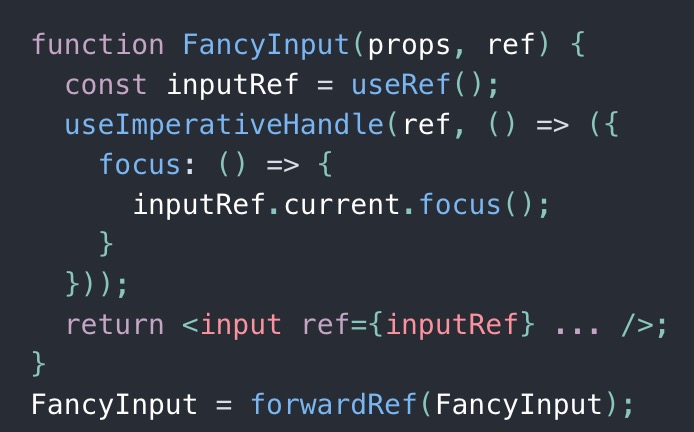
useRef 类似于类组件的 this



### 10、useImperativeHandle()

useImperativeHandle（ref, createHandle, [deps]）

useImperativeHandle 可以让你在使用 ref 时自定义暴露给父组件的实例值。在大多数情况下，应当避免使用 ref 这样的命令式代码。useImperativeHandle 应当与 [forwardRef](https://zh-hans.reactjs.org/docs/react-api.html" \l "reactforwardref) 一起使用



### 11、useLayoutEffect()

### 其函数签名与 useEffect 相同，但它会在所有的 DOM 变更之后同步调用 effect。可以使用它来读取 DOM 布局并同步触发重渲染。在浏览器执行绘制之前，useLayoutEffect 内部的更新计划将被同步刷新。

尽可能使用标准的 useEffect 以避免阻塞视觉更新。

12、hook使用规则

只能在**函数最外层**调用 Hook。不要在循环、条件判断或者子函数中调 用。

只能在 **React 的函数组件**中调用 Hook。不要在其他 JavaScript 函数中调用。（还 有一个地方可以调用 Hook —— 就是自定义的 Hook 中，我们稍后会学习到。