# **useState**

具有类似this.state功能，让静态组件具备state

参数一个是state名，一个是回调函数，useState可以设置初始值

const [count,setCount]=useState(0);

改变state，直接调用第二个方法即可

<button onClick={()=>setCount(count+1)}>+</button>

# **useEffect**

具有生命周期方法的功能，让静态组件具有生命周期方法

--componentDidMount

--componentDidUpdate

--componentWillUnmount

useEffect中的值，第一个是一个回调函数，可以再里面请求数据，写异步操作

第二个参数如果没有，任何state值改变都会调用，如果第二个参数是一个空数组，就相当于componentDidMount，只在页面初始化时候调用，如果数组中有值，就在监控的state值发生改变时调用userEffect

uerEffect配合async、await使用

使用useEffect时，外侧不要有循环，条件或嵌套函数

**react.Memo**

类组件中使用pureComponent可以进行浅层的比较，函数组件中可以使用react。Memo对子组件进行包裹，使子组件在获得props属性的时候，可以进行前后的对比

# **useCallback**

对**匿名**函进行缓存---重新render，就不再重新创建匿名函数

# useMemo

调用函数对函数返回的私有函数进行缓存 --重新render，就不再重新创建私有函数

说明:useCallback useMemo 结合高级组件React.memo() 对静态组件进行优化

<Child name={name} setName={useCallback(()=>{setName("jarry")},[])}/> --匿名，不会调用

    <Child name={name} setName={useMemo(doSomeThing,[])}/> --有名，会调用

说明:useCallback缓存函数的引用，useMemo缓存计算数据的值。

唯一区别就是：

useCallback是根据依赖缓存第一个入参的(callback)。

useMemo是根据依赖缓存第一个入参(callback)调用后的值。

useMemo会调用一次，useCallback不会调用

# useReducer

是 useState 的另一种替代方案，也可以修改state，重新render

// --将初始state initialState赋值给state

// --注册reducer，但是不调用

// --解构出来一个dispatch函数

useReducer会接收reducer和一个初始state值，解构可以获得state值和dispatch，可以显示state值和通过dispatch发送action

 let [state,dispatch]=useReducer(reducer,初始state的值)

*// 与react 不同的*useReducer*会注册reducer，但是不会调用reducer，*

# useContext

声明使用全局容器数据， 可以实现组件跨层通信，全局共享数据context，类似于prop-types 提供context

--<1>.创建全局容器 const Context= React.createContext({defaultValue})

--<2>.向后代传递数据 <Context.Provider value={要传递的数据}></Contxt.Provider>

--<3>.声明使用全局容器数据获取state和dispatch

let {state,dispatch} = useContext(CountContext)

说明:useReducer+useContext实现 redux

# useRef

**使用场景**

1 .获取子组件的实例，只有类组件可以使用  
2 .在函数组件中定义一个全局变量，不会因为重复render重复声明，类似于组件的this. 一个参数  
3. 虽然可以通过useState来保留变量的值，因为useState会触发组件render,可以用useRef来实现组件不会render的问题

4 .**他创建的是一个普通的js对象，而useRef和自己创建一个对象的区别是，useRef会在每次渲染时返回对同一个ref对象**  
5 .当ref内容发生变化时，useRef是不会通知你的，变更属性也不会引发组件的重新渲染

# 自定义hook

可以自定义hook，返回state值，和一些事件方法，调用useState中的方法，修改state，

在组件中调用自定义hook，可以返回value值，通过value值调用自定义hook中的state和自定义hook中的方法

# 1. 自定义hooks怎样实现组件的复用，如何通过hook实现

使用 React 自定义 Hooks，我们可以**把请求数据、更新加载状态、更新错误状态的代码全部写到自定义的 Hooks 中，然后在需要它的组件中调用即可**，一般只需要一行代码。

React 自定义 Hooks可以让业务逻辑从展示组件中抽离出来，并可以多次复用，极大的减少了代码量，提高了效率。

**它的特点和注意事项：**

自定义 hook 以 use 开头

可以接受参数

可以返回任何类型的返回值

可以使用其他内置或自定义的 hooks

每次调用 hook，其中的状态和逻辑都是隔离的

# 2.为什么使用hooks，hooks优势

1.React Hooks不必写class组件就可以用state和其他的React特性；

2.你也可以编写自己的hooks在不同的组件之间复用；

hooks优势：

**2.更容易复用代码:**它通过自定义hooks来复用状态，从而解决了类组件有些时候难以复用逻辑的问题

**3.函数式编程风格**:函数式组件、状态保存在运行环境、每个功能都包裹在函数中，整体风格更清爽，更优雅

**4.代码量更少**

**5.更容易拆分组件**

**1.没有破坏性改动**

完全可选的。 你无需重写任何已有代码就可以在一些组件中尝试 Hook。

100% 向后兼容的。 Hook 不包含任何破坏性改动。

# 3.如果子组件是函数组件不存在生命周期和钩子怎么进行性能优化？

（1）hooks中的usermemo （2） hooks中的useCallback

# 3-1.hooks优化,useCallback功能

用React.memo进行包裹

父组件向子组件重新render的话 子组件也会进行重新render 即使传的值每次都是一样的 子组件也会重新进行render

但是向子组件传函数的话 还是每次都渲染子组件 可以用usecallback进行优化

用usecallback进行包裹

# 4.useCallback和useMemo关系和区别?

**相同点：**

useCallback 和 useMemo 参数相同，第一个参数是函数，第二个参数是依赖项的数组。

useMemo、useCallback 都是使参数（函数）不会因为其他不相关的参数变化而重新渲染。

与 useEffect 类似，[] 内可以放入你改变数值就重新渲染参数（函数）的对象。如果 [] 为空就是只渲染一次，之后都不会渲染。

**不同点，区别：**

主要区别是 React.useMemo 将调用函数并返回其结果，

而React.useCallback 将返回函数而不调用它

# 5.useEffect的执行是怎样的？

useEffect 具有生命周期方法的功能，让静态组件具有生命周期方法

--componentDidMount

--componentDidUpdate

--componentWillUnMount

useEffetc(()=>{}) 任何状态变量改变 componentDidMount/componentDidUpdate都会执行

useEffect(()=>{},[状态变量]) 仅这个状态变量改变 componentDidMount/componentDidUpdate都会执行

useEffect(()=>{},[]) 仅调用componentDidMount，不会调用componentDidUpdate

userEffect(()=>{return 语句体}) 当移除组件时，回执行return后的语句