**React钩子函数应用探究**

1. **案例描述**

React 16.8.0版本添加了Hooks功能，新版本React有两套API，一套是基于原来的类(class)的API，另一套是基于函数的Hook API。因为函数式组件更加轻量级且符合React的函数式思想。希望对React常用的Hooks函数做一些总结，并在未来的开发中推广。

1. **案例分析与解决**

函数钩子的灵活性比较大，而类有很多强制的语法约束。本文希望通过对比类组件和函数组件的实现方法，以及具体分析函数组件的钩子函数的应用，说明钩子函数的使用方法以及在项目中的具体应用。

1. **React Hooks方法探究**
   1. **类组件与函数式组件比较**

类与函数两种实现组件的方式，代表了不同的编程方法论。类（class）是数据和逻辑的封装。 也就是说，组件的状态和操作方法是封装在一起的。如果选择了类的写法，就应该把相关的数据和操作，都写在同一个 class 里面。函数一般来说，只应该做一件事，就是返回一个值。 如果你有多个操作，每个操作应该写成一个单独的函数。而且，数据的状态应该与操作方法分离。根据这种理念，React 的函数组件只应该做一件事情：返回组件的JSX代码，而没有其他的功能。React 16.8.0在还没有引入Hooks之前，使用两种不同类型组件的对比如下。

class component：

1. 在组件结构比较复杂的时候，比较难拆分，难拆分导致了在组件的基础上做其他事情也比较麻烦。
2. 组件之间数据的传递使用props，在组件层次比较深的情况下要层层传递较麻烦。

function component：

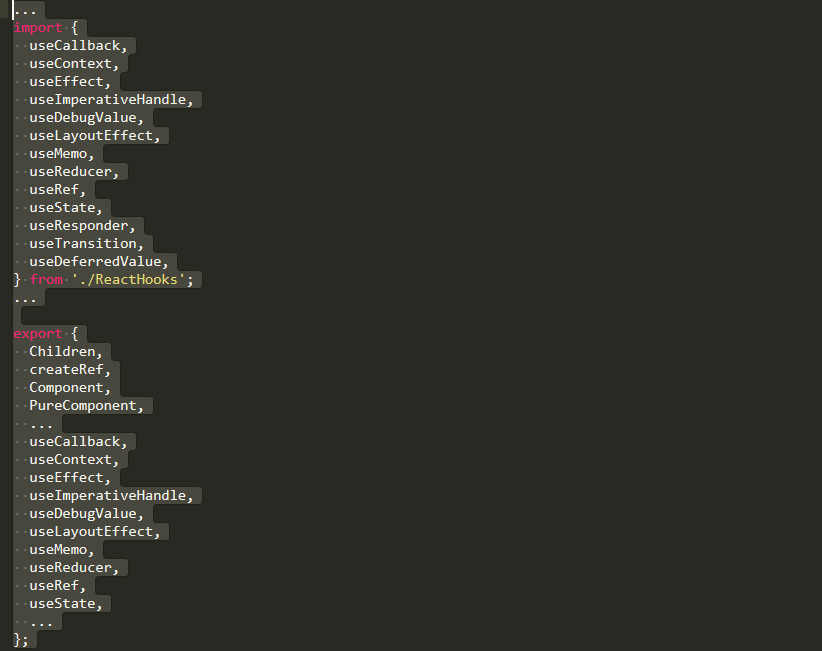
1. 函数式组件只做渲染，实现代码相比类组件更少，但是在只需要做展示的时候，函数式组件更加友好。
2. 因为是无状态组件，不能使用state或者其他的生命周期函数。

由于函数式组件主要是用于数据计算，并输出JSX，由于函数式组件没有状态和生命周期函数，那么和数据计算无关的一些操作应该怎么用函数式组件实现呢，函数式编程将与数据计算无关的操作，都称为 "副效应" （side effect） 。如果函数内部直接包含产生副效应的操作，就不再是纯函数了，纯函数内部只有通过间接的手段（即通过其他函数调用），才能包含副效应。而钩子（hook）就是 React 在16.8.0版本之后提供的函数组件的副效应解决方案，用来为函数组件引入副效应。

* 1. **Hook的功能**

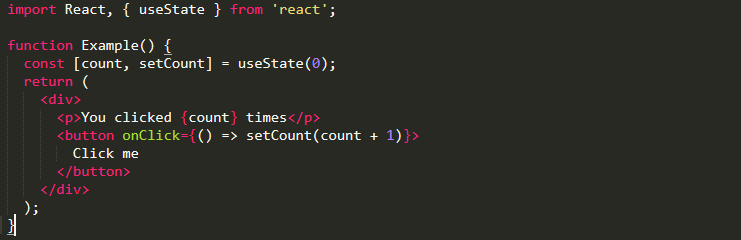
钩子（hook）就是 React 函数组件的副效应解决方案，用来为函数组件引入副效应。函数组件的主体只应该用来返回组件的JSX 代码，所有的其他操作（副效应）都必须通过钩子引入。

React Hook 在 Function Component 上开了一些 Hook，通过内置的一些 Hook 可以让 Function Component 拥有自己的状态（state）和生命周期，通过React的源码，我们可以找到所有react hooks的定义，如下图所示（部分代码已省略）。



本文将结合实际例子，对最常用的useState，useEffect, useMemo, useCallback几种Hook函数进行说明。

* 1. **State Hook**

****

useState是State Hook的 API。传入参数是 initialState，返回一个 数组，第一个值是 state，第二个值是改变state的函数。如果 initialState 的提高需要消耗大量的计算力，同时不期望这些计算阻塞后面要干的事情的话。initialState可以是个函数，会在render前调用达到 Lazy Calculation的效果。

* 1. **Effect Hook**

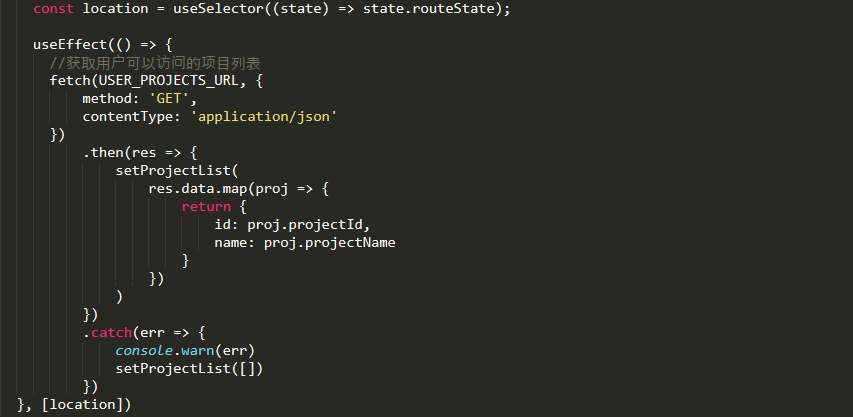
useEffect相当于Class Component中componentDidMount，componentDidUpdate, componentWillUnmount 三个生命周期的综合，在组件挂载、更新、卸载的时候都会执行 effect 里面的函数。用 “after render” 理解是最好的。值的注意的是，Effect Hook 中的内容不会像 componentDidMount、componentDidUpdate 一样阻塞渲染。下图是一个用useEffect来获取数据的例子。

****

在一个 Function Component 里，和 useState 一样可以可以使用多次 useEffect，这样在组织业务逻辑的时候，就可以按照业务逻辑去划分代码片段了（而不是 Class Component 中只能按照生命周期去划分代码片段）。

Effect Hook 为了避免每个render都执行所有的 Effect Hook，useEffect 提供了第二个入参（是个数组），组件 rerender 后数组中的值发生了变化后才会执行该 Effect Hook，如上图例子所示，由于获取数据只需要执行一次，所以上例的useEffect的第二个参数为一个空数组。值得注意的是，如果useEffect函数第二个参数不传，useEffect会在每次渲染后执行，而不是只执行一次。

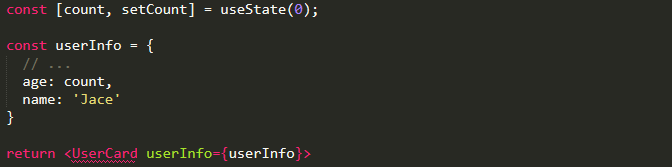
另一种情况是，并不仅仅是在组件挂载时需要触发，但也不需要在每次render的时候都触发useEffect影响性能。而是要根据组件某一个或几个状态是否改变，来决定是否触发次useEffect函数，下图代码就是一个典型的例子。

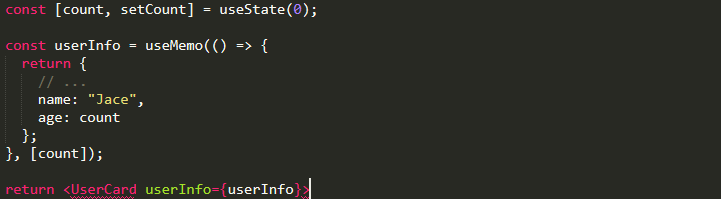


如上图代码所示，只有在location这个state的状态变化时，会触发这个useEffect以重新获取用户可以访问的项目列表，其他情况下不会触发此useEffect。

* 1. **Memo Hook**

useMemo这个hook，传递一个创建函数和依赖项，创建函数会需要返回一个值，只有在依赖项发生改变的时候，才会重新调用此函数，返回一个新的值。它的作用可以通过下图两块代码的对比体现。



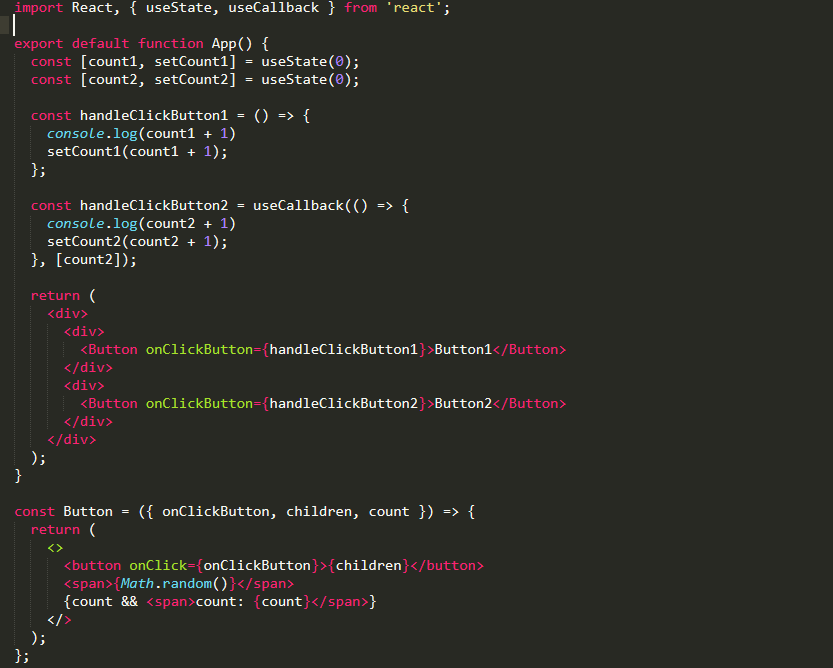


第一张图的代码 userInfo 每次都将是一个新的对象，无论 count 发生改变与否，都会导致 UserCard 重新渲染，而下面的代码则会在 count 改变后才会返回新的对象。因为传入参数的第二项数组中，只有count一个依赖。根据这个特性，我们可以把一些昂贵的计算逻辑放到 useMemo 中，只有当依赖值发生改变的时候才去更新。

* 1. **Callback Hook**

useCallback 和 useMemo 的参数跟 useEffect 一致，他们之间最大的区别有是 useEffect 会用于处理副作用，而前两个hooks不能。useCallback不会自动执行任何操作，它返回一个可以由任何需要触发它的代码执行的函数，这一点与useEffect不同。useCallback会在组件第一次渲染的时候执行，之后会在其依赖的变量发生改变时再次执行，返回一个函数。

如下图所示，是对点击事件在用useCallback和不不用useCallback两种方式对比。

****

在案例中可以点击 Button1 和 Button2 两个按钮来查看效果，点击 Button1 的时候只会更新 Button1 后面的内容，点击 Button2 会将两个按钮后的内容都更新。这就表示在点击 Button2 的时候导致了两个按钮都重新渲染了。而点击Button2由于只依赖count2，所以只有在点击Button2时候渲染，点击Button1时不渲染。

1. **经验总结，预防措施核对规范的建议等**

本文主要结合具体示例介绍了几种常见React钩子函数的用法，后续会在项目中更多应用文中介绍的常用的React Hooks来实现组件，并会进一步研究其他React Hooks的用法。

1. **从本文可导出的检查项（checklist）**

无