class组件业务逻辑分散在组件的各个方法之中，重复逻辑

组件类引入了复杂的编程模式，例render props/高阶组件

大型组件难拆分，重构，测试。

1、代码可读性更强，方便阅读维护。例如，每个生命周期中常常会包含一些不相关的逻辑。一般我们都会在 componentDidMount 和 componentDidUpdate 中获取数据。但是，同一个 componentDidMount 中可能也包含很多其它的逻辑，如设置事件监听，而之后需在 componentWillUnmount 中清除。相互关联且需要对照修改的代码被进行了拆分，而完全不相关的代码却在同一个方法中组合在一起。如此很容易产生 bug，并且导致逻辑不一致。React Hooks 可以将功能代码聚合

2、组件树层级变浅。在原本的代码中，我们经常使用 HOC/render props 等方式来复用组件的状态，增强功能等，无疑增加了组件树层数及渲染，在 React DevTools 中观察过 React 应用，你会发现由 providers，consumers，高阶组件，render props 等其他抽象层组成的组件会形成“嵌套地狱”。而在 React Hooks 中，这些功能都可以通过强大的自定义的 Hooks 来实现。

3、不用再去考虑 this 的指向问题。在类组件中，你必须去理解 JavaScript 中 this 的工作方式。

**fiber 节点**保留了组件的 state， props, 和潜在的需要渲染到的目标 DOM 元素。因为 **fiber 节点**是可以改变的，React 不需要每次更新都为每个**节点**重新创建一遍 - 它可以简单地克隆，当有更新的时候更新这个**节点**

不需要清除的副作用： 记录渲染次数

需要清除的：定时器，获取数据

解绑函数return：解绑方法，写解绑时需要做的事情