PureComponent和React.memo

- **PureComponent:**
  PureComponent实现了组件内部的shouldComponentUpdate生命周期方法，在该方法内部，对state和props分别进行**浅比较**，当state和props都不变时，shouldComponentUpdate方法返回false，组件不重渲染
- **React.memo():**
  ReactMemo是React16.6新增的Api，它是一个高阶组件，同样在内部实现了对传入的props进行浅比较，如果props不变则组件不会重渲染，用React.memo包裹函数式组件，效果相当于类组件继承PureComponent

useCallback和useMemo：

useCallback和useMemo是React提供的两个hook，用于保留对象的引用

- **useCallback：(Function, Array) => Function**
  useCallback用于保留函数引用

```
const MyComponent = (props) => {
    const print = React.useCallback(() => {
        console.log('abc');
    }, []);

    return <button onClick={print}>打印abc</button>
}
```

  该hook接收两个参数，第一个参数是要保留引用的函数f，第二个参数是依赖数组dependencies，当依赖数组内的元素没有变化时，useCallback的返回值总是**f**的同一个引用
- **useMemo：(Function, Array) => Object**
  useCallback用于保留对象的引用

```
const MyComponent = (props) => {
    const {a, b, c, d} = props
    const obj = React.useCallback(() => {
        return {
            a,
            b,
            c,
            d
        };
    }, [a, b, c, d]);

    return <AnotherComponent someObject={obj} />;
}
```

该hook接收两个参数，第一个参数是要return需要保留引用的对象的函数f，第二个参数是依赖数组dependencies，当依赖数组内的元素没有变化时，useMemo的返回值总是**f的返回值**的同一个引用

性能优化完整解决方案

```
const Parent = () => {
    const someFunc = React.useCallback(() => {
        console.log('abc');
    }, []);

    const someObj = React.useMemo(() => {
        return {
            a: 'a',
            b: 'b'
        };
    }, []);

    return <Child func={someFunc} obj={someObj} />;
}
```

如上示例，Child组件是一个pureComponent组件或被React.memo包裹的组件，因此，只有它的props中的每一项都变化时，Child才会重渲染，但是由于它的两个props func和obj都是引用类型，而在Parent组件中，如果不用useCallback和useMemo，每次渲染过程中产生的someFunc和someObj的引用都不一样，因此Child组件总是会重渲染。用了useCallback和useMemo后，可以保证每次渲染中，someFunc和someObj总是不变，从而Child组件不会重渲染