深入认识setState

setState，它对状态的改变，**可能**是异步的

> 如果改变状态的代码处于某个HTML元素的事件中，则其是异步的，否则是同步

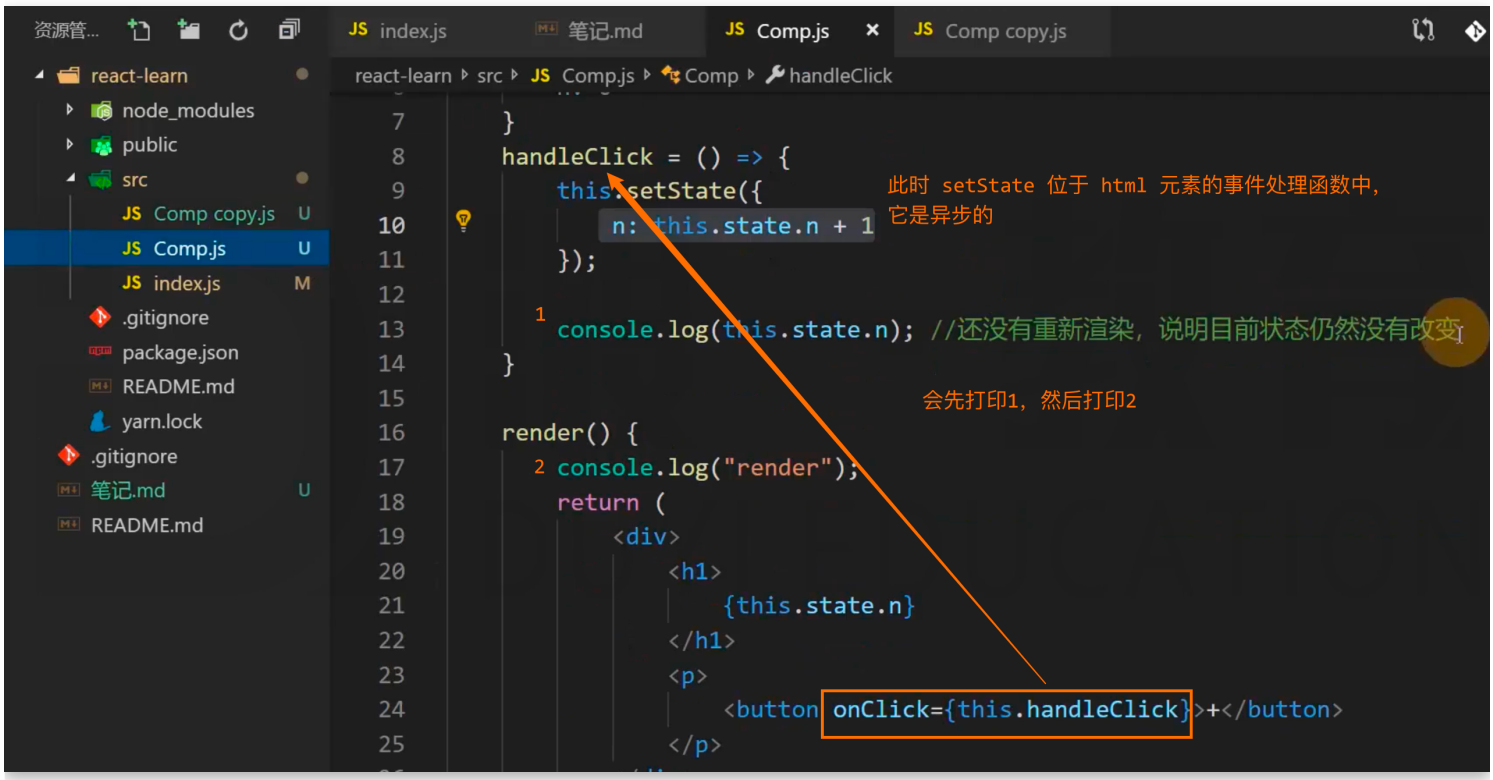
如果遇到某个事件中，需要同步调用多次，需要使用函数的方式得到最新状态

最佳实践：

1. 把所有的setState当作是异步的
2. 永远不要信任setState调用之后的状态
3. 如果要使用改变之后的状态，需要使用回调函数（setState的第二个参数）
4. 如果新的状态要根据之前的状态进行运算，使用函数的方式改变状态（setState的第一个函数）

React会对异步的setState进行优化，将多次setState进行合并（将多次状态改变完成后，再统一对state进行改变，然后触发render）

截图



```
handleClick = () => {
    this.setState({
        n: this.state.n + 1
    }, () => {
        //状态完成改变之后触发，该回调运行在render之后
        console.log(this.state.n);
    });
}
```

```
handleClick = () => {
    this.setState({
        n: this.state.n + 1
    });

    this.setState({
        n: this.state.n + 1
    });

    this.setState({
        n: this.state.n + 1
    });
}
```

当点击增加按钮后，
结果是 1，而不是 3

原因：setState
是异步的

1 执行完之后，状态并没有立刻
改变，在执行 2、3 的时候
state 还是之前的值

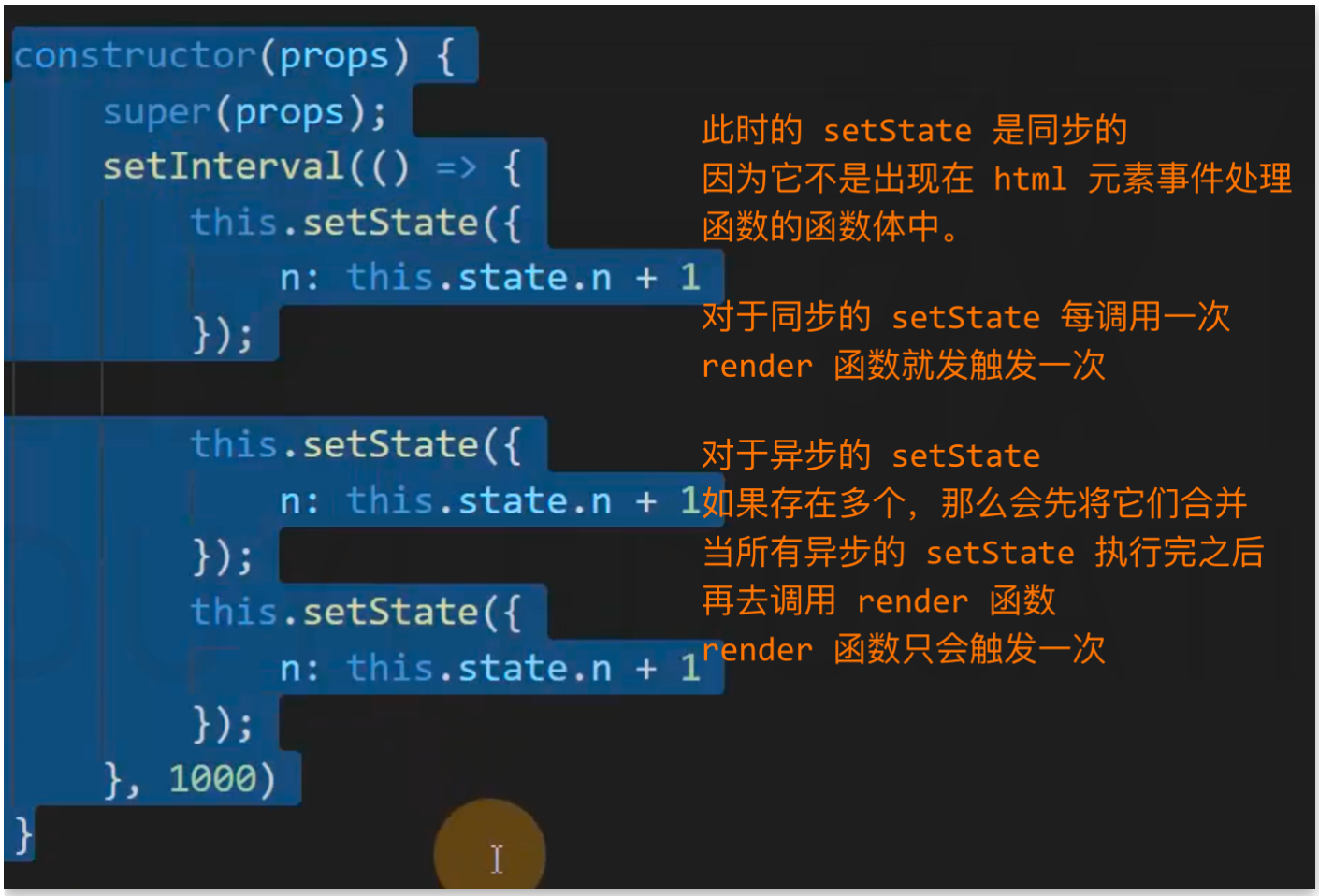
```
handleClick = () => {
    this.setState({
        n: this.state.n + 1
    }, () => {
        this.setState({
            n: this.state.n + 1
        }, ()=>{
            this.setState({
                n: this.state.n + 1
            });
        })
    });
}
```

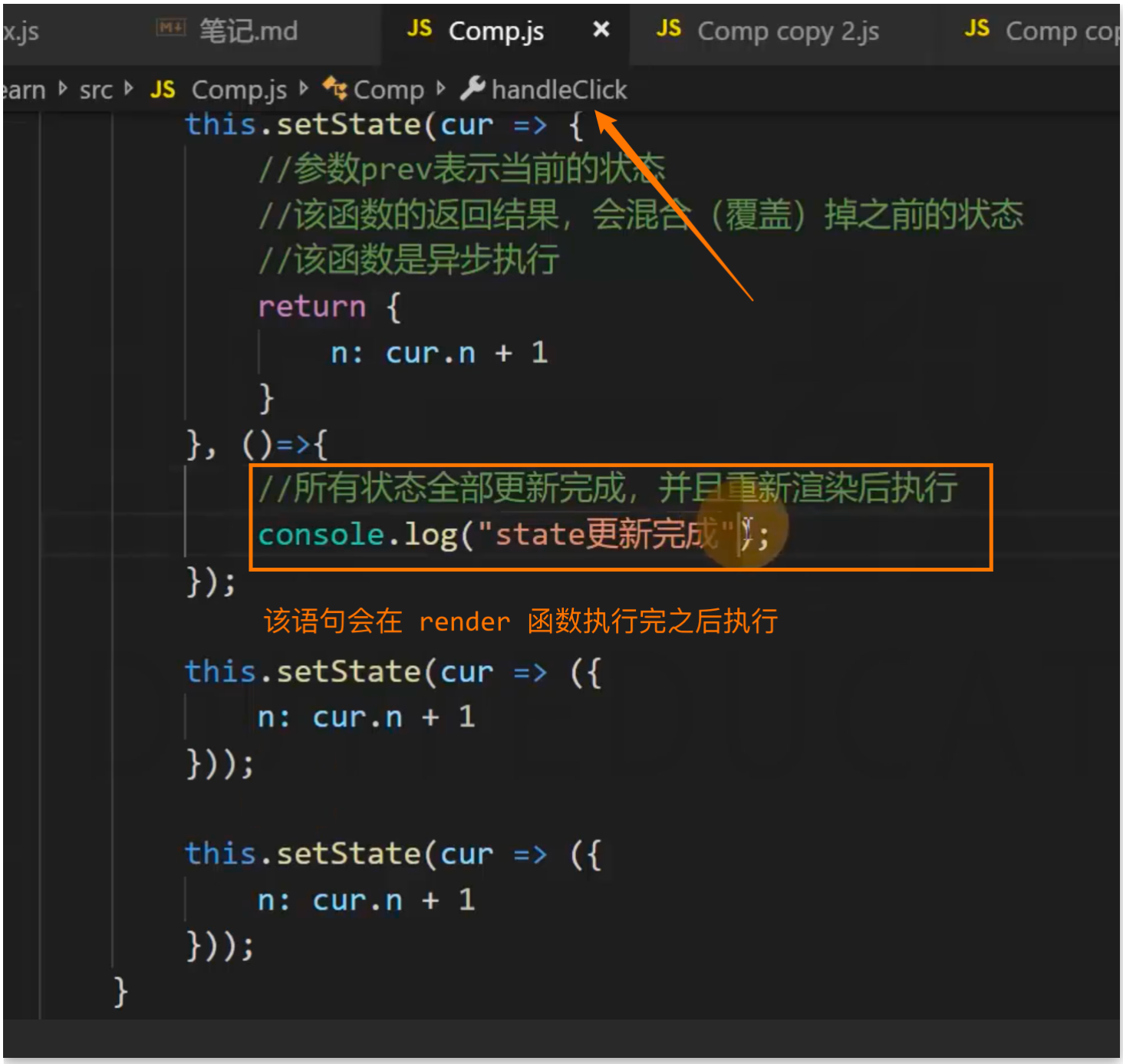
```
handleClick = () => {
    this.setState(cur => {
        //参数prev表示当前的状态
        //该函数的返回结果，会混合（覆盖）掉之前的状态
        //该函数是异步执行
        return {
            n: cur.n + 1
        }
    });

    this.setState(cur => ({
        n: cur.n + 1
    }));

    this.setState(cur => ({
        n: cur.n + 1
    }));
}
```

cur

setState 的第一个参数可以是一个回调函数
第一个参数：表示当前状态
这个状态是可以信任的
即便 setState 依旧是异步的

源码层面：这么写，传入的 3 个回调最终会依次丢到队列中，然后依次取出执行


constructor(props) {
super(props);
setInterval(() => {
this.setState({
n: this.state.n + 1
});
this.setState({
n: this.state.n + 1
});
this.setState({
n: this.state.n + 1
});
}, 1000)
}
此时的 setState 是同步的
因为它不是出现在 html 元素事件处理
函数的函数体中。
对于同步的 setState 每调用一次
render 函数就发触发一次
对于异步的 setState
如果存在多个，那么会先将它们合并
当所有异步的 setState 执行完之后
再去调用 render 函数
render 函数只会触发一次


x.js
笔记.md
JS Comp.js
JS Comp copy 2.js
JS Comp co
earn ▸ src ▸ JS Comp.js ▸ Comp ▸ handleClick
this.setState(cur => {
//参数prev表示当前的状态
//该函数的返回结果，会混合（覆盖）掉之前的状态
//该函数是异步执行
return {
n: cur.n + 1
}
}, ()=>{
//所有状态全部更新完成，并且重新渲染后执行
console.log("state更新完成");
});
该语句会在 render 函数执行完之后执行
this.setState(cur => ({
n: cur.n + 1
}));
this.setState(cur => ({
n: cur.n + 1
}));
}