

useState 原理

《React 造轮子》系列课程

版权声明

本内容版权属杭州饥人谷教育科技有限公司（简称饥人谷）所有。

任何媒体、网站或个人未经本网协议授权不得转载、链接、转贴，或以其他方式复制、发布和发表。

已获得饥人谷授权的媒体、网站或个人在使用时须注明「资料来源：饥人谷」。

对于违反者，饥人谷将依法追究法律责任。

联系方式

如果你想要购买本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

如果你发现有人盗用本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

useState 用法

```
5  function App() {  
6    const [n, setN] = React.useState(0);  
7    return (  
8      <div className="App">  
9        <p>{n}</p>  
10       <p>  
11         <button onClick={() => setN(n + 1)}>+1</button>  
12       </p>  
13     </div>  
14   );  
15 }  
16  
17 ReactDOM.render(<App />, rootElement);  
18
```

脑补一下点击 button 会发生什么

首次渲染 render <App/>

调用 App() 得到虚拟 DIV 创建真实 DIV

用户点击 button, 调用 setN(n+1), 再次 render <App/>

```
function App() {  
  const [n, setN] = React.useState(0);  
  return (  
    <div className="App">  
      ...  
    </div>  
  );  
}
```

ReactDOM.render(<App />, rootElement);

调用 App() 得到虚拟 DIV, DOM Diff, 更新真实 DIV

每次调用 App(), 都会运行 useState(0)

脑补之后

- 问自己几个问题

- ✓ 执行 `setN` 的时候会发生什么？`n` 会变吗？`App()` 会重新执行吗？
- ✓ 如果 `App()` 会重新执行，那么 `useState(0)` 的时候，`n` 每次的值会有不同吗？
- ✓ 通过 `console.log` 你就能得到答案！

分析

- **setN**

- ✓ setN 一定会修改数据 x ，将 $n+1$ 存入 x
- ✓ setN 一定会触发 `<App/>` 重新渲染(re-render)

- **useState**

- ✓ useState 肯定会从 x 读取 n 的最新值

- **X**

- ✓ 每个组件有自己的数据 x ，我们将其命名为 `state`

尝试实现 React.useState

```
5  function myUseState(initialValue) {
6    var state = initialValue;
7    function setState(newState) {
8      state = newState;
9      render();
10   }
11   return [state, setState];
12 }
13
14 // 教学需要, 不用在意 render 的实现
15 const render = () => ReactDOM.render(<App />, rootElement);
16
17 function App() {
18   const [n, setN] = myUseState(0);
19   return (
20     <div className="App">
21       <p>{n}</p>
22       <p>
23         <button onClick={() => setN(n + 1)}>+1</button>
24       </p>
25     </div>
26   );
27 }
28
29 ReactDOM.render(<App />, rootElement);
```

注：代码参考了[这里](#)

完全没有变化啊

因为 myUseState 会将 state 重置

我们需要一个不会被 myUseState 重置的变量

那么这个变量只要声明在 myUseState 外面即可

再尝试实现 React.useState

```
5  let _state
6
7  function myUseState(initialValue) {
8    _state = _state || initialValue;
9    function setState(newState) {
10     _state = newState;
11     render();
12   }
13   return [_state, setState];
14 }
15
16 // 教学需要, 不用在意 render 的实现
17 const render = () => ReactDOM.render(<App />, rootElement);
18
19 function App() {
```

点击图片[运行代码](#)

注：代码参考了[这里](#)

useState 就这么简单？

别急，还有问题

如果一个组件 用了两个 `useState` 怎么办

由于所有数据都放在 `_state`，所以会冲突

改进思路

- 把 `_state` 做成一个对象

- ✓ 比如 `_state = {n: 0, m: 0}`
- ✓ 不行，因为 `useState(0)` 并不知道变量叫 `n` 还是 `m`

- 把 `_state` 做成数组

- ✓ 比如 `_state = [0, 0]`
- ✓ 貌似可行，我们来试试看

多个 useState

```
5  let _state = [];  
6  let index = 0;  
7  
8  function myUseState(initialValue) {  
9    const currentIndex = index;  
10   index += 1;  
11   _state[currentIndex] = _state[currentIndex] || initialValue;  
12   const setState = newState => {  
13     _state[currentIndex] = newState;  
14     render();  
15   };  
16   return [_state[currentIndex], setState];  
17 }  
18  
19 // 教学需要, 不用在意 render 的实现  
20 const render = () => {  
21   index = 0;  
22   ReactDOM.render(<App />, rootElement);  
23 };  
24  
25 function App() {
```

运行看看，注意 index、currentIndex 和第 21 行

_state 数组方案缺点

- useState 调用顺序

- ✓ 若第一次渲染时 n 是第一个，m 是第二个，k 是第三个
- ✓ 则第二次渲染时必须保证顺序完全一致
- ✓ 所以React不允许出现如下代码

- 不允许出现的代码

The screenshot shows a code editor with a file named `index.js`. The code defines a function `App()` that uses `React.useState(0)` to initialize a state variable `n`. Inside the function, there is a conditional statement `if (n % 2 === 1) { [m, setM] = React.useState(0); }`. A red arrow points from the `useState(0)` call in the `if` block to a console error message. The console error message is an "Uncaught Invariant Violation: Rendered more hooks than during the previous render." with a stack trace pointing to the `useState` hook call.

```
22 ReactDOM.render(<App />, rootElement);
23 };
24
25 function App() {
26   const [n, setN] = React.useState(0);
27   let m, setM;
28   if (n % 2 === 1) {
29     [m, setM] = React.useState(0);
30   }
31   return (
32     <div className="App">
33       <p>{n}</p>
34       <p>
35         <button onClick={() => setN(n + 1)}>+1</button>
36       </p>
37     </div>
38   );
39 }
```

Console was cleared

Uncaught Invariant Violation: Rendered more hooks than during the previous render.

at invariant (https://j6y72.csb.app/node_modules/react-dom/cjs/react-dom.development.js:55:15)
at updateWorkInProgressHook (https://j6y72.csb.app/node_modules/react-dom/cjs/react-dom.development.js:13092:35)
at updateReducer (https://j6y72.csb.app/node_modules/react-dom/cjs/react-dom.development.js:13149:14)
at updateState (https://j6y72.csb.app/node_modules/react-dom/cjs/react-dom.development.js:13294:10)

现在的代码还有一个问题

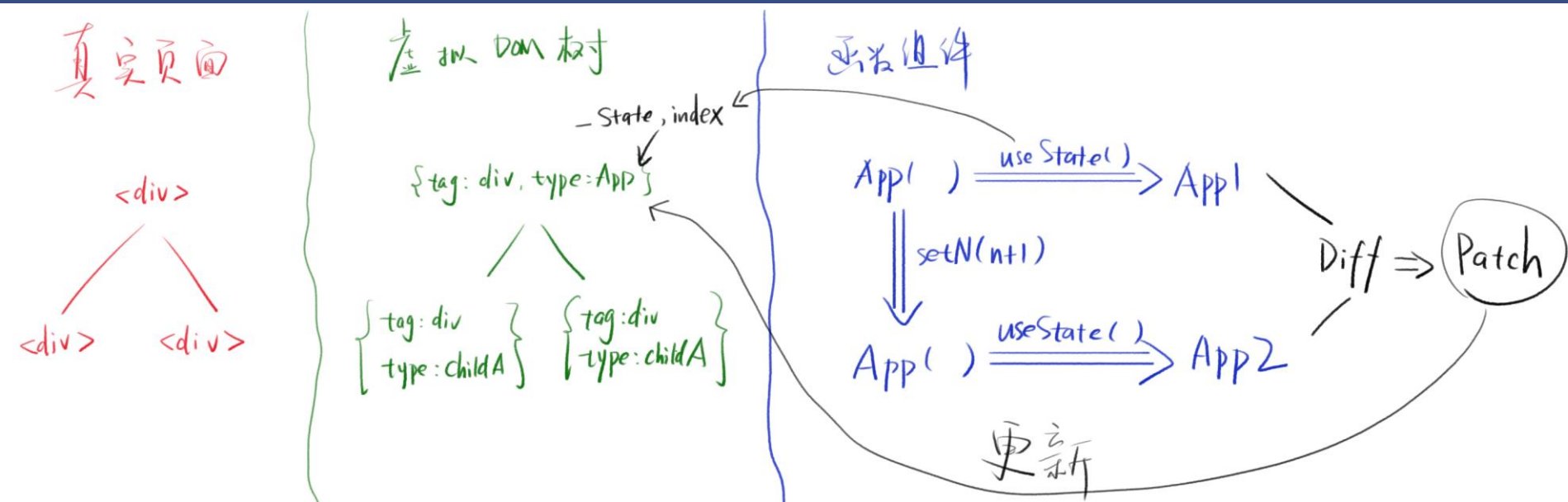
App 用了 `_state` 和 `index`，那其他组件用什么？

解决办法：给每个组件创建一个 `_state` 和 `index`

又有问题：放在全局作用域里重名了咋整？

解决办法：放在组件对应的虚拟节点对象上

图示



上图只画了 App 组件的更新过程，两个 ChildA 组件也有同样的过程

总结

- 每个函数组件对应一个 React 节点*
- 每个节点保存着 state 和 index
- useState 会读取 state[index]
- index 由 useState 出现的顺序决定
- setState 会修改 state，并触发更新
- 目前的内容其实就这么简单

注意：本节课对 React 的实现做了简化，React 节点应该是 FiberNode，_state 的真实名称为 memorizedState，index 的实现则用到了链表，有兴趣的同学可以[自行学习](#)。

n 的分身

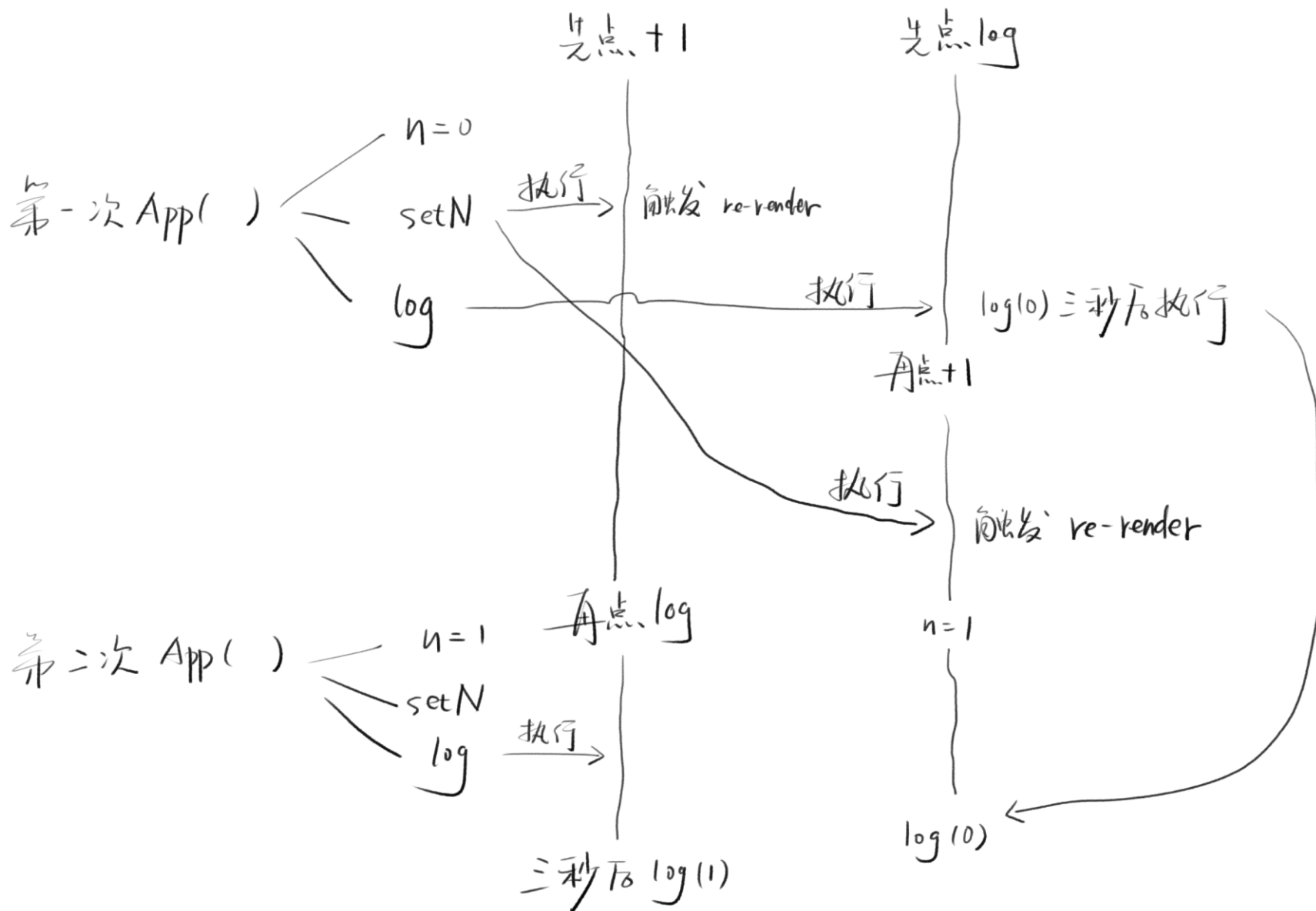
```
5 function App() {  
6   const [n, setN] = React.useState(0);  
7   const log = () => setTimeout(() => console.log(`n: ${n}`), 3000);  
8   return (  
9     <div className="App">  
10       <p>{n}</p>  
11       <p>  
12         <button onClick={() => setN(n + 1)}>+1</button>  
13         <button onClick={log}>log</button>  
14       </p>  
15     </div>  
16   );  
17 }
```

[点击运行](#)

• 两种操作

- ✓ 一、点击 +1 再点击 log —— 无 bug
- ✓ 二、点击 log 再点击 +1 —— 有 bug
- ✓ 问题：为什么 log 出了旧数据？

因为有多多个 n



如果我希望有一个
贯穿始终的状态，怎么做？

办法很多

贯穿始终的状态

- 全局变量

- ✓ 用 window.xxx 即可，但太 low 了

- useRef

- ✓ useRef 不仅可以用于 div，还能用于任意数据
- ✓ useRef 例子
- ✓ 强制更新的例子（不推荐使用）

- useContext

- ✓ useContext 不仅能贯穿始终，还能贯穿不同组件
- ✓ useContext 例子

总结

- 每次重新渲染，组件函数就会执行
- 对应的所有 state 都会出现「分身」
- 如果你不希望出现分身
- 可以用 useRef / useContext 等

再见

React 并不难，用逻辑就能搞懂