setState 的异步和合并

```
class App extends React.Component {
  constructor() {
    super();
    this.state = {val: 0};
  }
  componentDidMount() {
      // 传入函数，不会合并
      this.setState((prevState, props) => {
          console.log("a--", prevState.val + 1);
      });
      this.setState((prevState, props) => {
          console.log("b--", prevState.val + 1);
      });

    // this.setState({ val: this.state.val + 1 });
    // console.log("a--", this.state.val);  // a-- 0

    // this.setState({ val: this.state.val + 1 });
    // console.log("b--", this.state.val); // b-- 0

    // this.setState(
    //     { val: this.state.val + 1 },
    //     () => {
    //         console.log("bbb--", this.state.val); // bbb-- 1
    //     }
    // );

    setTimeout(() => {
      this.setState({ val: this.state.val + 1 });
      console.log("c--", this.state.val);  // c-- 2

      this.setState({ val: this.state.val + 1 });
      console.log("d--", this.state.val);  // d-- 3
    }, 0)

    // 自定义的 DOM 事件
    // document.getElementById("p1").addEventListener("click", () => {
    //   this.setState({ val: this.state.val + 1 });
    //   console.log("e--", this.state.val);
    // });
  }
  render() {
    return (
      <p id="p1">pargaraph</p>
    )
  }
}
ReactDOM.createRoot(document.getElementById("root")).render(<App />);
```

setState 默认是异步更新，默认会合并后更新。

setState 默认是异步的，前两条打印是同步代码，获取到的还是之前的初始值 0。

setState 设置第二个参数（回调函数），可以获取到更新后的值，但是 `bbb--` 获取到的还是 1，是因为 setState 是合并后更新。前面的几次 +1 会被本次 +1 合并。后者覆盖前者。

setState 也有同步更新的情况（不在 React 上下文中触发时）：

- 在 setTimeout, setInterval, promise.then。
- 在自定义的 DOM 事件中。
- ajax 回调中触发。

上述场景中， React 18 可以同步更新。

React 18 中需要将 `ReactDOM.render(<App />, document.getElementById("root"))` 替换为 `ReactDOM.createRoot(document.getElementById("root")).render(<App />)`;

setState 也有不合并的情况

- 比如同步代码中，都已经更新完了，就没法合并了。
- 传入函数的时候。函数没法合并，传入对象还可以和之前的对象合并。

传入函数时，不会合并，后面的 +1 会生效。

函数没法合并，对象还可以通过 `Object.assign({a: 1}, {b: 2})`，函数需要在执行的时候才知道里面内容。