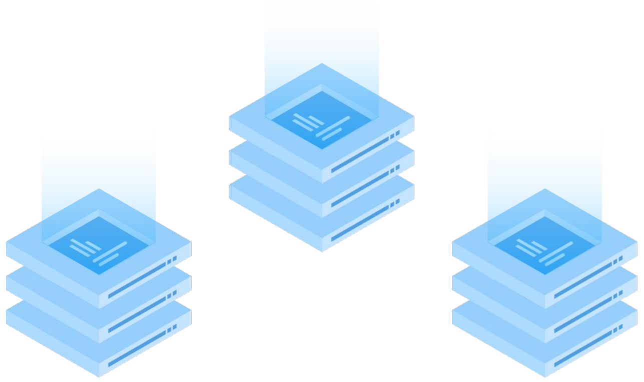
React Hooks 介绍

Presented by baurine@2020.08

PingCAP TiDB

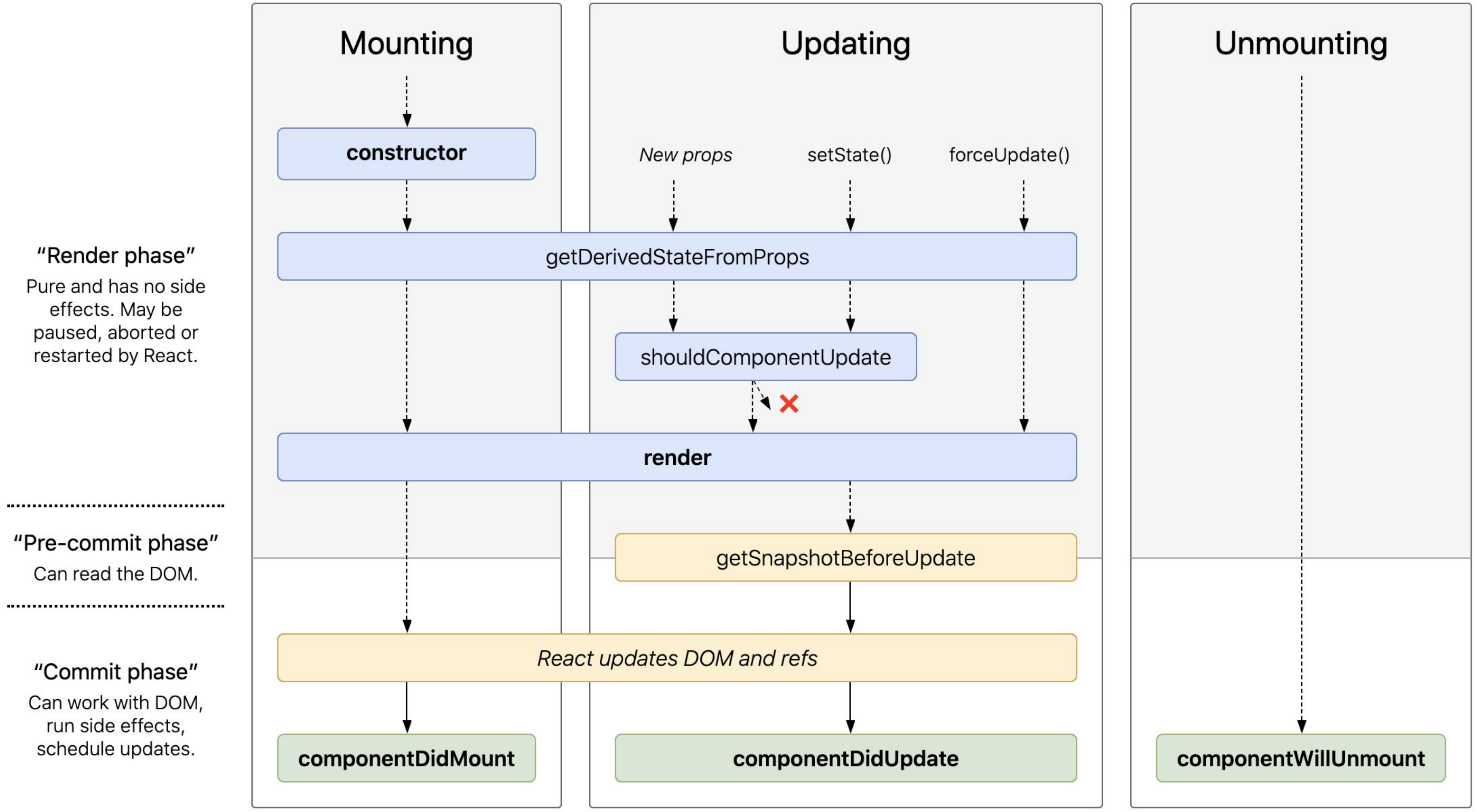


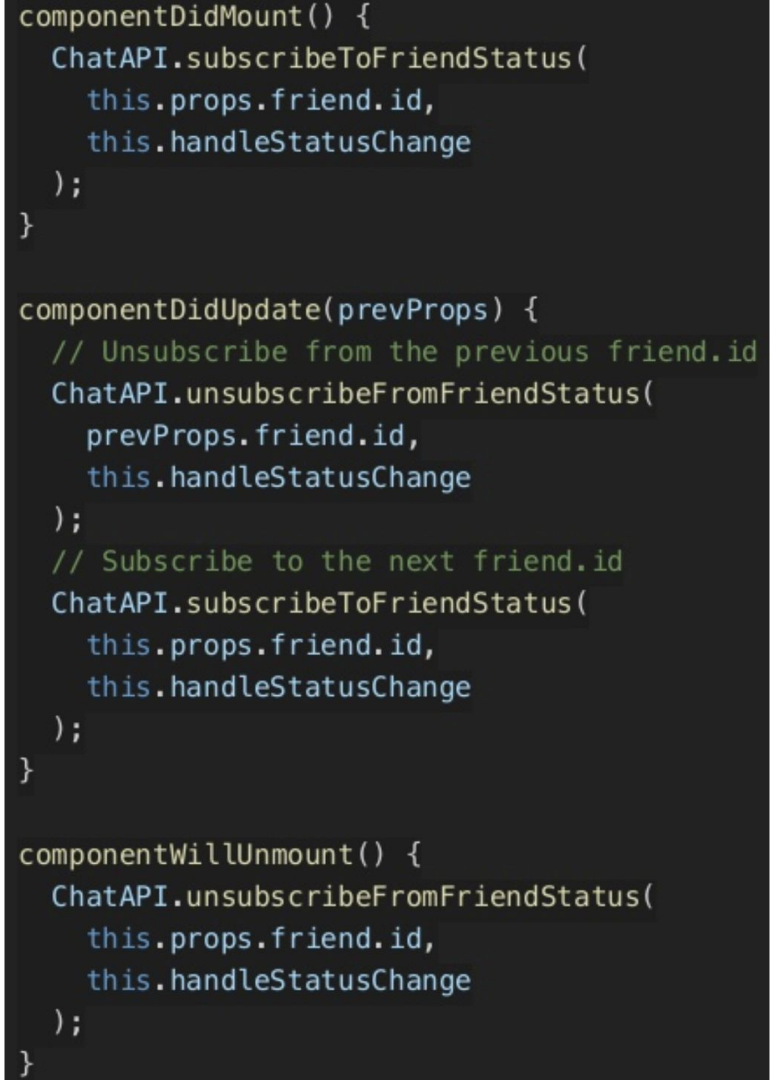
Hooks
React devs
Render Props
Higher Order Components

Part I - Why

Before hooks

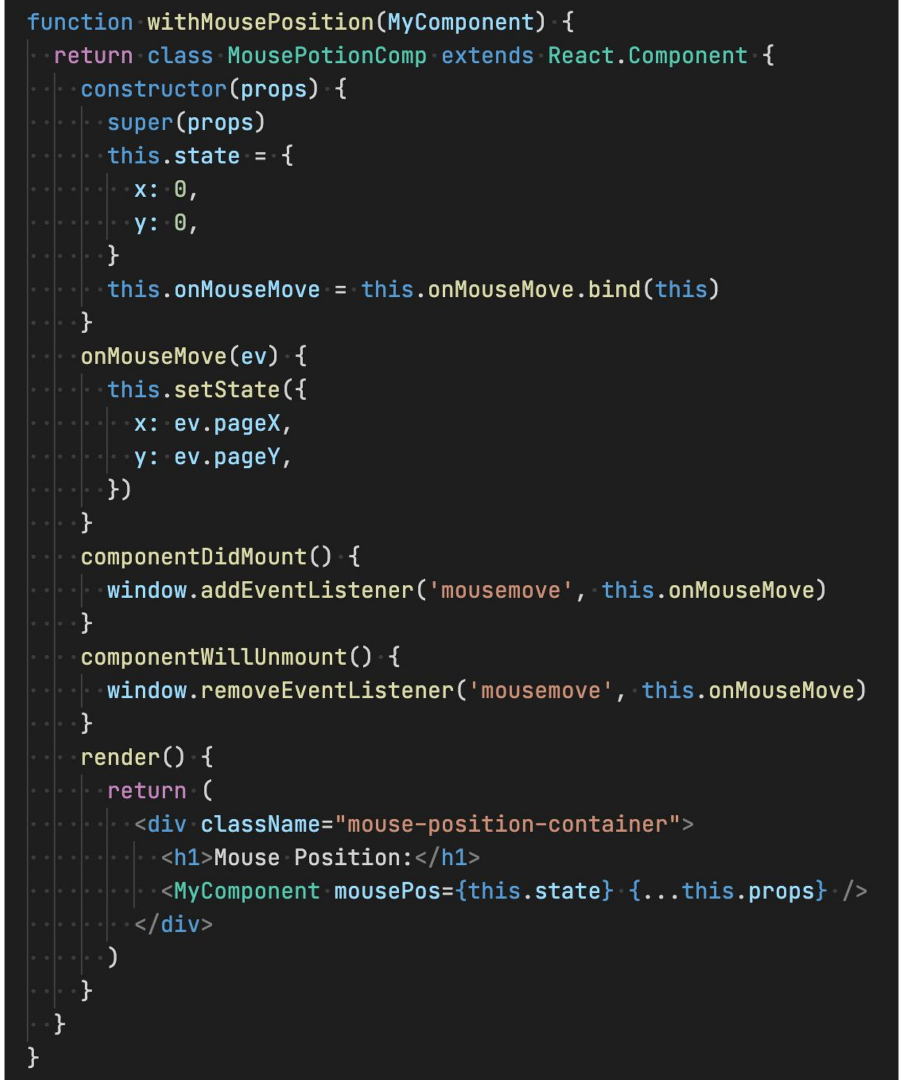
- 两种组件：Function (stateless) / Class，转换修改大
- class 组件：生命周期函数多 (头晕)
- class 组件：this 指针问题 (一堆 bind, 或是使用箭头函数)
- class 组件：经常要写重复代码 (componentDidMounted/componentDidUpdate)
- ...
- 最重要的是：组件之间复用状态逻辑很难 (被生命周期函数所割裂)
    - HOC (High Order Component)
    - render props


Mounting
Updating
Unmounting
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
constructor
New props
setState()
forceUpdate()
getDerivedStateFromProps
shouldComponentUpdate
render
"Pre-commit phase"
Can read the DOM.
getSnapshotBeforeUpdate
"Commit phase"
Can work with DOM, run side effects, schedule updates.
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount



```
componentDidMount() {
  ChatAPI.subscribeToFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}

componentDidUpdate(prevProps) {
  // Unsubscribe from the previous friend.id
  ChatAPI.unsubscribeFromFriendStatus(
    prevProps.friend.id,
    this.handleStatusChange
  );
  // Subscribe to the next friend.id
  ChatAPI.subscribeToFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}

componentWillUnmount() {
  ChatAPI.unsubscribeFromFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}
```

HOC

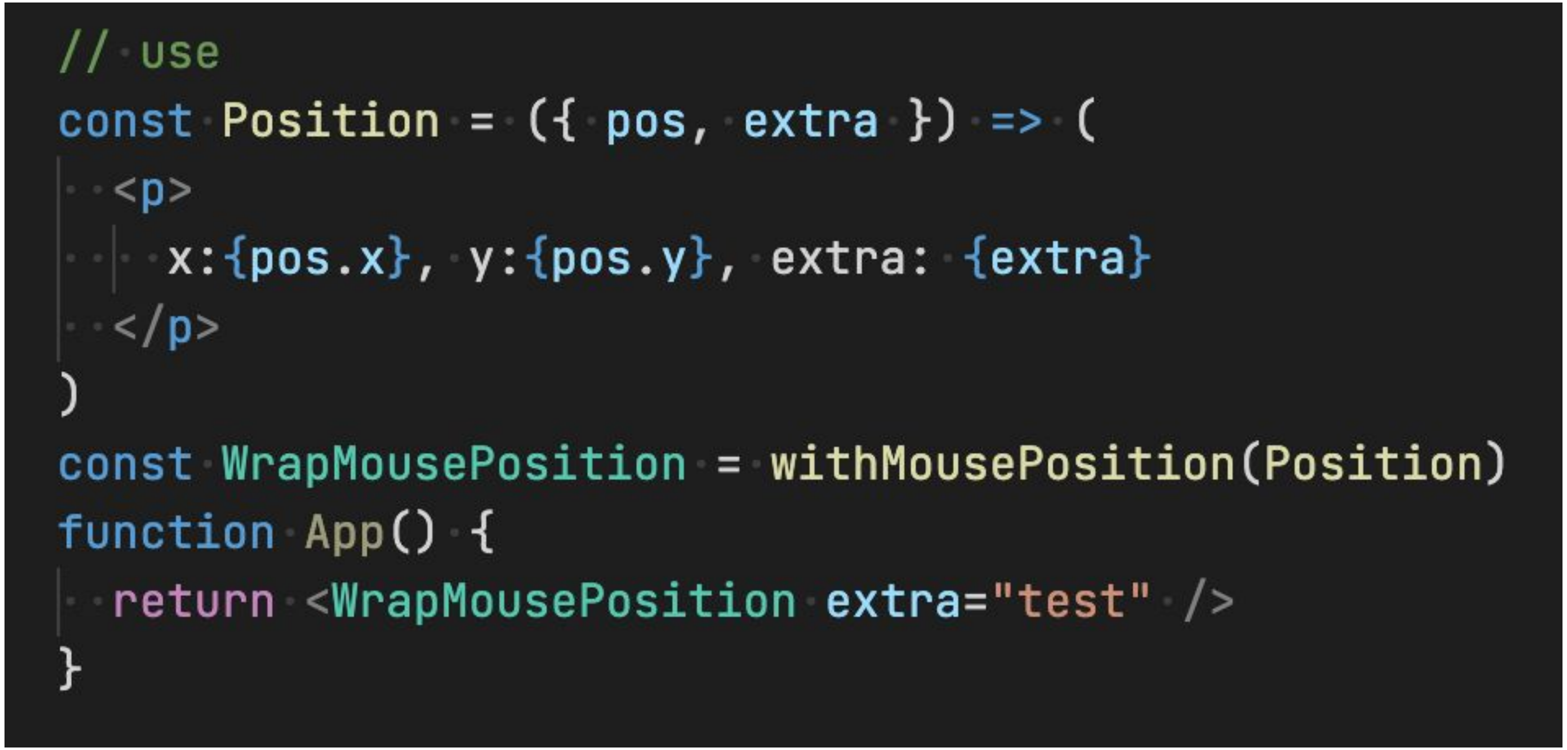


```
function withMousePosition(MyComponent) {
  return class MousePotionComp extends React.Component {
    constructor(props) {
      super(props)
      this.state = {
        x: 0,
        y: 0,
      }
      this.onMouseMove = this.onMouseMove.bind(this)
    }
    onMouseMove(ev) {
      this.setState({
        x: ev.pageX,
        y: ev.pageY,
      })
    }
    componentDidMount() {
      window.addEventListener('mousemove', this.onMouseMove)
    }
    componentWillUnmount() {
      window.removeEventListener('mousemove', this.onMouseMove)
    }
    render() {
      return (
        <div className="mouse-position-container">
          <h1>Mouse Position:</h1>
          <MyComponent mousePos={this.state} {...this.props} />
        </div>
      )
    }
  }
}
```

PingCAP

PingCAP.com

HOC



```
// use
const Position = ({ pos, extra }) => (
  <p>
    x:{pos.x}, y:{pos.y}, extra: {extra}
  </p>
)
const WrapMousePosition = withMousePosition(Position)
function App() {
  return <WrapMousePosition extra="test" />
}
```

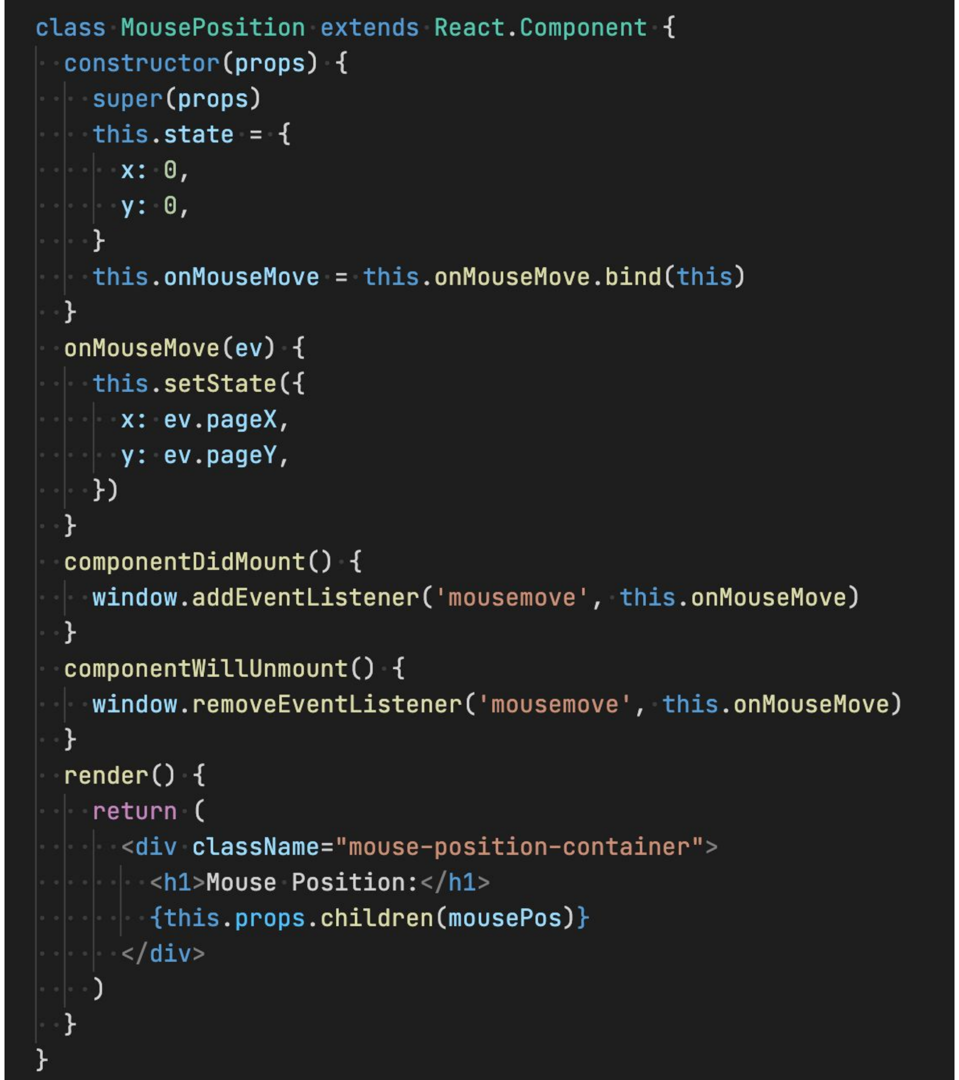
HOC

如果再来一个 withWindowSize() 的高阶组件函数，则代码如下所示：

```
const WrapComponent = withWindowSize(withMousePosition(MyComponent))
```

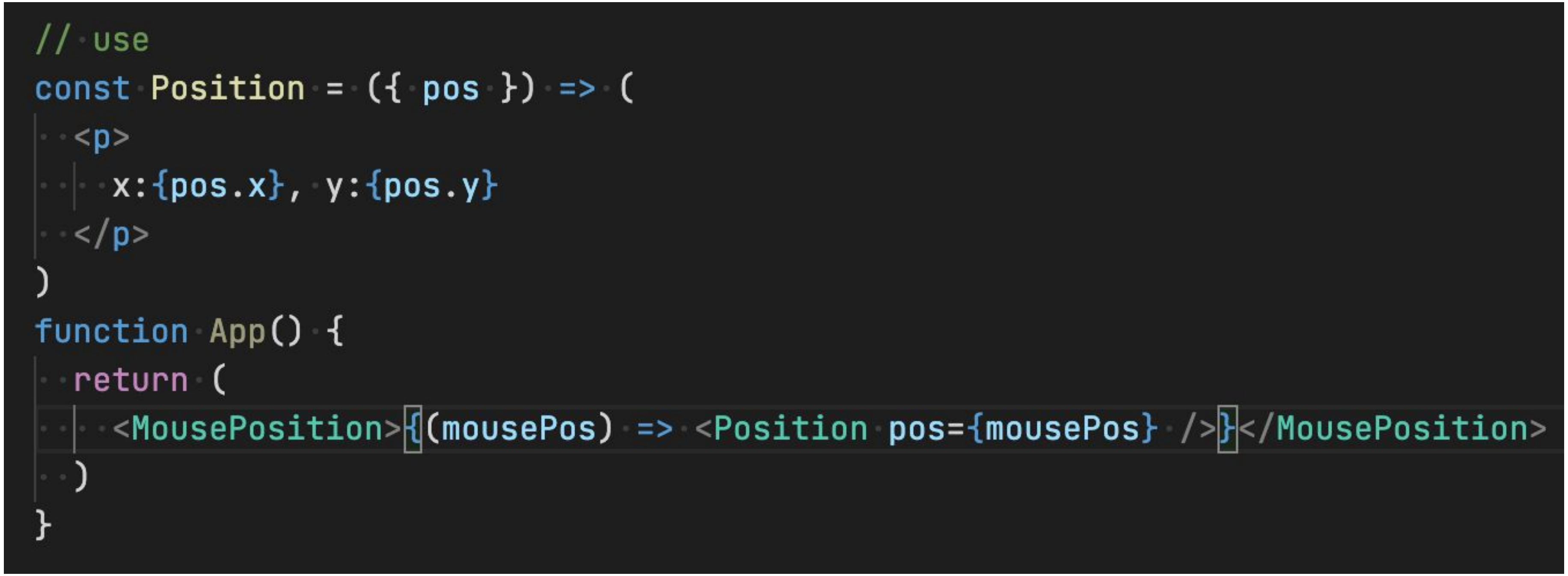
三层嵌套

render props



```
class MousePosition extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      x: 0,
      y: 0,
    }
    this.onMouseMove = this.onMouseMove.bind(this)
  }
  onMouseMove(ev) {
    this.setState({
      x: ev.pageX,
      y: ev.pageY,
    })
  }
  componentDidMount() {
    window.addEventListener('mousemove', this.onMouseMove)
  }
  componentWillUnmount() {
    window.removeEventListener('mousemove', this.onMouseMove)
  }
  render() {
    return (
      <div className="mouse-position-container">
        <h1>Mouse Position:</h1>
        {this.props.children(mousePos)}
      </div>
    )
  }
}
```

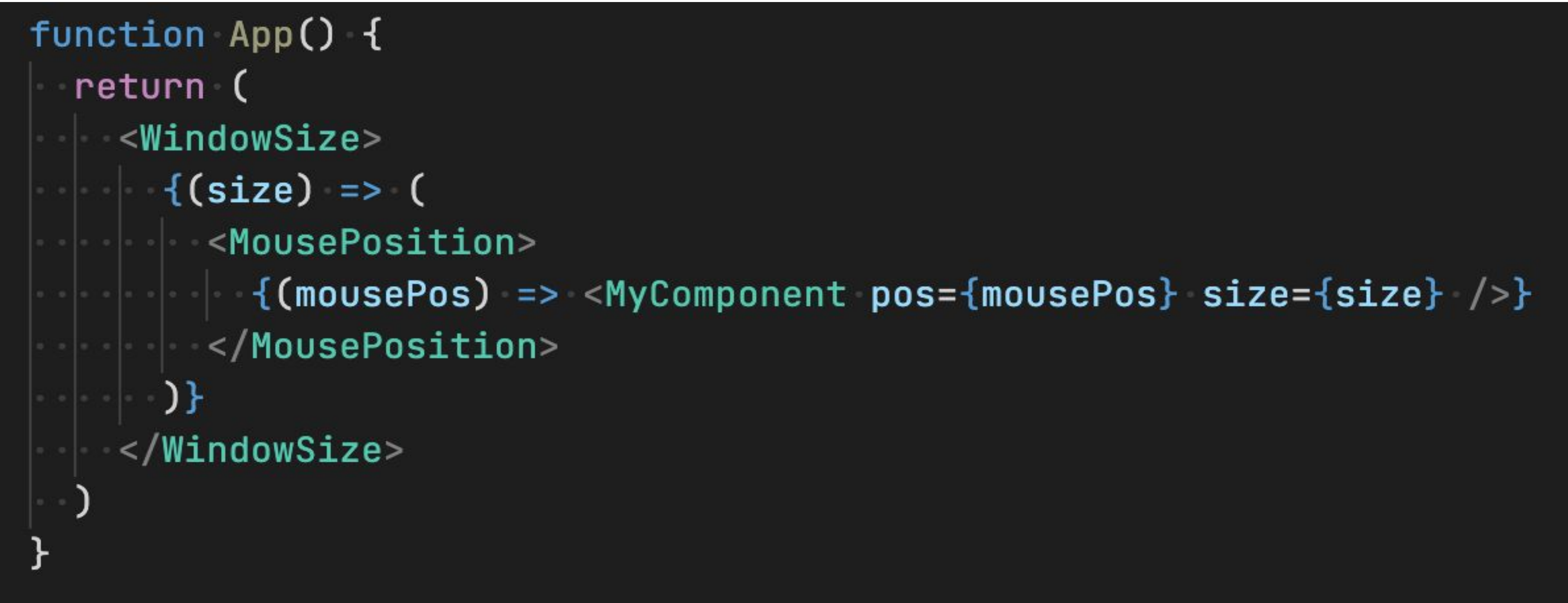
render props



```
// use
const Position = ({ pos }) => (
  <p>
    x:{pos.x}, y:{pos.y}
  </p>
)
function App() {
  return (
    <MousePosition>{(mousePos) => <Position pos={mousePos} />}</MousePosition>
  )
}
```

render props

如果再来一个 WindowSize 的组件，则这样使用：

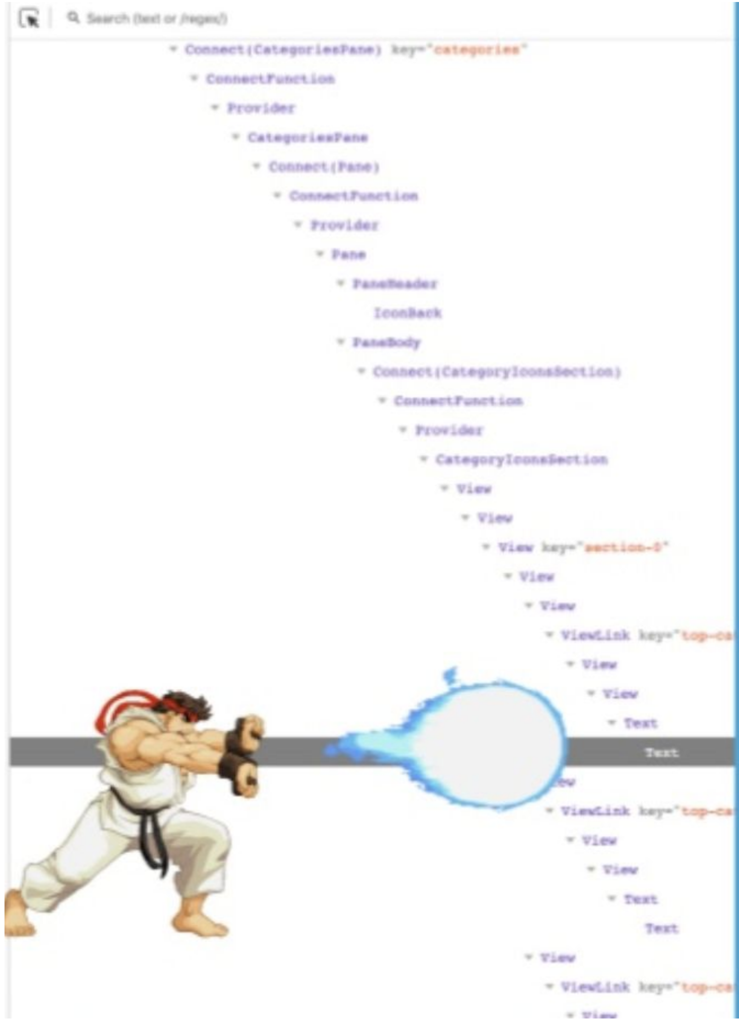


```
function App() {
  return (
    <WindowSize>
      {(size) => (
        <MousePosition>
          {(mousePos) => <MyComponent pos={mousePos} size={size} />}
        </MousePosition>
      )}
    </WindowSize>
  )
}
```

为啥要嵌套？

mouse Pos， window size 这些与生命周期有关的状态逻辑和 ui 应该是平级，而不需要嵌套。

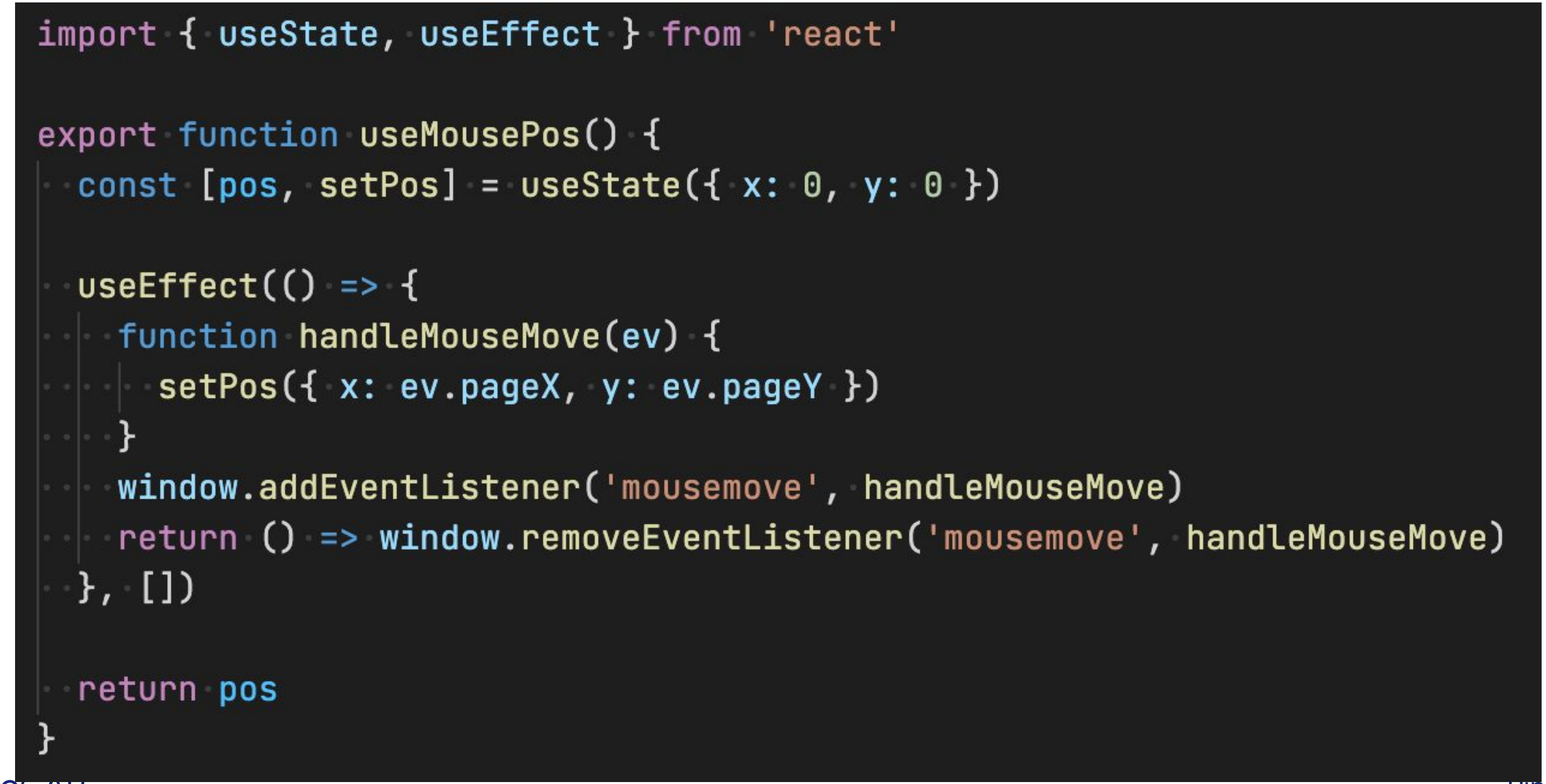
但由于一切皆组件，只有组件才有生命周期，造成了无谓的嵌套。



```
Search (text or /regex/)
▾ Connect(CategoriesPane) key="categories"
  ▾ ConnectFunction
    ▾ Provider
      ▾ CategoriesPane
        ▾ Connect(Pane)
          ▾ ConnectFunction
            ▾ Provider
              ▾ Pane
                ▾ PaneHeader
                    IconBack
                ▾ PaneBody
                  ▾ Connect(CategoryIconsSection)
                    ▾ ConnectFunction
                      ▾ Provider
                        ▾ CategoryIconsSection
                          ▾ View
                            ▾ View
                              ▾ View key="section-0"
                                ▾ View
                                  ▾ View
                                    ▾ ViewLink key="top-ca
                                      ▾ View
                                        ▾ View
                                          ▾ Text
                                              Text
                                  ew
                                    ▾ ViewLink key="top-ca
                                      ▾ View
                                        ▾ View
                                          ▾ Text
                                              Text
                                  ▾ View
                                    ▾ ViewLink key="top-ca
                                      ▾ View
```

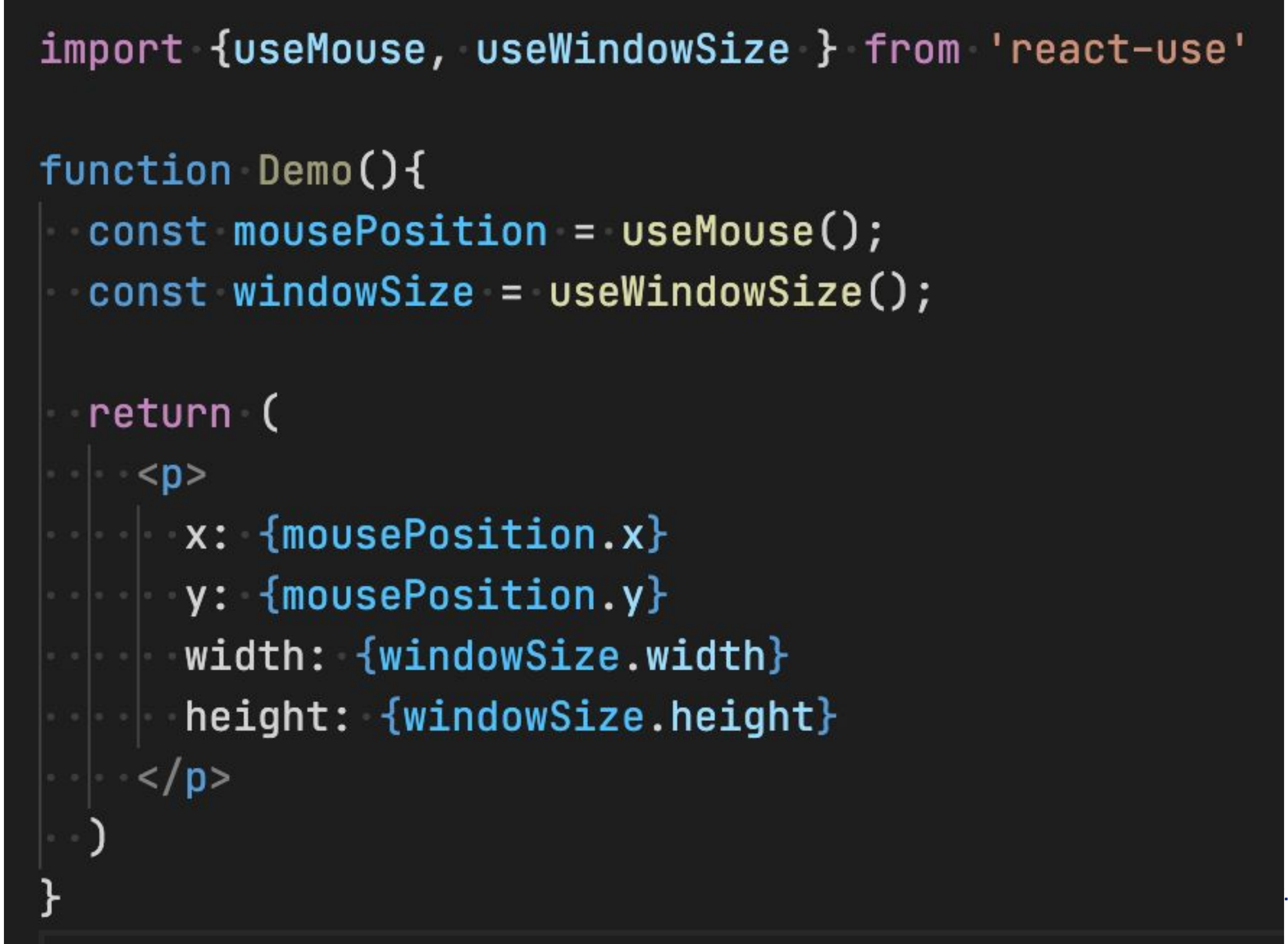
WRAPPER HELL

Hooks



```
import { useState, useEffect } from 'react'

export function useMousePos() {
  const [pos, setPos] = useState({ x: 0, y: 0 })

  useEffect(() => {
    function handleMouseMove(ev) {
      setPos({ x: ev.pageX, y: ev.pageY })
    }
    window.addEventListener('mousemove', handleMouseMove)
    return () => window.removeEventListener('mousemove', handleMouseMove)
  }, [])

  return pos
}
```

Hooks


```
import {useMouse, useWindowSize } from 'react-use'

function Demo(){
  const mousePosition = useMouse();
  const windowSize = useWindowSize();

  return (
    <p>
      x: {mousePosition.x}
      y: {mousePosition.y}
      width: {windowSize.width}
      height: {windowSize.height}
    </p>
  )
}
```

PingCAP

Part II - What

Hooks

> 它可以让你在不编写 class 的情况下使用 state 以及其他的 React 特性。

- 不再需要 class，只有一种 function component
- 不再有 this 指针问题
- 不再有诸多的生命周期函数 (隐含在内部)
- 自定义 Hook，非常方便地复用状态逻辑，且不用和组件嵌套，扁平 (即插即用，自由组合)
- ui 和状态更易分离 (除 ui 外的逻辑可以全部塞到一个自定义 hook 中)

Hooks 的一些问题

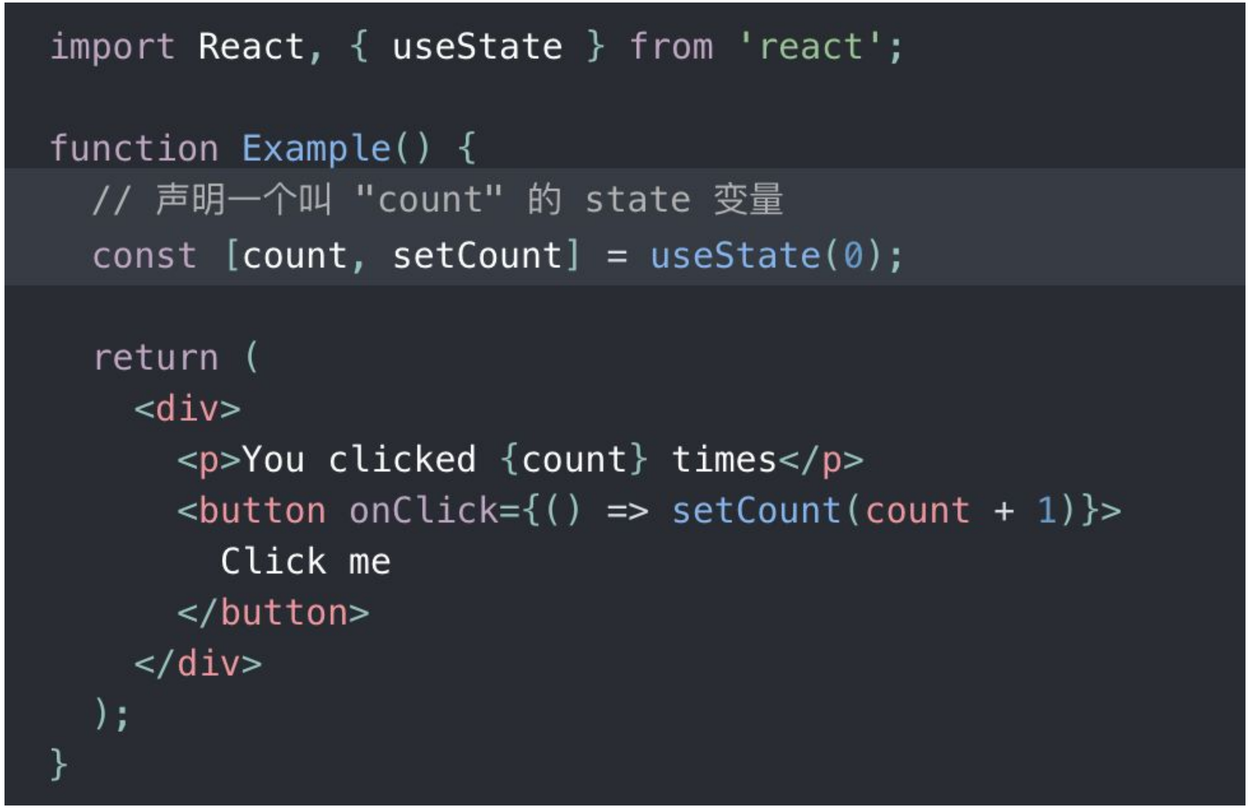
- 对闭包的理解要求很高
- 使用上要更加小心谨慎，使用不慎容易引起死循环，手工管理依赖，有心智负担 (有插件可以帮忙)

Part III - How

Hooks primitives

- useState
- useEffect
- ----
- useMemo
- useCallback
- ----
- useRef
- useContext
- useReducer

useState



```
import React, { useState } from 'react';

function Example() {
  // 声明一个叫 "count" 的 state 变量
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useState

与 class component 相比，将一个大的 state object 拆分成多个小的 state

before:

```
    this.state  = {age: 10, gender: ‘male’}
```

after:

```
    const [age, setAge] = useState(10);

    const [gender, setGender] = useState(‘male’);
```

useState

```
setAge(11);

setGender(‘female’);
```

连续的 setState 不会进行 merge，每一次 setState 都会进行 re-render()。如果需要修改的 state 很多，可以考虑用 useReducer() 优化。

useState(initState())，如果 initialState 是通过计算得到且代价较大，则可以用 useState(() => initState()) 进行优化，只在第一次执行计算函数。

useEffect

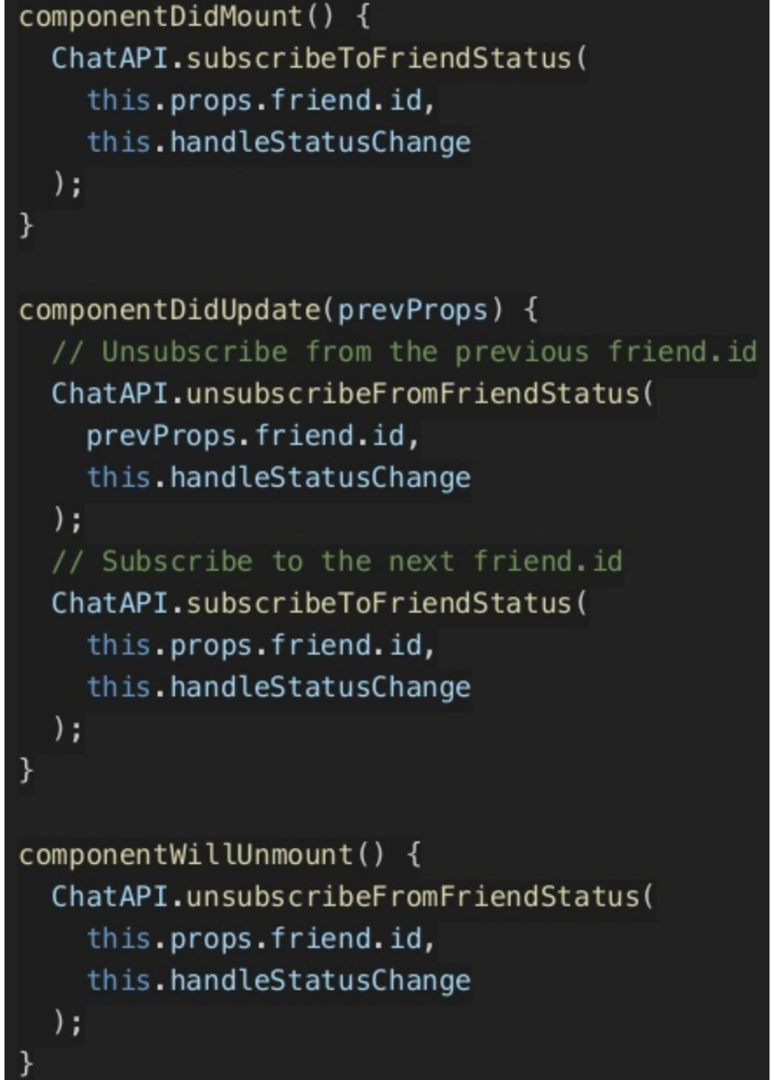
正如其名，用来执行副作用，比如访问网络，监听 dom。

useEffect(() => {...}, [deps])

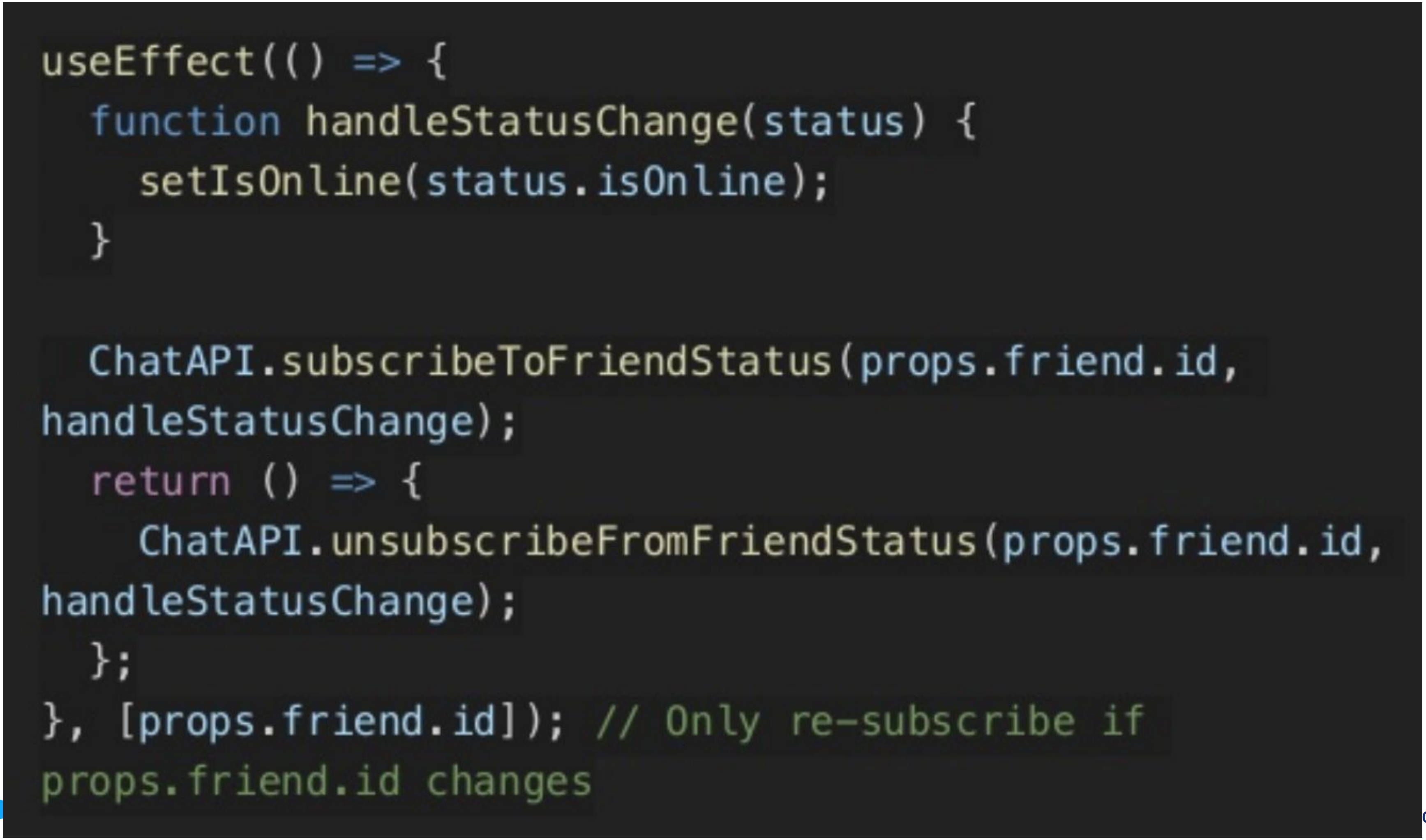
和 useState 是使用率最高的两个 hooks，也是使用最复杂的一个 hooks。

必看且反复看：

[useEffect 完整指南]



```
componentDidMount() {
  ChatAPI.subscribeToFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}

componentDidUpdate(prevProps) {
  // Unsubscribe from the previous friend.id
  ChatAPI.unsubscribeFromFriendStatus(
    prevProps.friend.id,
    this.handleStatusChange
  );
  // Subscribe to the next friend.id
  ChatAPI.subscribeToFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}

componentWillUnmount() {
  ChatAPI.unsubscribeFromFriendStatus(
    this.props.friend.id,
    this.handleStatusChange
  );
}
```



```
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  ChatAPI.subscribeToFriendStatus(props.friend.id,
handleStatusChange);
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id,
handleStatusChange);
  };
}, [props.friend.id]); // Only re-subscribe if
props.friend.id changes
```

useEffect

第二个参数是 deps。

如果 deps 不写，则每次 re-render useEffect 中的函数都会执行，后果就是可能不停地访问 API。

如果写成空数组 []，表示只在初次 render() 后执行一次，相当于 componentDidMount() 的效果。

useMemo / useCallback

- 优化性能
- useMemo 用来缓存通过 state 计算得到的值
- useCallback 用来缓存方法

useMemo

```
const [arr, setArr] = useState([])
const sortedArr = sort(arr)  // expensive
return <ul>{sortedArr.map(...)}</ul>
```

每次重新渲染时都要执行一次 sort(arr)。使用 useMemo() 优化之

```
const sortedArr = useMemo(() => sort(arr), [arr])
```

仅在 arr 发生变化时才会重新执行 sort(arr)

useCallback

```
function handleStatusChange(status) {
  setIsOnline(status.isOnline);
}

useEffect(() => {
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  return () => {
   ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
 }, [props.friend.id, handleStatusChange]);
```

产生的问题：死循环

useCallback

```
function handleStatusChange(status) {
  setIsOnline(status.isOnline);
}
```

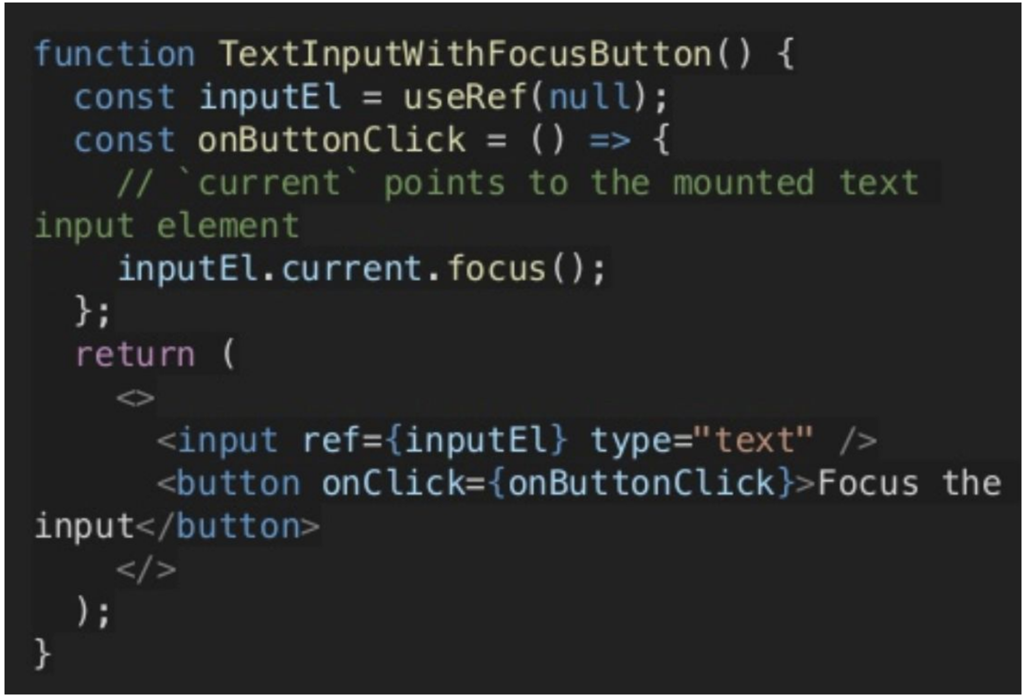
⇒

```
const handleStatusChange = useCallback((status) =>
setIsOnline(status.isOnline), [])
```

useRef

store mutable object, won’t cause re-render when changing ref object.

一般用来操作 dom



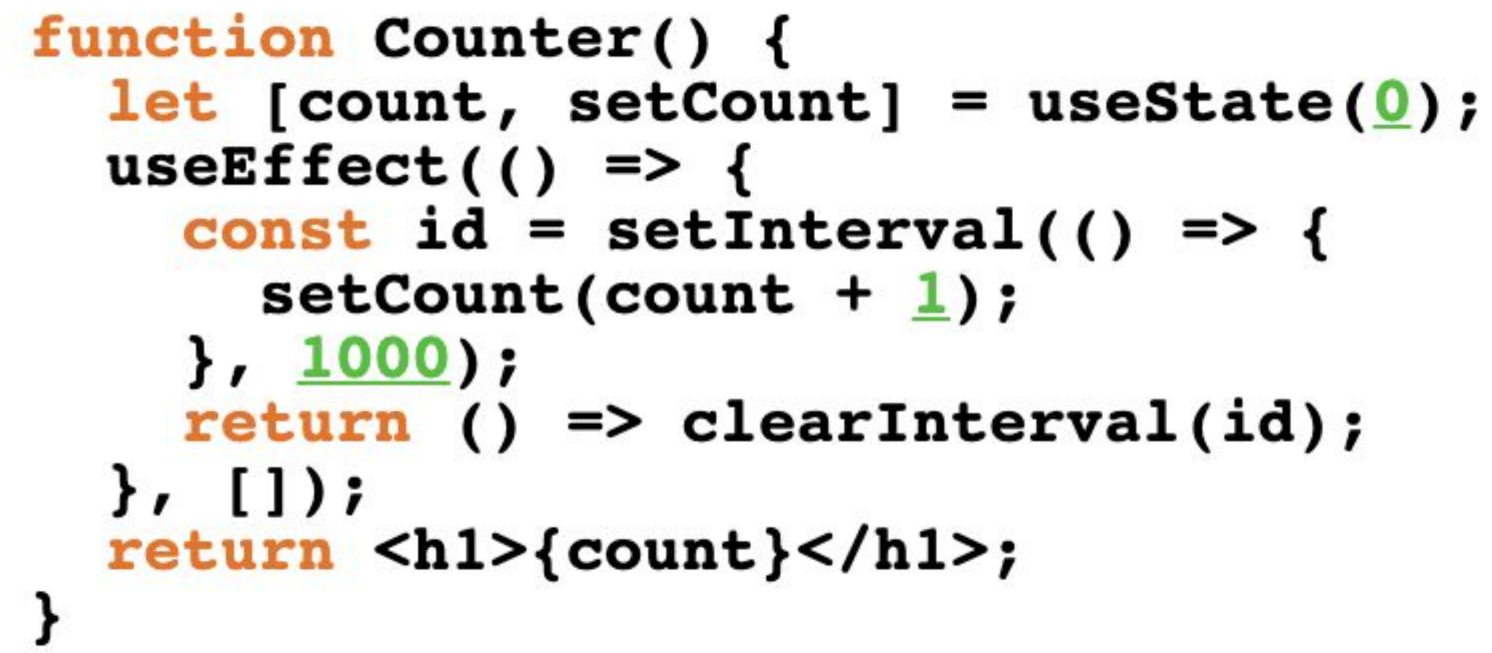
```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text
input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the
input</button>
    </>
  );
}
```

useRef

使用 React Hooks 声明 setInterval

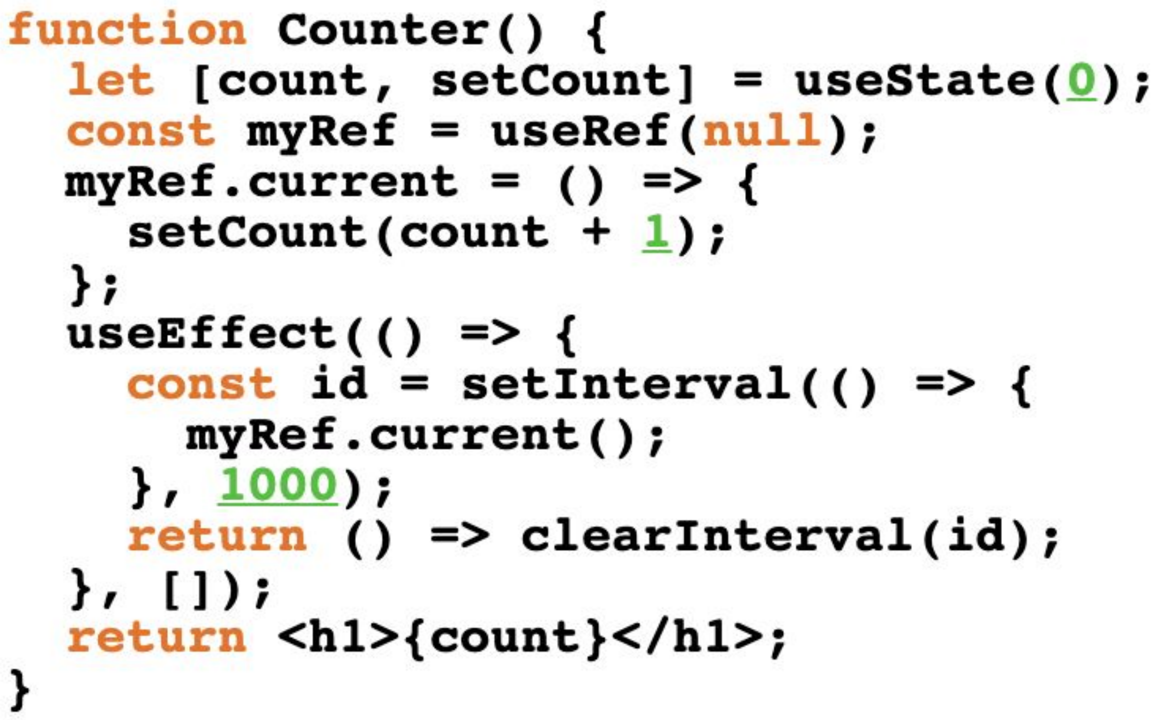
REACT HOOKS 与 SETINTERVAL

useRef



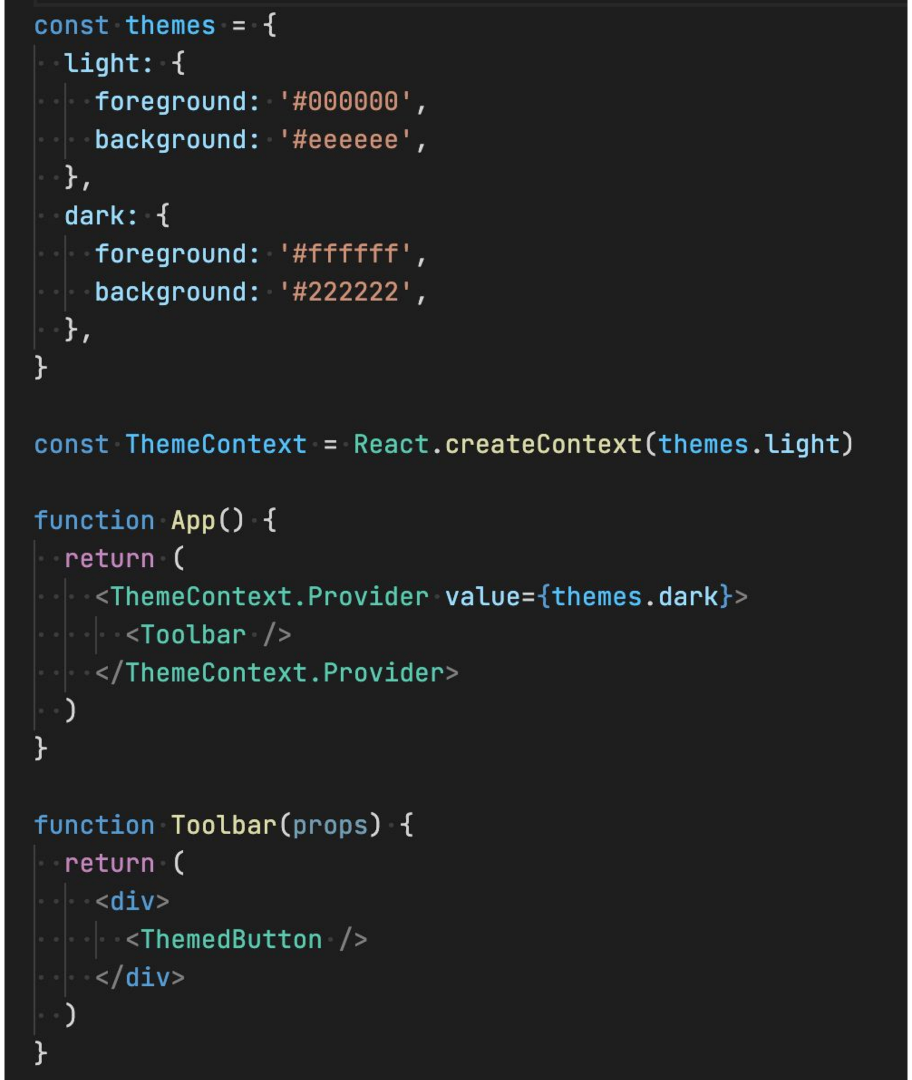
```
function Counter() {
  let [count, setCount] = useState(0);
  useEffect(() => {
    const id = setInterval(() => {
      setCount(count + 1);
    }, 1000);
    return () => clearInterval(id);
  }, []);
  return <h1>{count}</h1>;
}
```

useRef

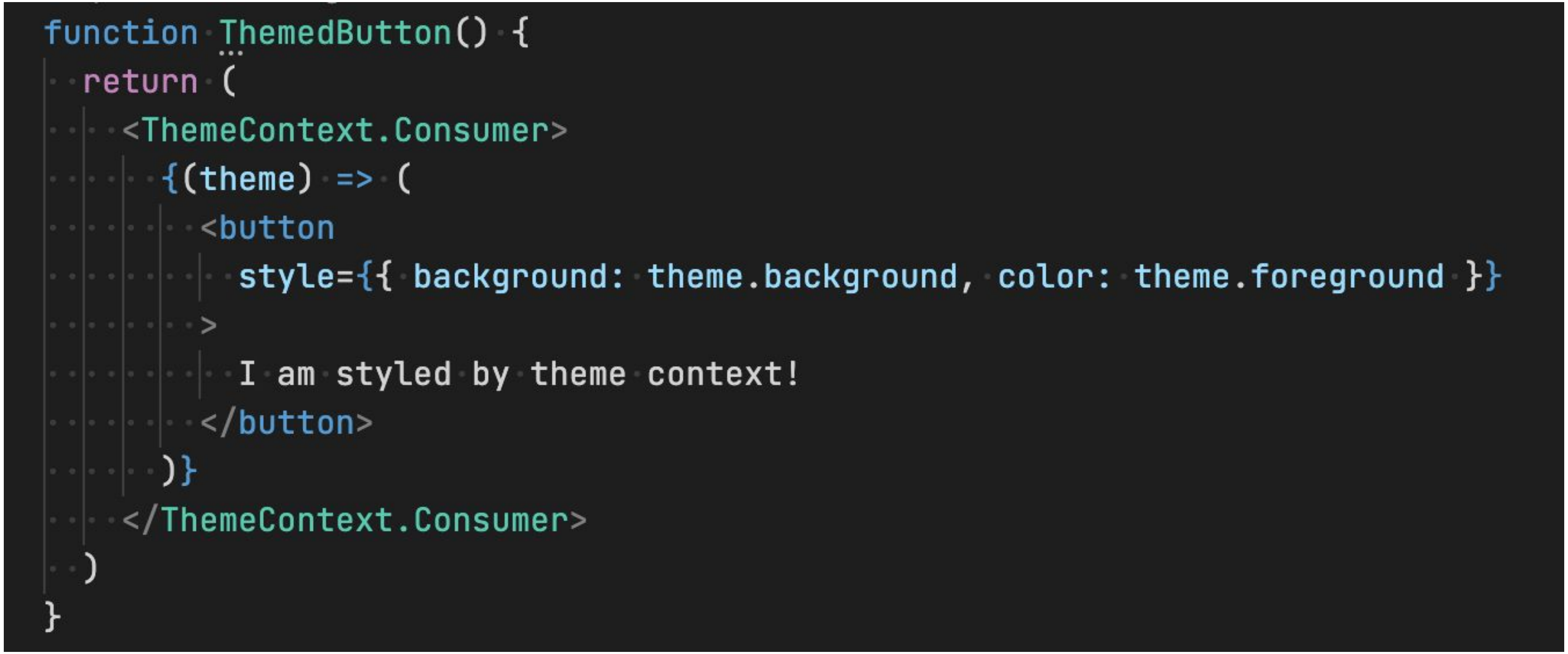


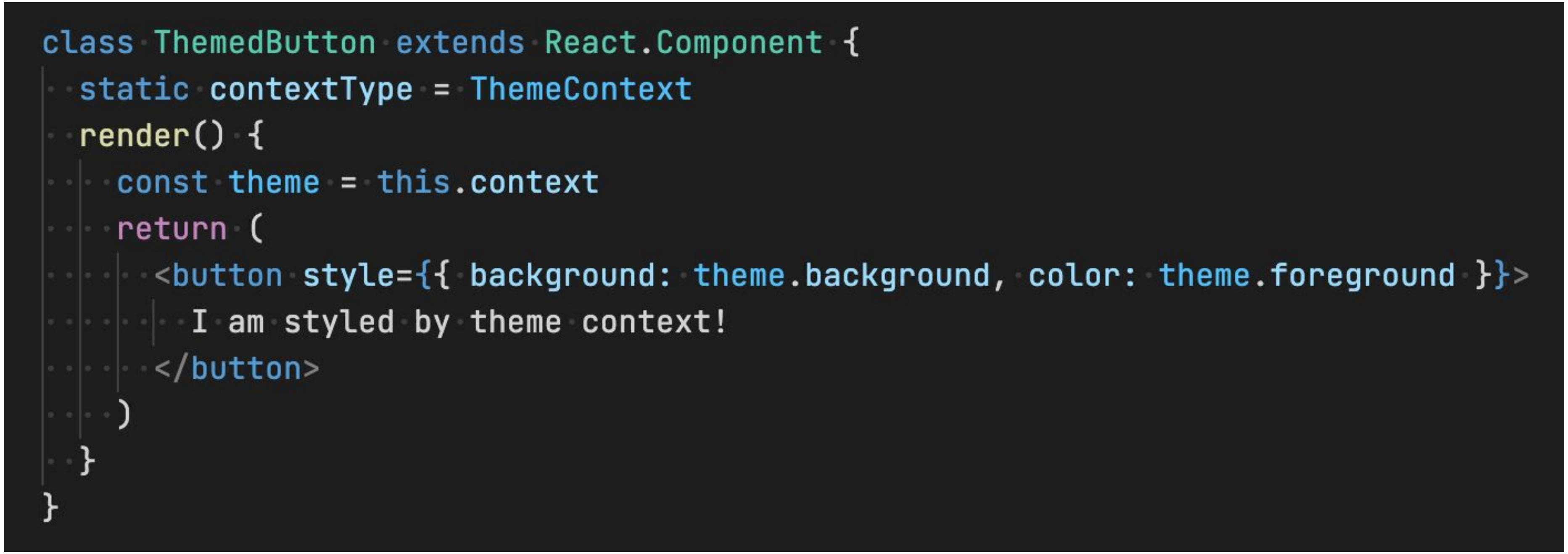
```
function Counter() {
  let [count, setCount] = useState(0);
  const myRef = useRef(null);
  myRef.current = () => {
    setCount(count + 1);
  };
  useEffect(() => {
    const id = setInterval(() => {
      myRef.current();
    }, 1000);
    return () => clearInterval(id);
  }, []);
  return <h1>{count}</h1>;
}
```

useContext

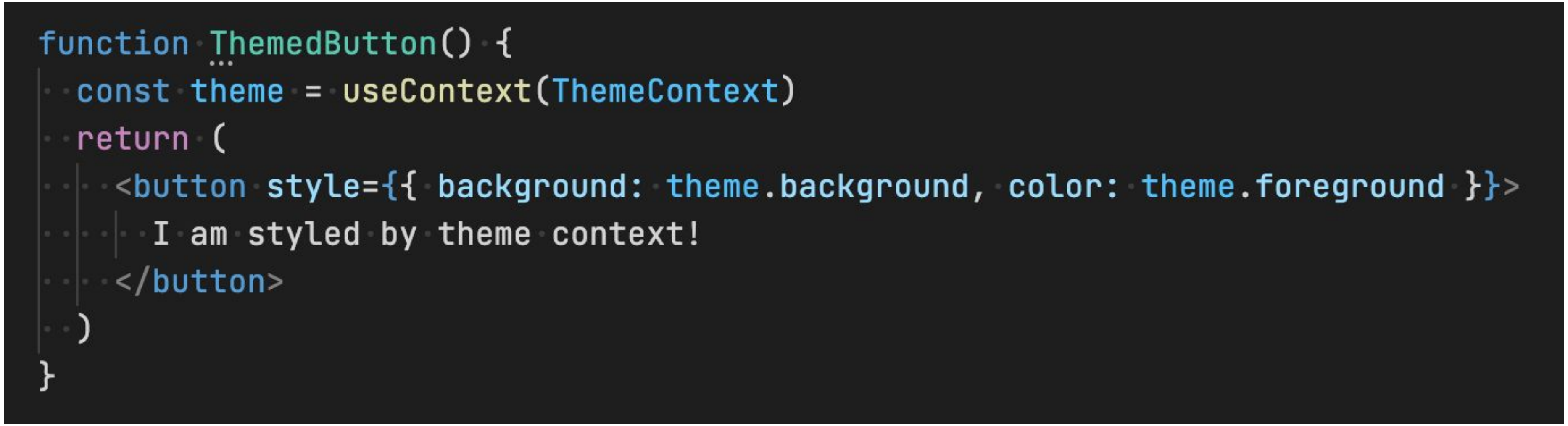


```
const themes = {
  light: {
    foreground: '#000000',
    background: '#eeeeee',
  },
  dark: {
    foreground: '#ffffff',
    background: '#222222',
  },
}

const ThemeContext = React.createContext(themes.light)

function App() {
  return (
    <ThemeContext.Provider value={themes.dark}>
      <Toolbar />
    </ThemeContext.Provider>
  )
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  )
}
```


```
function ThemedButton() {
  return (
    <ThemeContext.Consumer>
      {(theme) => (
        <button
          style={{ background: theme.background, color: theme.foreground }}
        >
          I am styled by theme context!
        </button>
      )}
    </ThemeContext.Consumer>
  )
}
```

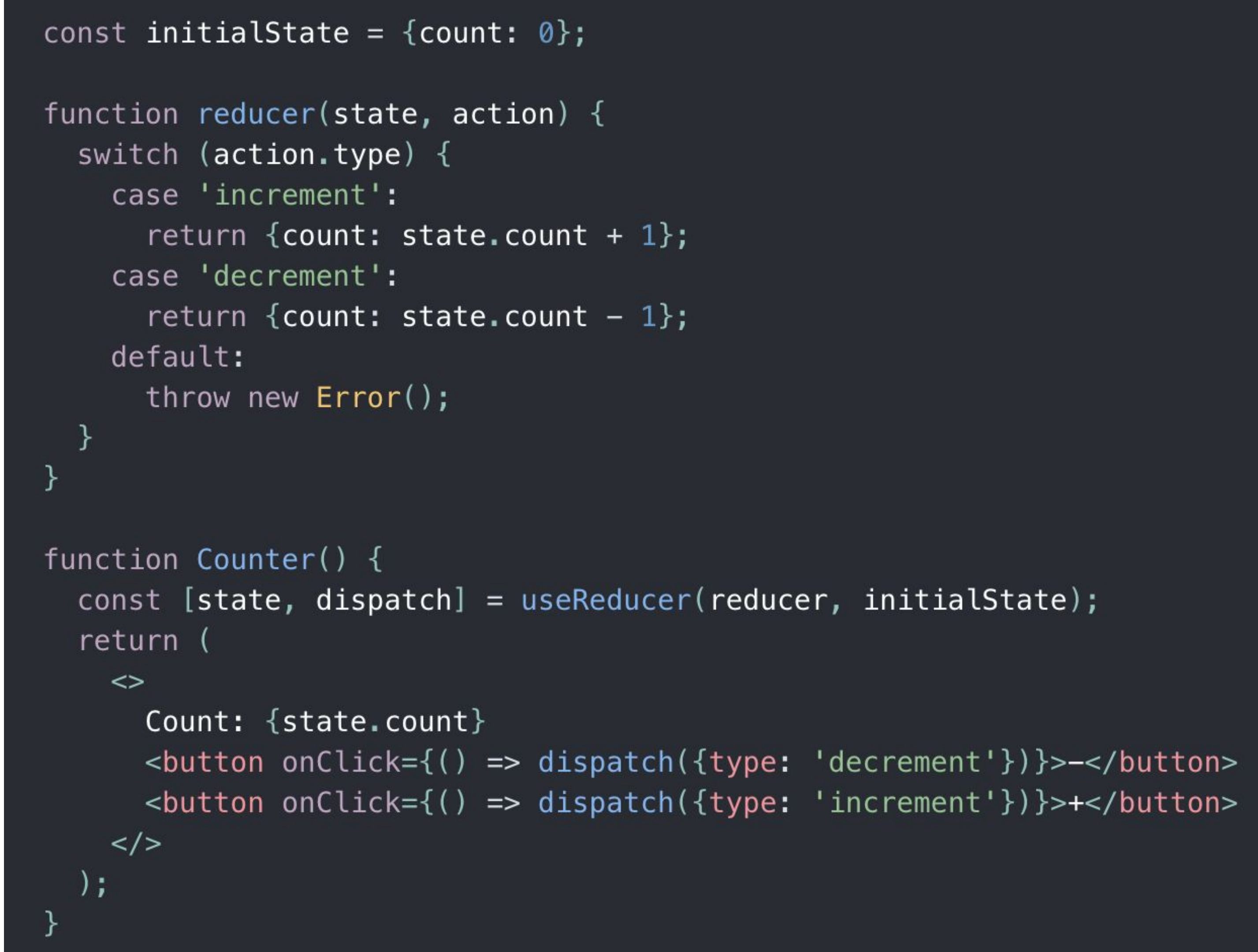


```
class ThemedButton extends React.Component {
  static contextType = ThemeContext
  render() {
    const theme = this.context
    return (
      <button style={{ background: theme.background, color: theme.foreground }}>
        I am styled by theme context!
      </button>
    )
  }
}
```



```
function ThemedButton() {
  const theme = useContext(ThemeContext)
  return (
    <button style={{ background: theme.background, color: theme.foreground }}>
      I am styled by theme context!
    </button>
  )
}
```

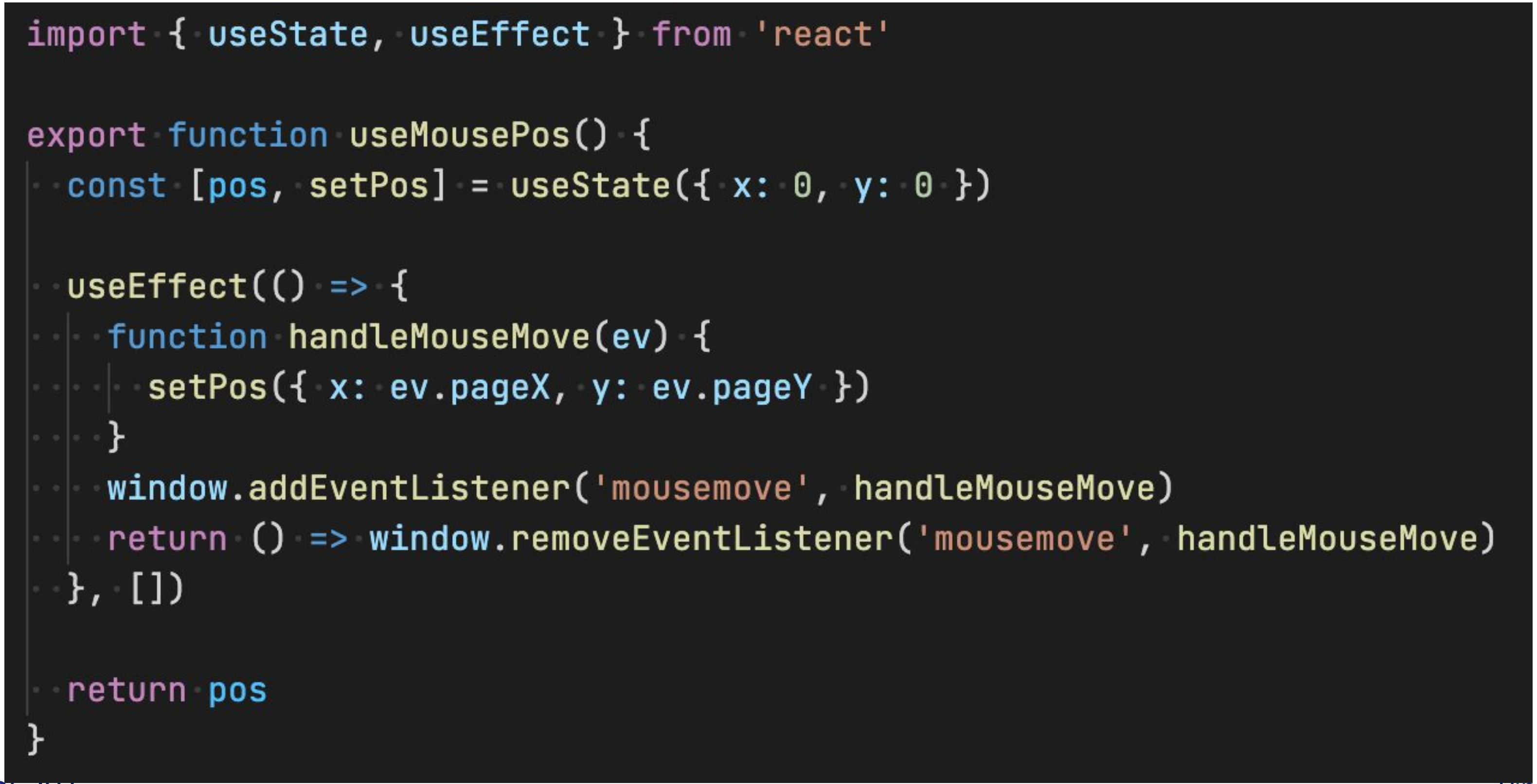
useReducer



```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

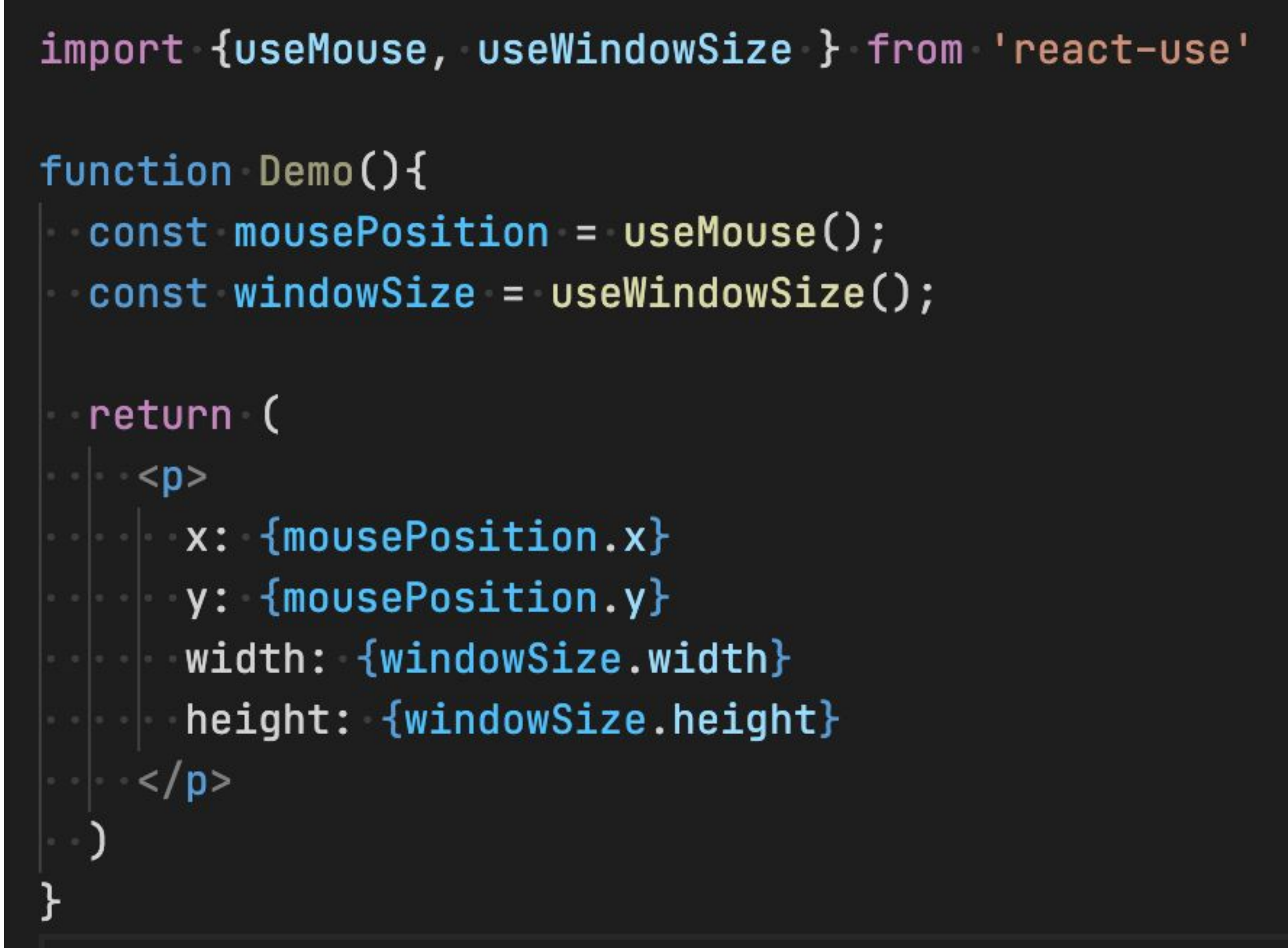
PingCAP

自定义 Hook



```
import { useState, useEffect } from 'react'

export function useMousePos() {
  const [pos, setPos] = useState({ x: 0, y: 0 })

  useEffect(() => {
    function handleMouseMove(ev) {
      setPos({ x: ev.pageX, y: ev.pageY })
    }
    window.addEventListener('mousemove', handleMouseMove)
    return () => window.removeEventListener('mousemove', handleMouseMove)
  }, [])

  return pos
}
```

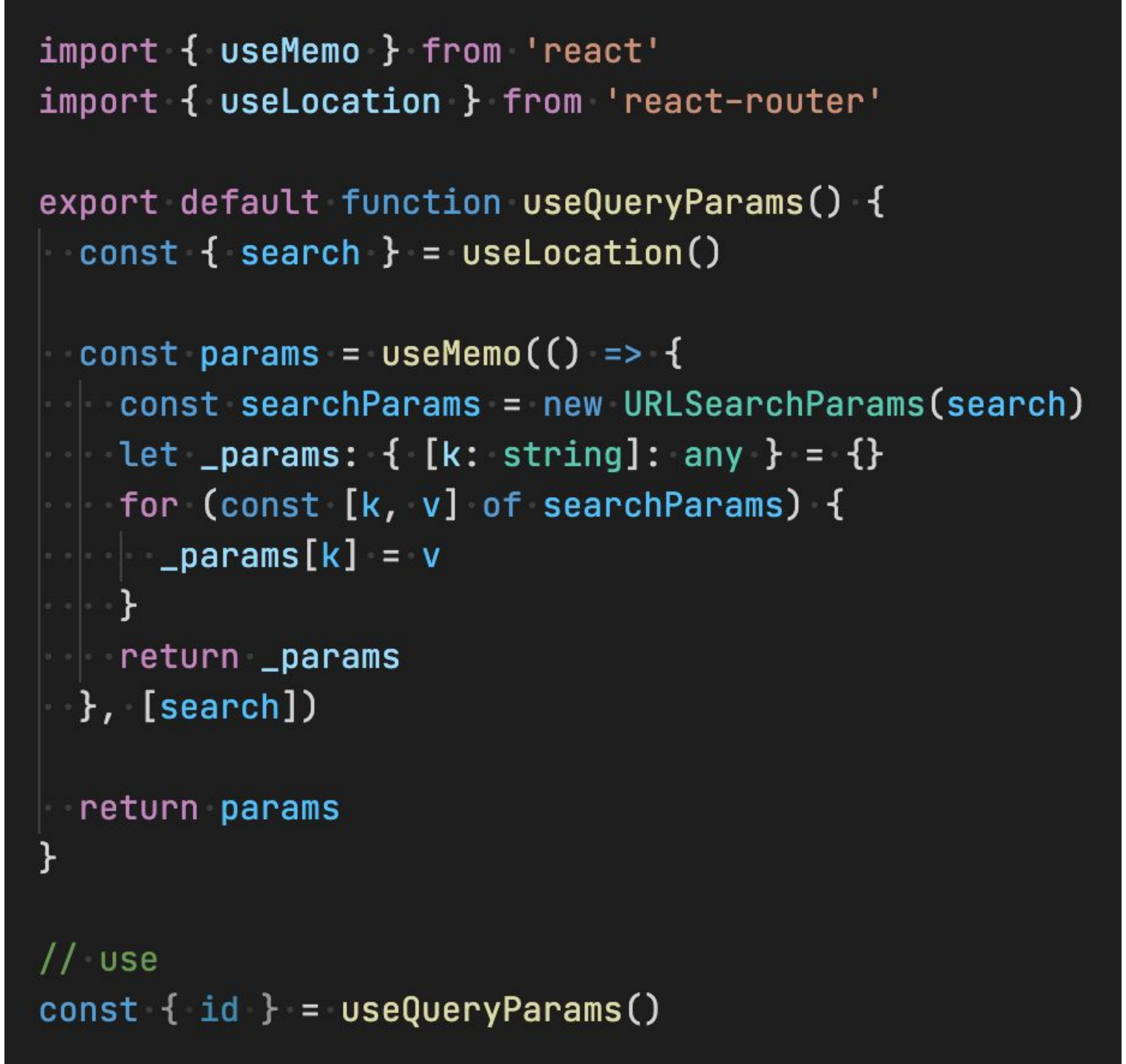
自定义 Hooks



```
import {useMouse, useWindowSize } from 'react-use'

function Demo(){
  const mousePosition = useMouse();
  const windowSize = useWindowSize();

  return (
    <p>
      x: {mousePosition.x}
      y: {mousePosition.y}
      width: {windowSize.width}
      height: {windowSize.height}
    </p>
  )
}
```

PingCAP

自定义 Hooks



```
import { useMemo } from 'react'
import { useLocation } from 'react-router'

export default function useQueryParams() {
  const { search } = useLocation()

  const params = useMemo(() => {
    const searchParams = new URLSearchParams(search)
    let _params: { [k: string]: any } = {}
    for (const [k, v] of searchParams) {
      _params[k] = v
    }
    return _params
  }, [search])

  return params
}

// use
const { id } = useQueryParams()
```

自定义 hooks

一个复杂一点的例子：

https://github.com/pingcap-incubator/tidb-dashboard/blob/master/ui/lib/apps/Statement/utils/useStatement.ts

使用注意事项

- 只能在最顶层使用，不能在循环，条件或嵌套函数中使用。
- 只能在 React 函数中 (包括自定义 hooks) 使用 hooks，在普通函数中使用 hooks 毫无意义
- 自定义 hooks 必须以 useXXX 格式命名

参考：

- 官方文档
- How do we use hooks
- React Hooks完全上手指南
- Umi Hooks - 助力拥抱 React Hooks
- 精读《React Hooks》

PingCAP

Thanks
Q&A