你可能并不需要useEffect

背景

相信大家在写react时都有这样的经历：在项目中使用了大量的useEffect，以至于让我们的代码变得混乱和难以维护。

难道说useEffect这个hook不好吗？并不是这样的，只是我们一直在滥用而已。

在这篇文章中，我将展示怎样使用其他方法来代替useEffect。

什么是useEffect

useEffect允许我们在函数组件中执行副作用。它可以模拟 componentDidMount、componentDidUpdate 和componentWillUnmount。我们可以用它来做很多事情。但是它也是一个非常危险的钩子，可能会导致很多bug。

为什么useEffect是容易出现bug的

来看一个定时器的例子：

javascript 复制代码

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

const Counter = () => {
  const [count, setCount] = useState(0)

  useEffect(() => {
    const timer = setInterval(() => {
      setCount((c) => c + 1)
    }, 1000)
    return () => clearInterval(timer)
```

```
  }, [])

  return <div>{count}</div>
}
```

这是一个非常常见的例子，但是它是非常不好。因为如果组件由于某种原因重新渲染，就会重新设置定时器。该定时器将每秒调用两次，很容易导致内存泄漏。

怎样修复它?

useRef

```javascript
import React, { useEffect, useRef } from 'react'

const Counter = () => {
  const [count, setCount] = useState(0)
  const timerRef = useRef()

  useEffect(() => {
    timerRef.current = setInterval(() => {
      setCount((c) => c + 1)
    }, 1000)
    return () => clearInterval(timerRef.current)
  }, [])

  return <div>{count}</div>
}
```

它不会在每次组件重新渲染时设置定时器。但是我们在项目中并不是这么简单的代码。而是各种状态，做各种事情。

你以为你写的useEffect

```scss
useEffect(() => {
  doSomething()
```

```
  return () => cleanup()
}, [whenThisChanges])
```

实际上是这样的

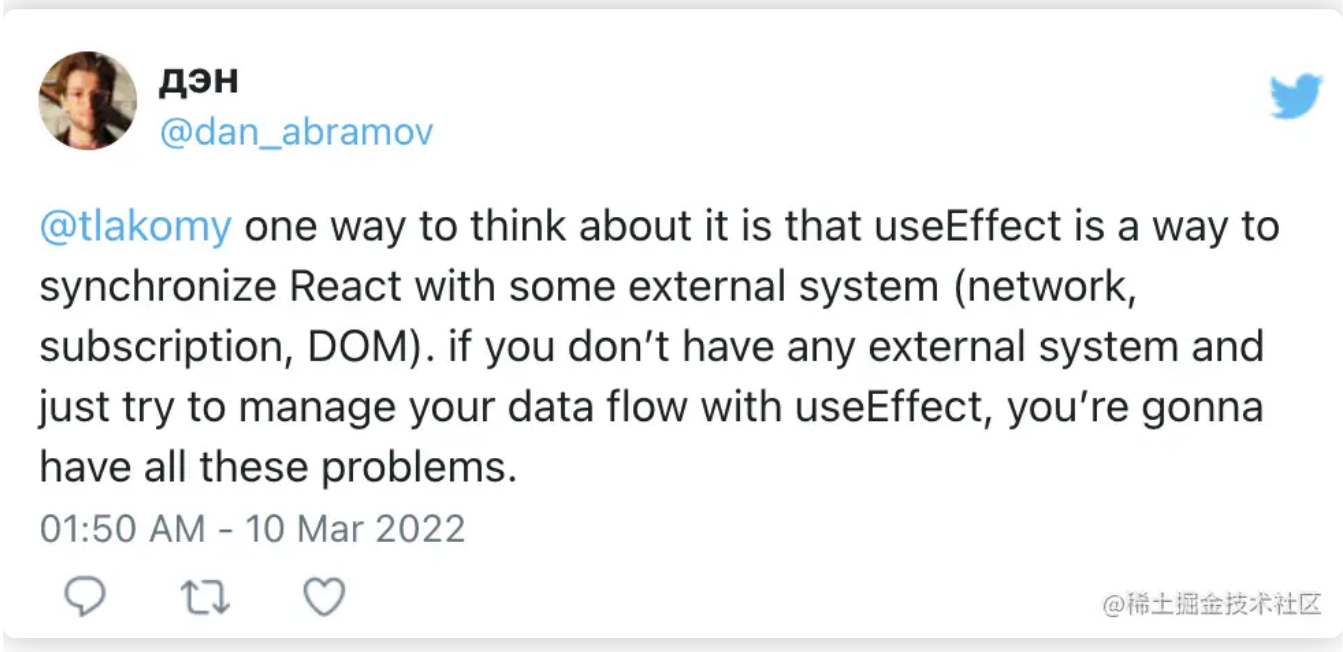
scss 复制代码

```
useEffect(() => {
  if (foo && bar && (baz || quo)) {
    doSomething()
  } else {
    doSomethingElse()
  }
  // 遗忘清理函数。
}, [foo, bar, baz, quo, ...])
```

写了一堆的逻辑，这种代码非常混乱难以维护。

useEffect 到底是用来干啥的

useEffect是一种将React与一些外部系统(网络、订阅、DOM)同步的方法。如果你没有任何外部系统，只是试图用useEffect管理数据流，你就会遇到问题。



有时我们并不需要useEffect

1.我们不需要useEffect转化数据

```scss
const Cart = () => {
  const [items, setItems] = useState([])
  const [total, setTotal] = useState(0)

  useEffect(() => {
    setTotal(items.reduce((total, item) => total + item.price, 0))
  }, [items])

  // ...
}
```

上面代码使用useEffect来进行数据的转化，效率很低。其实并不需要使用useEffect。当某些值可以从现有的props或state中计算出来时，不要把它放在状态中，在渲染期间计算它。

```scss
const Cart = () => {
  const [items, setItems] = useState([])
  const [total, setTotal] = useState(0)

  const totalNum = items.reduce((total, item) => total + item.price, 0)

  // ...
}
```

如果计算逻辑比较复杂，可以使用useMemo：

```javascript
const Cart = () => {
  const [items, setItems] = useState([])
  const total = useMemo(() => {
    return items.reduce((total, item) => total + item.price, 0)
  }, [items])
```

```
  // ...
}
```

2.使用useSyncExternalStore代替useEffect

useSyncExternalStore

常见方式：

```javascript
const Store = () => {
  const [isConnected, setIsConnected] = useState(true)

  useEffect(() => {
    const sub = storeApi.subscribe(({ status }) => {
      setIsConnected(status === 'connected')
    })

    return () => {
      sub.unsubscribe()
    }
  }, [])

  // ...
}
```

更好的方式：

```javascript
const Store = () => {
  const isConnected = useSyncExternalStore(
    storeApi.subscribe,
    () => storeApi.getStatus() === 'connected',
    true
  )

  // ...
}
```

3.没必要使用useEffect与父组件通信

```
复制代码
```

```scss
const ChildProduct = ({ onOpen, onClose }) => {
  const [isOpen, setIsOpen] = useState(false)

  useEffect(() => {
    if (isOpen) {
      onOpen()
    } else {
      onClose()
    }
  }, [isOpen])

  return (
    <div>
      <button
        onClick={() => {
          setIsOpen(!isOpen)
        }}
      >
        Toggle quick view
      </button>
    </div>
  )
}
```

更好的方式，可以使用事件处理函数代替：

```javascript 复制代码
const ChildProduct = ({ onOpen, onClose }) => {
  const [isOpen, setIsOpen] = useState(false)

const handleToggle = () => {
  const nextIsOpen = !isOpen;
  setIsOpen(nextIsOpen)

  if (nextIsOpen) {
    onOpen()
  } else {
    onClose()
  }
}

  return (
    <div>
      <button
        onClick={}
      >
        Toggle quick view
```

```
      </button>
    </div>
  )
}
```

4.没必要使用useEffect初始化应用程序

```scss
const Store = () => {
  useEffect(() => {
    storeApi.authenticate()
  }, [])

  // ...
}
```

更好的方式：

方式一：

```scss
const Store = () => {
  const didAuthenticateRef = useRef()

  useEffect(() => {
    if (didAuthenticateRef.current) return

    storeApi.authenticate()

    didAuthenticateRef.current = true
  }, [])

  // ...
}
```

方式二：

```ini
let didAuthenticate = false

const Store = () => {
  useEffect(() => {
```

```
    if (didAuthenticate) return

    storeApi.authenticate()

    didAuthenticate = true
  }, [])

  // ...
}
```

方式三：

```javascript 复制代码
if (typeof window !== 'undefined') {
  storeApi.authenticate()
}

const Store = () => {
  // ...
}
```

5.没必要在useEffect请求数据

常见写法

```scss 复制代码
const Store = () => {
  const [items, setItems] = useState([])

  useEffect(() => {
    let isCanceled = false

    getItems().then((data) => {
      if (isCanceled) return

      setItems(data)
    })

    return () => {
      isCanceled = true
    }
  })
```

```
  // ...
}
```

更好的方式：

没有必要使用useEffect，可以使用swr：

```javascript 复制代码
import useSWR from 'swr'

export default function Page() {
  const { data, error } = useSWR('/api/data', fetcher)

  if (error) return <div>failed to load</div>
  if (!data) return <div>loading...</div>

  return <div>hello {data}!</div>
}
```

使用react-query：

```javascript 复制代码
import { getItems } from './storeApi'
import { useQuery, useQueryClient } from 'react-query'

const Store = () => {
  const queryClient = useQueryClient()

  return (
    <button
      onClick={() => {
        queryClient.prefetchQuery('items', getItems)
      }}
    >
      See items
    </button>
  )
}

const Items = () => {
  const { data, isLoading, isError } = useQuery('items', getItems)

  // ...
}
```

没有正式发布的react的 use函数：

```javascript
function Note({ id }) {
  const note = use(fetchNote(id))

  return (
    <div>
      <h1>{note.title}</h1>
      <section>{note.body}</section>
    </div>
  )
}
```