Lesson 8: Side effect with useEffect

Created	March 22, 2022 9:55 AM
Tags	Empty
Related to Untitle...	Empty

💡 Hiểu rõ side effect là gì và cách xử lý chúng trong React

1. Side Effect là gì

Nhiệm vụ của các component và React chủ yếu là tạo ra giao diện người dùng. React đảm bảo việc thay đổi các dữ liệu trên màn hình dựa vào dữ liệu và các tương tác với ứng dụng từ phía người dùng.

Tuy nhiên có rất nhiều những tương tác từ ứng dụng web tới nhiều thành phần khác nhau. Một vài ví dụ có thể kể ra là:

- Xử lý HTTP request, response
- Tương tác với Local Storage, Session Storage.
- Xử lý timer (`setTimeout`, `setInterval`)
- Thực hiện subscribes / unsubscribes các dịch vụ bên ngoài.
- Tương tác trực tiếp với DOM

Những logic được thực thi bên ngoài phạm vi của component đều được gọi là các “side effect”. Các side effect có thể cập nhật lại giao diện ứng dụng ở một thời điểm khác nhau.

Xét một ví dụ đơn giản như sau:

1. Người dùng tiến hành đăng nhập. Khi click vào button “Login”, tiến hành gửi thông tin user lên server.
2. Nếu người dùng nhập chính xác, tiến hành điều hướng về trang chủ.
3. Nếu người dùng nhập không chính xác, tiến hành hiển thị lỗi yêu cầu người dùng nhập lại.

Ở trong ví dụ trên, bước 1 là công việc của component. Tuy nhiên, bước 2 hoặc bước 3 sẽ được thực hiện dựa vào thông tin kết quả trả về. Đó có thể coi là các side effect, khi việc cập nhật giao diện ứng dụng sẽ phụ thuộc vào kết quả từ bên ngoài.

💡 Side effect là một khái niệm trong các ứng dụng nói chung, không chỉ riêng với React. Mỗi một công cụ sẽ có những cách khác nhau để xử lý các side effect. Vì vậy, việc hiểu rõ side effect là một điều quan trọng trong việc xây dựng các ứng dụng hiện đại.

2. Side effect cơ bản với `useEffect`

Để xử lý các logic bên ngoài component, React cung cấp một function hook đặc biệt là `useEffect`. Cú pháp cơ bản của useEffect như sau:

```
useEffect(() => { /* effect here */ })
```

Xét một ví dụ sau:

```
import {useState, useEffect} from 'react' const App = () => { const [count, setCount] = useState(0) useEffect(() => { document.title = `You clicked ${count} times`; }) return ( <div> <p>You clicked {count} times</p> <button onClick={() => setCount(count + 1)}> Click me </button> </div> ); }
```

🤔 **`useEffect` thực hiện việc gì?**
Với việc sử dụng useEffect hooks, chúng ta đã nói với React rằng hãy thực thi thêm công việc này sau khi component được render. React sẽ ghi nhớ function chúng ta đã truyền vào và gọi lại nó sau khi DOM được cập nhật.

🤔 **Tại sao `useEffect` được gọi bên trong component?**
Đặt `useEffect` vào bên trong component sẽ giúp effect truy cập được tới các giá trị state và props bên trong component.

🤔 **`useEffect` sẽ chạy sau mỗi lần render?**
Đúng. Mặc định, các useEffect sẽ luôn chạy sau lần render đầu tiên và sau mỗi lần giao diện được cập nhật. Chúng ta sẽ cùng học cách tuỳ chỉnh nó ở phần kế tiếp.

---

3. Side effect với “cleanup” function

Trong nhiều trường hợp, ví dụ như subscribe một dịch vụ, chúng ta sẽ cần phải thực hiện việc ngắt kết nối để tránh hiện tượng thất thoát bộ nhớ. `useEffect` có cơ chế để giải quyết vấn đề trên:

```
import {useEffect} from 'react' const App = () => { useEffect(() => { const handleScroll = () => { const position = document.documentElement.scrollTop; console.log("scrolling position: ", position); } document.addEventListener("scroll", handleScroll) return () => { document.removeEventListener("scroll", handleScroll) } }) return <div>Hello</div> }
```

Nếu `useEffect` trả về kết quả là một function, function đó sẽ được gọi mỗi khi React nhận thấy đó là lúc để “dọn dẹp” effect đó.

🤔 **Tại sao lại là trả về một function thay vì tạo ra một effect mới?**
Đây là cơ chế để thực hiện việc cleanup cho effect. Việc này sẽ giúp cho logic của một effect được đứng cùng nhau trong source code, vì chúng là các thành phần của chung một effect.

🤔 **Khi nào React thực hiện việc cleanup effect**
React thực hiện việc cleanup khi component được gỡ khỏi màn hình. Tuy nhiên, các effect được chạy sau mỗi lần render. Vì vậy, React cũng sẽ thực hiện việc cleanup với mỗi lần render. Mục đích là để xoá effect từ lần render trước đó trước khi thực hiện effect cho lần tới. Việc này cũng sẽ tạo ra vấn đề với hiệu năng của ứng dụng. Chúng ta sẽ học cách tránh điều này xảy ra ở phần bên dưới.

---

4. Các tips với `useEffect`

Sử dụng nhiều `useEffect` để chia nhỏ các vấn đề khác nhau

Trong một component, chúng ta có thể sử dụng nhiều effect cho các vấn đề khác nhau. Việc chia nhỏ useEffect này sẽ giúp việc tổ chức code bên trong một component **theo chức năng** của nó. Code sẽ dễ bảo trì hơn.

```
import {useState, useEffect} from 'react' const App = () => { const [count, setCount] = useState(0) // effect for update title useEffect(() => { document.title = `You clicked ${count} times`; }) // effect for scrolling useEffect(() => { document.addEventListener("scroll", () => { const position = document.documentElement.scrollTop; console.log("scrolling position: ", position); }) }) return ( <div> <p>You clicked {count} times</p> <button onClick={() => setCount(count + 1)}> Click me </button> </div> ); }
```

Tối ưu hoá hiệu năng của component với việc bỏ qua các effect.

Việc cleanup effect và áp dụng effect sau mỗi lần render sẽ tạo các vấn đề liên quan tới hiệu năng của ứng dụng, khi việc thay đổi là quá thường xuyên. Để tránh việc làm không cần thiết này. Chúng ta có thể nói với React để nó bỏ qua các lần chạy effect nếu như một vài giá trị nhất định không thay đổi. Việc này được thực hiện bằng cách **truyền thêm cho `useEffect` một tham số thứ 2 có giá trị là một array.** Array này thường được gọi là "dependencies array" hoặc ngắn gọn là "deps array".

```
import {useState, useEffect} from 'react' const App = () => { const [count, setCount] = useState(0) const [visible, setVisible] = useState(false) // effect for update title useEffect(() => { console.log("count changed") document.title = `You clicked ${count} times`; }, [count]) return ( <div> <p>You clicked {count} times</p> <button onClick={() => setCount(count + 1)}> Click me </button> <div> {visible && <p>This is hidden content</p>} <button onClick={() => setVisible(!visible)}>Change visible</button> </div> </div> ); }
```

Với ví dụ trên, chúng ta đã truyền `[count]` là tham số thứ 2 của `useEffect`. Điều này có nghĩa là: khi giá trị của `count` = `0`, component được render với giá trị count tương ứng, React tiến hành so sánh giá trị cũ `[0]` và giá trị mới `[0]` của `count`. Vì tất cả các phần tử đều bằng nhau giữa 2 lần render (`0 === 0`), React sẽ bỏ qua effect của lần render đó.

Nếu giá trị `count` khác đi, do kết quả cũ và mới đã có sự khác nhau, React sẽ tiến hành thực thi effect đó như bình thường.

Khi sử dụng cách tối ưu hoá này cần lưu ý:

- Deps array cần bao gồm tất cả các giá trị nằm trong component có thể thay đổi theo các lần render: state, props và các giá trị tính toán khác. Nếu không, giá trị bên trong effect sẽ là giá trị của lần render cũ đã là lỗi thời.

- Deps array có thể nhận vào giá trị là một empty array. Với việc này, giá trị của deps array không bị thay đổi theo thời gian. Điều này nghĩa là effect đó sẽ chỉ được chạy một lần duy nhất và không bao giờ được chạy lại.

> 📌 Các function “setState” không cần thiết phải được liệt kê bên trong deps array của `useEffect`. Tuy nhiên các state hoặc props cần phải được liệt kê đầy đủ. Nếu muốn cập nhật giá trị state bên trong `useEffect`, chúng ta có thể sử dụng cú pháp `setState(prev => { return newState})` để tránh việc phải liệt kê các state bên trong deps array, có thể làm component bị rơi vào trường hợp lặp vô tận.