Khóa học Front-end ATT Lab - ReactJS

Tổng quan 🎯

1. Cleanup function
2. Strict mode
3. Custom hook

1. Cleanup function

Tác dụng:

- Thu gom rác khi một `component unmount`: Khi một component bị gỡ bỏ khỏi cây DOM, các side effect như subscription, timer, hoặc sự kiện được thiết lập trong useEffect cần được dọn dẹp để tránh tình trạng memory leak.
- Tránh tình trạng `memory leak`: `Memory leak` xảy ra khi tài nguyên không được giải phóng đúng cách, dẫn đến ứng dụng sử dụng nhiều bộ nhớ hơn mức cần thiết và có thể làm chậm hoặc gây lỗi ứng dụng.

Cách sử dụng

Lý thuyết:

Cleanup function được định nghĩa trong useEffect và được trả về từ hàm mà bạn truyền vào useEffect. React sẽ gọi hàm này khi component bị unmount hoặc trước khi thực hiện hiệu ứng tiếp theo (khi dependencies thay đổi).

Cú pháp:

```
useEffect(() => {
  // Side effect logic

  return () => {
    // Cleanup logic
  };
}, [dependencies]);
```

Ví dụ minh họa

Ví dụ 1: Cleanup function với timer

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

function TimerComponent() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const timer = setInterval(() => {
      setCount(prevCount => prevCount + 1);
    }, 1000);

    // Cleanup function to clear the interval
    return () => {
      clearInterval(timer);
    };
  }, []);
```

```
  return (
    <div>
      <p>Count: {count}</p>
    </div>
  );
}

export default TimerComponent;
```

> Trong ví dụ này, clearInterval(timer) được gọi trong cleanup function để xóa bỏ setInterval khi component bị unmount hoặc trước khi tạo một interval mới.

Ví dụ 2: Cleanup function với event listener

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

function ResizeComponent() {
  const [windowWidth, setWindowWidth] = useState(window.innerWidth);

  useEffect(() => {
    const handleResize = () => {
      setWindowWidth(window.innerWidth);
    };

    window.addEventListener('resize', handleResize);

    // Cleanup function to remove the event listener
    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []);

  return (
    <div>
      <p>Window width: {windowWidth}px</p>
    </div>
  );
}

export default ResizeComponent;
```

Bài tập

- https://codesandbox.io/p/sandbox/loving-pond-9zmf32
- https://codesandbox.io/p/sandbox/nameless-wave-k434ch

2. Strict Mode

2.1. Giới thiệu Strict Mode

Lý thuyết:

StrictMode là một công cụ dành cho nhà phát triển được cung cấp bởi React để giúp bạn viết mã một cách an toàn và hiệu quả hơn. Nó không render bất kỳ UI nào, nhưng sẽ kích hoạt các kiểm tra bổ sung và cảnh báo trong chế độ phát triển.

Các chức năng của StrictMode:

- Phát hiện các side effect không mong muốn trong render.
- Xác định các thành phần sử dụng các API đã lỗi thời.
- Phát hiện các lỗi tiềm ẩn trong ứng dụng.

2.2. Cách sử dụng Strict Mode

Lý thuyết:

Để sử dụng StrictMode, bạn chỉ cần bao bọc phần cây component mà bạn muốn kiểm tra bằng React.StrictMode.

Ví dụ:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

> Trong ví dụ này, toàn bộ ứng dụng App được bao bọc bởi React.StrictMode, giúp kích hoạt các kiểm tra bổ sung trong chế độ phát triển.

2.3. Một số ví dụ về cảnh báo của Strict Mode

Ví dụ:

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

function StrictModeDemo() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Component mounted');
  }, []);

  return (
```

```
      <div>
        <p>Count: {count}</p>
        <button onClick={() => setCount(count + 1)}>Increment</button>
      </div>
    );
}

export default StrictModeDemo;
```

Trong chế độ phát triển, React.StrictMode sẽ cảnh báo nếu bạn sử dụng bất kỳ API lỗi thời hoặc có side effect không mong muốn.

Kết luận

> StrictMode giúp phát hiện các vấn đề tiềm ẩn trong ứng dụng React, đảm bảo mã của bạn an toàn và hiệu quả hơn. Custom Hook giúp tái sử dụng logic stateful, giúp mã của bạn dễ quản lý và tái sử dụng. Bằng cách học và áp dụng các kỹ thuật này, bạn sẽ trở thành một nhà phát triển React chuyên nghiệp và hiệu quả hơn.

3. Custom Hook

3.1. Giới thiệu Custom Hook

Lý thuyết:

Custom Hook là một hàm JavaScript mà bạn viết để sử dụng lại logic stateful giữa các component trong React. Nó cho phép bạn trừu tượng hóa và chia sẻ logic giữa các component mà không cần dùng đến các lớp (class).

Một Custom Hook:

Là một hàm bắt đầu với tiền tố `use`. Có thể sử dụng các Hook khác như useState, useEffect, v.v.

3.2. Cách tạo Custom Hook

Lý thuyết:

Custom Hook cho phép bạn trừu tượng hóa logic và tái sử dụng nó trong các component khác nhau.

Ví dụ:

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

// Custom Hook để quản lý bộ đếm
function useCounter(initialValue = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(initialValue);
```

```
  return [count, increment, decrement, reset];
}

// Component sử dụng Custom Hook
function CounterComponent() {
  const [count, increment, decrement, reset] = useCounter(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

export default CounterComponent;
```

Bài tập

- https://codesandbox.io/p/sandbox/buoi07-bt1-3yndck
- https://codesandbox.io/p/sandbox/buoi07-bt2-c6rscw
- https://codesandbox.io/p/sandbox/buoi07-bt2-4gkwpg