Khóa học Front-end ATT Lab - ReactJS

Tổng quan 🎯

1. Tại sao React re-render
2. Giới thiệu useMemo
3. Giới thiệu React.memo
4. Giới thiệu useCallback

1. Tại sao React re-render

1.1. Lý thuyết

Tại sao React re-render:

React re-render một component khi:

- Props của component thay đổi.
- State của component thay đổi.
- Parent component re-render.

> Hiểu lý do tại sao React re-render giúp bạn tối ưu hóa ứng dụng và tránh những re-render không cần thiết.

1.2. Ví dụ Ví dụ: Một component đơn giản thay đổi state khi người dùng nhấn nút.

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

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

  const increment = () => {
    setCount(count + 1);
  };

  console.log('Counter re-rendered');

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

export default Counter;
```

2. Giới thiệu useMemo

2.1. Lý thuyết

useMemo:

useMemo là một hook giúp bạn ghi nhớ (memoize) giá trị của một tính toán đắt đỏ (expensive calculation) chỉ khi một trong những dependencies thay đổi. Điều này giúp tránh việc tính toán lại không cần thiết khi component re-render.

2.2. Ví dụ

Ví dụ:

Component tính toán giá trị gấp đôi của count và chỉ tính toán lại khi count thay đổi.

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

function ExpensiveCalculation() {
  const [count, setCount] = useState(0);
  const [otherState, setOtherState] = useState(false);

  const doubleCount = useMemo(() => {
    console.log('Calculating double count');

    return count * 2;
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <p>Double Count: {doubleCount}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setOtherState(!otherState)}>Toggle Other
State</button>
    </div>
  );
}

export default ExpensiveCalculation;
```

- Link tham khảo: https://alexsidorenko.com/blog/react-render-always-rerenders/

3. Giới thiệu React.memo

3.1. Lý thuyết

React.memo là gì?

React.memo là một higher-order component (HOC) trong React, được sử dụng để tối ưu hóa hiệu suất bằng cách memoize (ghi nhớ) kết quả của một component và tránh việc re-render không cần thiết. Khi nào sử dụng React.memo?

Sử dụng React.memo khi bạn muốn tránh re-render cho một component function, đặc biệt là trong các trường hợp component nhận các props không thay đổi thường xuyên.

3.2. Ví dụ

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

// Component con, không sử dụng React.memo
```

```
const ChildComponent = ({ name }) => {
  console.log('ChildComponent re-rendered');
  return <p>Hello, {name}!</p>;
};

// Component cha
const ParentComponent = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <ChildComponent name="Alice" />
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default ParentComponent;
```

Trong ví dụ trên, ChildComponent không sử dụng React.memo, do đó mỗi lần ParentComponent re-render, ChildComponent cũng sẽ re-render, dù props của nó không thay đổi.

3.3. Áp dụng React.memo

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

// Component con, sử dụng React.memo
const ChildComponent = React.memo(({ name }) => {
  console.log('ChildComponent re-rendered');
  return <p>Hello, {name}!</p>;
});

// Component cha
const ParentComponent = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <ChildComponent name="Alice" />
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default ParentComponent;
```

> Khi sử dụng React.memo cho ChildComponent, component chỉ được re-render lại khi props của nó thay đổi, giúp tối ưu hóa hiệu suất của ứng dụng.

4. Giới thiệu useCallback

4.1. Lý thuyết

useCallback:

useCallback là một hook giúp bạn ghi nhớ (memoize) một hàm callback chỉ khi một trong những dependencies thay đổi. Điều này hữu ích khi bạn truyền callback vào các component con để tránh re-render không cần thiết.

4.2. Ví dụ

Ví dụ: Component cha sử dụng useCallback để memoize hàm increment và truyền nó xuống component con.

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

function Child({ onIncrement }) {
  console.log('Child re-rendered');
  return (
    <div>
      <button onClick={onIncrement}>Increment</button>
    </div>
  );
}

function Parent() {
  const [count, setCount] = useState(0);
  const [otherState, setOtherState] = useState(false);

  const increment = useCallback(() => {
    setCount(count + 1);
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <Child onIncrement={increment} />
      <button onClick={() => setOtherState(!otherState)}>Toggle Other
State</button>
    </div>
  );
}

export default Parent;
```

Kết luận

> Hiểu lý do tại sao React re-render giúp bạn tối ưu hóa hiệu suất ứng dụng. useMemo và useCallback là hai hook hữu ích để tối ưu hóa các tính toán đắt đỏ và hàm callback trong React, giúp tránh re-

> render không cần thiết.

Bài tập

- https://codesandbox.io/p/sandbox/buoi08-bt3-memo-w7wn9d
- https://codesandbox.io/p/sandbox/buoi08-bt4-usememo-rqjnjf
- https://codesandbox.io/p/sandbox/buoi09-bt3-usecallback-njdlv5