React Function Component Lifecycle Methods

Function components don't have lifecycle methods like class components, but React provides the useEffect hook to handle side effects and mimic lifecycle behavior.

useEffect Hook

The **useEffect** hook lets you perform side effects in function components. It's equivalent to **componentDidMount, componentDidUpdate, and componentWillUnmount** in class components.

The useEffect hook is a powerful and versatile tool in React function components for **handling side effects, such as fetching data, subscribing to events, or manipulating the DOM.**

javascript Copy code

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

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

  useEffect(() => {
    // This code runs after the component renders
    document.title = `You clicked ${count} times`;

    // Cleanup function
    return () => {
      // This code runs before the component unmounts or before the next effect
      console.log('Cleanup code');
    };
  }, [count]); // Only re-run the effect if count changes

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

Fetching Data from APIs

To fetch data from APIs in a function component, you typically use the useEffect hook along with **fetch** or a library like Axios.

Here's an example using **fetch**:

javascript Copy code

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

function DataFetchingComponent() {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.example.com/data');
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        const result = await response.json();
        setData(result);
      } catch (error) {
        setError(error);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []); // Empty dependency array means this effect runs once after the initial render

  if (loading) {
    return <div>Loading...</div>;
  }

  if (error) {
    return <div>Error: {error.message}</div>;
  }

  return (
    <div>
      {data.map(item => (
        <div key={item.id}>{item.name}</div>
      ))}
    </div>
  );
}
```

Key Points

1. **useEffect Dependencies**: The second argument to useEffect is an array of dependencies. The effect runs whenever one of these dependencies changes. If the array is empty, the effect runs only once, after the initial render.
2. **Cleanup Function**: The cleanup function inside useEffect runs before the component unmounts or before the next effect. It's useful for cleaning up subscriptions, timers, or other resources to avoid memory leaks.
3. **Fetching Data**: When fetching data, handle loading and error states to provide feedback to the user. Use async/await for cleaner asynchronous code.