DECEMBER 4, 2023 / #REACT

How to Use React Hooks – useEffect, useState, and useContext Code Examples

Joan Ayebola

React is a powerful JavaScript library for building user interfaces. And it has undergone significant changes over the years.

One of the most noteworthy additions is the introduction of hooks, which [illegible] more concise and expressive way to handle state and lifecycle methods. Hooks allow you to use state and other React features without writing a class.

What are React Hooks?

React hooks are functions that enable functional components to use state and lifecycle features that were previously only available in class components. They were introduced in React 16.8 to provide a more consistent way to manage stateful logic in functional components.

How to Use the `useState` Hook

The `useState` hook is perhaps the most basic and essential hook in React. It [illegible] Hooks allow you to use state and other React features without writing a class.

What are React Hooks?

React hooks are functions that enable functional components to use state and lifecycle features that were previously only available in class components. They were introduced in React 16.8 to provide a more consistent way to manage stateful logic in functional components.

How to Use the `useState` Hook

The `useState` hook is perhaps the most basic and essential hook in React. It enables you to add state to your functional components, allowing them to keep track of data that changes over time. Let's dive into how `useState` works with a simple example.

Basic Usage of `useState`

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

const Counter = () => {
  // Declare a state variable named 'count' with an initial value of 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
export default Counter;
```

In this example, we import the `useState` hook from the 'react' library. The `useState` function returns an array with two elements: the current state value (`count`) and a function (`setCount`) to update it. We initialize `count` to 0, and clicking the "Increment" button increases its value.

How to Use Multiple `useState` Hooks

You can use the `useState` hook multiple times in a single component to manage different pieces of state independently. Let's modify our `Counter` component to include a second piece of state.

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

const Counter = () => {
  const [count, setCount] = useState(0);
  const [isEven, setIsEven] = useState(false);

  return (
    <div>
      <p>Count: {count}</p>
      <p>{isEven ? 'Even' : 'Odd'}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setIsEven(!isEven)}>Toggle Even/Odd</button>
    </div>
  );
};

export default Counter;
```

Now, our `Counter` component has two independent pieces of state: `count` and `isEven`. Clicking the "Toggle Even/Odd" button will switch the value of `isEven`.

How to Use the `useEffect` Hook

The `useEffect` hook is used to perform side effects in your functional components, such as fetching data, subscribing to external events, or manually changing the DOM. It combines the functionality of `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in class components.

Basic Usage of `useEffect`

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

const DataFetcher = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Fetch data from an API
    fetch('https://api.example.com/data')
      .then((response) => response.json())
      .then((result) => setData(result))
      .catch((error) => console.error('Error fetching data:', error));
  }, []); // Empty dependency array means this effect runs once after the initial render

  return (
    <div>
      {data ? (
        <ul>
          {data.map((item) => (
            <li key={item.id}>{item.name}</li>
          ))}
        </ul>
      ) : (
        <p>Loading data...</p>
      )}
    </div>
  );
};

export default DataFetcher;
```

In this example, the `useEffect` hook is used to fetch data from an API when the component mounts. The second argument of `useEffect` is an array of dependencies. If the dependencies change between renders, the effect will run again. An empty array means the effect runs once after the initial render.

Cleaning Up in `useEffect`

Sometimes, side effects need to be cleaned up, especially when dealing with subscriptions or timers to prevent memory leaks. The `useEffect` hook can return a cleanup function that will be executed when the component unmounts.

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

const Timer = () => {
  const [seconds, setSeconds] = useState(0);

  useEffect(() => {
    const intervalId = setInterval(() => {
      setSeconds((prevSeconds) => prevSeconds + 1);
    }, 1000);

    // Cleanup function to clear the interval when the component unmounts
    return () => clearInterval(intervalId);
  }, []); // Empty dependency array for initial setup only

  return <p>Seconds: {seconds}</p>;
};

export default Timer;
```

In this example, the `setInterval` function is used to update the `seconds` state every second. The cleanup function returned by `useEffect` clears the interval when the component is unmounted.

How to Use the `useContext` Hook

The `useContext` hook is used to consume values from a React context. Context provides a way to pass data through the component tree without having to pass props manually at every level. Let's explore how `useContext` works with a simple example.

How to Create a Context

First, let's create a context to hold a user's authentication status.

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```


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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
```

```
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```


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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);
```

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

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const login = () => {
    setIsAuthenticated(true);
  };

  const logout = () => {
    setIsAuthenticated(false);
  };

  return (
    <AuthContext.Provider value={{ isAuthenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
```