React - Hooks

Topics : React JS
Written on January 03, 2024

React Hooks are functions that allow you to use state and other React features in functional components. They were introduced in React version 16.8 to provide a more convenient way to work with state and side effects in functional components, eliminating the need for class components in many cases.

Here are some commonly used React Hooks:

1. useState:

`useState` allows you to add state to functional components. It returns an array with two elements: the current state value and a function to update the state.

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

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

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

2. useEffect:

`useEffect` is used for side effects in functional components. It can replace lifecycle methods in class components.

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

function Example() {
const [data, setData] = useState([]);

useEffect(() => {
// Fetch data or perform side effects here
// This function will run after every render
fetchData();
}, []); // The empty dependency array means this effect runs once after the initial render
```

```
return (
<div>
{/* Render UI based on the data */}
</div>
);
}
```

3. useContext:

`useContext` allows you to subscribe to React context without introducing nesting.

```
import React, { useContext } from 'react';
import MyContext from './MyContext';

function Example() {
const contextValue = useContext(MyContext);

return (
<div>
{/* Use the context value */}
</div>
);
}
```

4. useReducer:

`useReducer` is a hook for managing more complex state logic. It is often preferable to `useState` when the next state depends on the previous one.

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

const initialState = { count: 0 };

function reducer(state, action) {
switch (action.type) {
case 'increment':
return { count: state.count + 1 };
default:
return state;
}
}

function Example() {
const [state, dispatch] = useReducer(reducer, initialState);

return (
<div>
<p>Count: {state.count}</p>
<button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
</div>
);
}
```

5. useCallback and useMemo:

`useCallback` and `useMemo` are used to optimize performance by memoizing functions and values.

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

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

const increment = useCallback(() => {
setCount(count + 1);
}, [count]); // Re-created only if count changes

const doubledValue = useMemo(() => count * 2, [count]); // Recalculated only if count changes

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