

Frontend Development

useReducer Hook:

The useReducer hook in React is a more powerful alternative to useState when dealing with complex state logic in your components. It accepts a reducer function and an initial state, and returns the current state and a dispatch function to update the state based on actions dispatched to the reducer.

The useReducer hook is a function in React that is used for state management. It is an alternative to the useState hook when the state logic is more complex and involves multiple sub-values or when the next state depends on the previous one. The useReducer hook is inspired by the reducer functions in JavaScript, which are commonly used with arrays to reduce them to a single value.

const [state, dispatch] = useReducer(reducer, initialArgument, init);

state: The Current State Value.

dispatch: A Function That Allows You To Dispatch Actions To Modify The State.

Reducer: A Function That Takes The Current State & An Action As Arguments & Returns The New State.

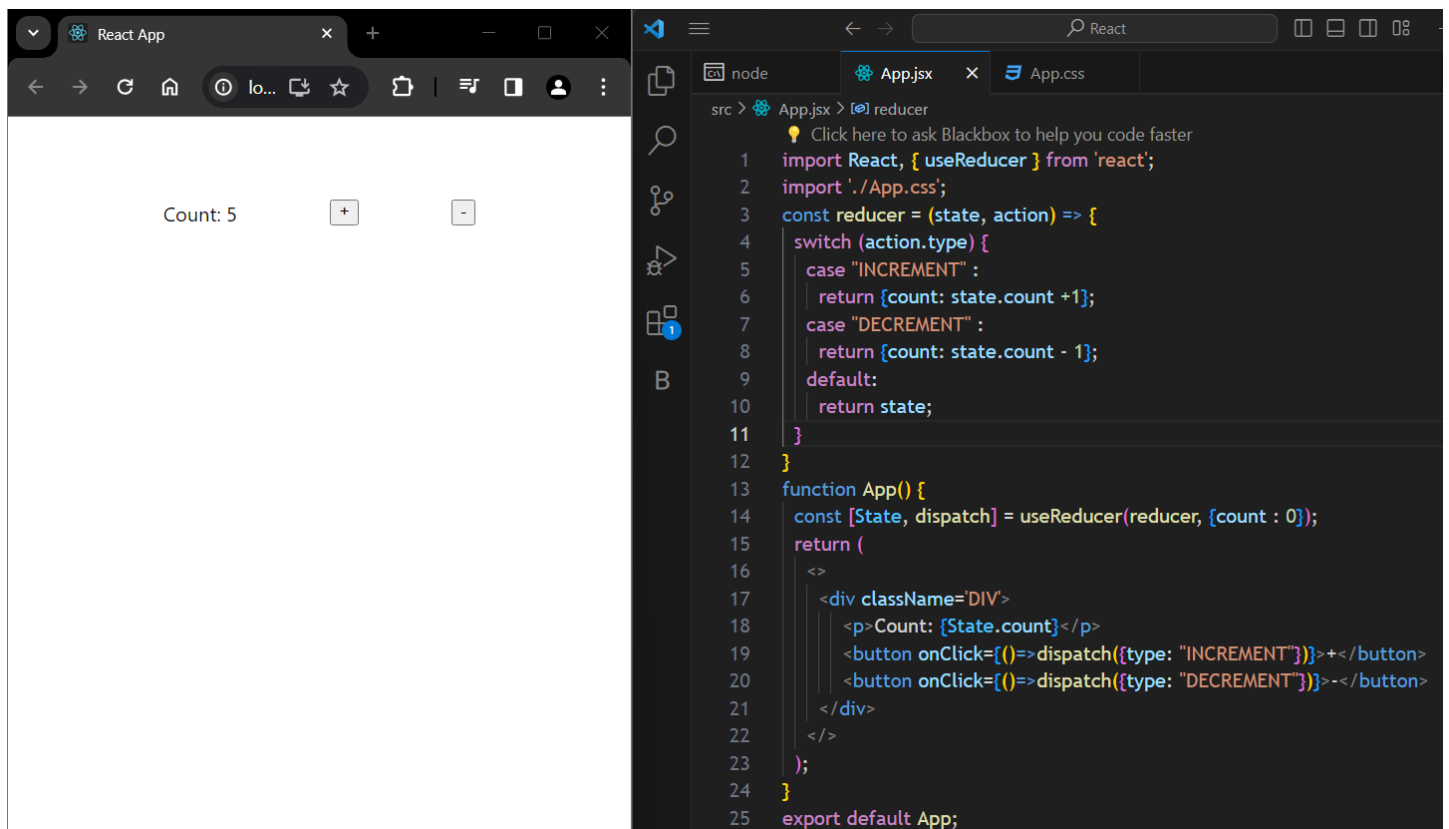
initialArgument: The Initial State Value Or A Function That Returns The Initial State. The Argument Is Optional.

Init: A Function That Returns The Initial State. This Argument Is Optional.

How useReducer Function Works:

useReducer takes a reducer function and an initial state as arguments. When the dispatch function is called with an action, useReducer passes the current state and the action to the reducer function.

The reducer function then calculates the new state based on the current state and the action. useReducer returns the new state, which can be accessed through the state variable, and a dispatch function that can be used to update the state.



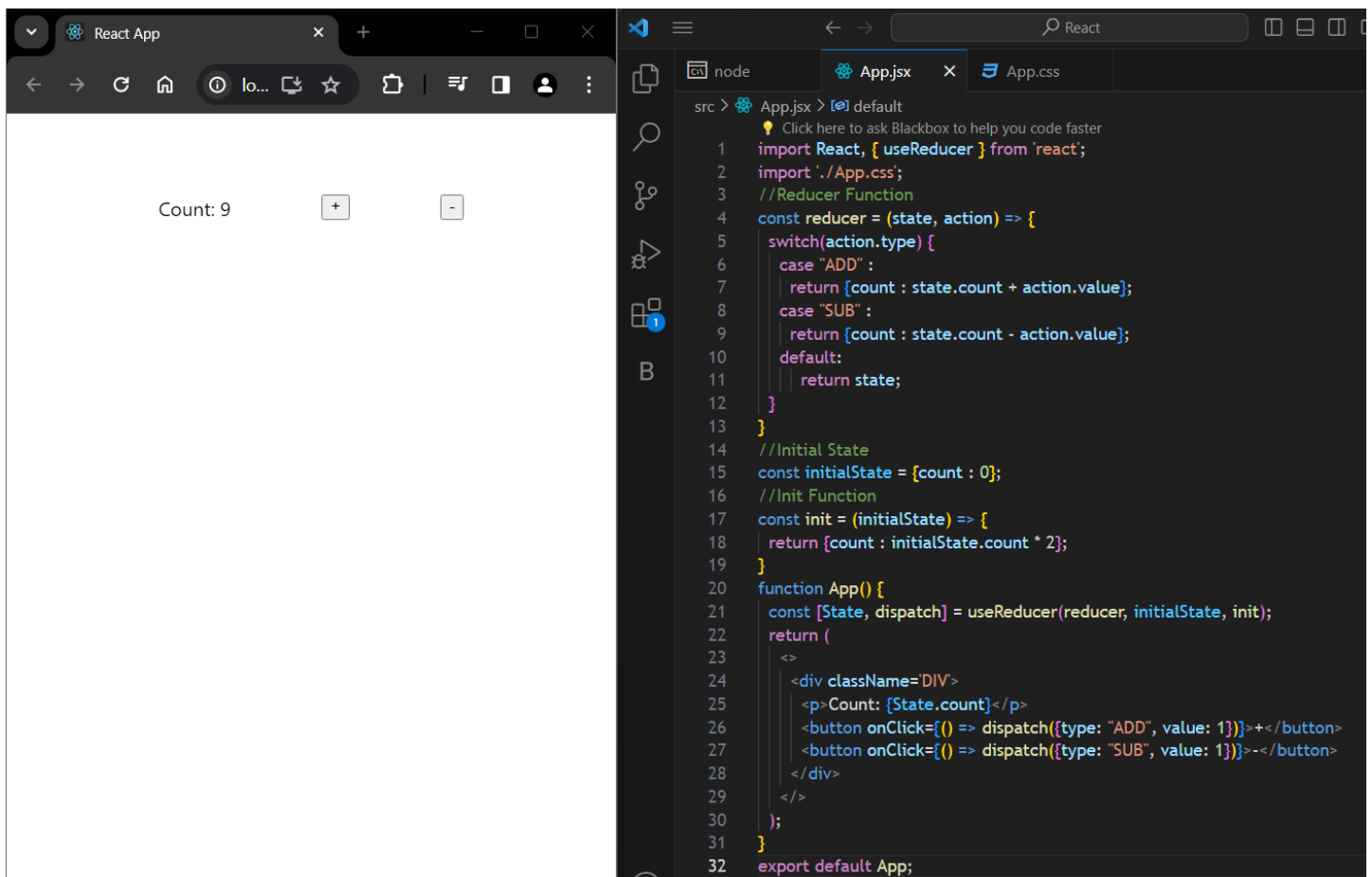
1. We define a reducer function that takes the current state and an action, then returns the new state based on the action.
2. In the Counter component, we use `useReducer` with the reducer function and an initial state `{ count: 0 }`.
3. We use the `dispatch` function to increment or decrement the count value in the state.

Why useReducer in this above example:

1. We use `useReducer` because the state logic involves incrementing and decrementing a count value, which is a bit more complex than a simple state update.
2. The reducer function helps manage the state changes in a more structured way compared to handling the logic directly in the component.
3. `useReducer` allows us to separate the state management logic from the component rendering logic, making the code easier to understand and maintain.

Code Flow:

1. The initial state is `{ count: 0 }`. The Counter component renders, displaying the count value and two buttons for incrementing and decrementing.
2. When the "Increment" button is clicked, it calls `dispatch({ type: 'INCREMENT' })`. `useReducer` calls the reducer function with the current state `{ count: 0 }` and the action `{ type: 'INCREMENT' }`.
3. The reducer function increments the count value and returns `{ count: 1 }`. `useReducer` updates the state to `{ count: 1 }` and re-renders the component.
4. The updated count value is displayed, and the process repeats for decrementing and subsequent increments.



The image shows a side-by-side view of a web browser and a code editor. The browser on the left displays a simple counter interface with the text "Count: 9" and two buttons, "+" and "-", for incrementing and decrementing the count. The code editor on the right shows the source code for the application. It includes the `App.jsx` file with the following code:

```
1 import React, { useReducer } from 'react';
2 import './App.css';
3 //Reducer Function
4 const reducer = (state, action) => {
5   switch(action.type) {
6     case "ADD":
7       return {count : state.count + action.value};
8     case "SUB":
9       return {count : state.count - action.value};
10    default:
11      return state;
12   }
13 }
14 //Initial State
15 const initialState = {count : 0};
16 //Init Function
17 const init = (initialState) => {
18   return {count : initialState.count * 2};
19 }
20 function App() {
21   const [State, dispatch] = useReducer(reducer, initialState, init);
22   return (
23     <>
24     <div className="DIV">
25       <p>Count: {State.count}</p>
26       <button onClick={() => dispatch({type: "ADD", value: 1})}>+</button>
27       <button onClick={() => dispatch({type: "SUB", value: 1})}>-</button>
28     </div>
29     </>
30   );
31 }
32 export default App;
```

1. `useReducer` is called with `reducer`, `initialState`, and `init`, and returns `state` and `dispatch`. The `init` function is used to initialize the state. In this case, it doubles the initial count value.
2. The reducer function handles `ADD` and `SUBTRACT` actions, updating the count value based on the action type and value.
3. The `Counter` component renders a count value from `state` and two buttons that dispatch `ADD` and `SUBTRACT` actions with a value of 1.
4. When a button is clicked, `dispatch` is called with the corresponding action type and value, which updates the state using the reducer. The component re-renders with the updated count value.