REACT useReducer HOOK

```
const Counter = () => {
  // Using useReducer with the reducer functio
  const [state, dispatch] = useReducer(reducer

  return (
    <div>
```


What is useReducer ?

useReducer is a React hook used to manage more complex state logic. It provides a way to update the state based on specific actions, allowing you to handle scenarios where the state changes in multiple ways or relies on previous state values. It returns a **state object** and a **dispatch** function, which sends actions to a **reducer function** to update the state.

When to use useReducer ?

1. **State logic is complex:** If you have multiple state values that need to be updated in response to different actions, or if state changes depend on the previous state.

2. State needs to be more predictable: useReducer helps manage state transitions in a way that's organized, predictable, and easy to follow, especially when managing more intricate state updates.

3. Multiple actions are needed: When your state needs to change in different ways depending on different actions (e.g., adding, removing, or updating values).

1. Basic Example: Counter

Counter.jsx

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

// Reducer function that tells how to update the state
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };  // Increase count
    case 'decrement':
      return { count: state.count - 1 };  // Decrease count
    default:
      return state;  // Return the state as is
  }
};

const Counter = () => {
  // Using useReducer with the reducer function and initial state
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>  {/* Display the current count */}
      {/* Increment the count */}
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      {/* Decrement the count */}
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
};

export default Counter;
```

2. What is the reducer Function?

The reducer function decides how the state changes. It takes:

Counter.jsx

```jsx
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };  // Increase the count
    case 'decrement':
      return { count: state.count - 1 };  // Decrease the count
    default:
      return state;  // If the action is unknown return the current state
  }
};
```

Components of a Reducer Function:

- **State:**
  - Represents the current state of your application.
  - For example, if your app tracks a counter, the state could be { count: 0 }.
- **Action:**
  - An object that describes what you want to do. It typically has a type property, which is a string (e.g., { type: 'increment' }).
  - The action can sometimes include additional data (payload) required to update the state, such as { type: 'setCount', payload: 10 }.
- **Return Value:**
  - The reducer function processes the action and returns a new state based on the current state.

3. Dispatching Actions

The **dispatch** function is used to send an action to the reducer. Actions are just objects with a **type** property that tells the reducer what to do.

Counter.jsx

```
dispatch({ type: 'increment' });  // Increases the count by 1
dispatch({ type: 'decrement' });  // Decreases the count by 1
```

- **Action Object**:
  - The type property in the action object specifies the operation to perform.
  - For example:
    - { type: 'increment' } tells the reducer to increase the count.
    - { type: 'decrement' } tells the reducer to decrease the count.

- **Flow of Dispatch**:
  - When dispatch is called, it sends the action to the reducer.
  - The reducer then matches the action.type with one of its cases and updates the state accordingly.

- **Reducer Behavior**:
  - If dispatch({ type: 'increment' }) is called, the reducer executes the code in the case 'increment': block, returning a new state with the count increased.
  - Similarly, dispatch({ type: 'decrement' }) decreases the count by executing the case 'decrement': block.

Hopefully You Found It Usefull!

"Be sure to save this post so you can come back to it later

like

Comment

Share

Slim toumi

Slim toumi