Hema Coding School

YouTube Link: https://www.youtube.com/@HemaCodingSchool

Introduction

- useReducer is a React hook used for managing complex state logic in functional components.
- It is an alternative to useState and is particularly useful for handling state transitions in a more structured and predictable manner.

Basic Usage

1. **Initialization**:
   - The useReducer hook requires two arguments: a reducer function and an initial state.
   - It returns an array with two elements: the current state and the dispatch function.
2. **Reducer Function**:
   - The reducer function takes two parameters: the current state and an action.
   - It returns a new state based on the action type and payload.
3. **Dispatch Function**:
   - The dispatch function is used to send actions to the reducer.
   - Actions are plain JavaScript objects that typically have a type property and optionally other properties (payload)

```
let[state, dispatch] = useReducer(reducer, intialValue)
```

When to Use useReducer

1. **Complex State Logic**:
   - When the state logic is complex and involves multiple sub-values.
   - When the next state depends on the previous state.
2. **Related State Updates**:
   - When multiple state variables need to be updated together.
3. **Debugging and Testing**:
   - When you want a more predictable state transition that is easier to debug and test.

```
import React,{useState,useReducer} from 'react';
import '../../Component/ClassBased.css'


function UseReducer() {
let intialValue = 0


let reducer = (state, action)=>{
  console.log(action) // action =  {type:'inc'}
  console.log(action.type) // inc

  // if(action.type == 'Inc'){
  //   return state + 1;
```

```
 // }
 switch(action.type){
   case 'Inc' :
     return state + 1;
     case 'Sub' :
     return state - 1;
     case 'Mul' :
     return state * 2;
     case 'Div' :
     return state / 2;
 }

}

let[state, dispatch] = useReducer(reducer, intialValue)

 return (
   <div className="train-info">
     <h1>{state}</h1>
   <button onClick={()=>dispatch({type:'Inc'})}>Count ++ </button>
   <button onClick={()=>dispatch({type:'Sub'})}>Count -- </button>
   <button onClick={()=>dispatch({type:'Mul'})}>Count ** </button>
   <button onClick={()=>dispatch({type:'Div'})}>Count  / </button>
   </div>
 )
}

export default UseReducer;
```

Comparison with useState

- **useState** is simple and ideal for managing individual state variables or simple state logic.
- **useReducer** is better suited for complex state logic or when you need a more structured approach to state management..

Interview Questions

Question 1: What is useReducer and how does it differ from useState?

Answer: useReducer is a React hook used for managing complex state logic. It provides an alternative to useState by allowing you to centralize state updates in a reducer function. While useState is great for simple state management with individual state variables, useReducer is more suited for complex state transitions and when the state logic involves multiple sub-values or needs to be managed in a more predictable manner.

Question 2: Explain the basic syntax of useReducer.

Answer: The basic syntax of useReducer involves defining a reducer function and an initial state, then calling useReducer with these arguments. It returns the current state and a dispatch function.

```
let[state, dispatch] = useReducer(reducer, intialValue)
```

The reducer function takes the current state and an action, and returns the new state based on the action type.

Question 3: When would you choose useReducer over useState?

Answer: You would choose useReducer over useState when:

1. The state logic is complex and involves multiple sub-values.
2. You need to manage state transitions in a more predictable and structured manner.
3. Multiple state variables need to be updated together.
4. You want to improve the readability and maintainability of your code.
5. The next state depends on the previous state.

Question 4: What is the role of the dispatch function in useReducer?

Answer: The dispatch function is used to send actions to the reducer function. These actions typically have a type property that specifies the type of state transition, and can also include additional data (payload) that informs the state update. The dispatch function triggers the reducer to compute the new state based on the current state and the action received

Question 5: How do you handle side effects with useReducer?

Answer: To handle side effects with useReducer, you typically use the useEffect hook. useEffect allows you to perform side effects in response to state changes or other dependencies.

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

const initialState = { data: null, loading: true, error: null };

function reducer(state, action) {
  switch (action.type) {
    case 'fetch_success':
      return { ...state, data: action.payload, loading: false };
    case 'fetch_error':
      return { ...state, error: action.payload, loading: false };
    default:
      throw new Error();
  }
}

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

  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => dispatch({ type: 'fetch_success', payload: data }))
      .catch(error => dispatch({ type: 'fetch_error', payload: error }));
  }, []);

  if (state.loading) return <p>Loading...</p>;
  if (state.error) return <p>Error: {state.error}</p>;
  return <div>Data: {JSON.stringify(state.data)}</div>;
}

export default DataFetcher;
```