useReducer Exercises

ex01 reduce() revision

challenge 1 (3m)

Take an array and calculate the sum of even and odd numbers using reduce.

```
const numList = [1, 3, 55, 22, 44]

function oddAndEvenSumReducer(acc, value) {
  // write your code here
}

const oddAndEvenSum = numList.reduce(oddAndEvenSumReducer)
```

COPY

Solution

```
function oddAndEvenSumReducer(acc, value) {
  if (value % 2 === 0) {
    return { ...acc, even: acc.even + value }
  }
  return { ...acc, odd: acc.odd + value }
}

const initialAccumulator = { odd: 0, even: 0 }

numList.reduce(oddAndEvenSumReducer, initialAccumulator)
```

COPY

challenge 2 (5m)

Modify your `oddAndEvenSumReducer` function to accommodate an array of objects.

The object will have two keys: type and payload.

The `type` will tell you whether the number is even or odd and the `payload` will have the number.

```
const numList = [
  { type: 'odd', payload: 1 },
  { type: 'odd', payload: 3 },
  { type: 'odd', payload: 55 },
  { type: 'even', payload: 22 },
  { type: 'even', payload: 44 },
]
```

COPY

Note:

This means you don't need the odd/even logic in the reducer function anymore.

```
if (value.type === 'even') {
}
```

COPY

Solution

```
const numList = [
  { type: 'odd', payload: 1 },
  { type: 'odd', payload: 3 },
  { type: 'odd', payload: 55 },
  { type: 'even', payload: 22 },
  { type: 'even', payload: 44 },
]

function reducer(state, action) {
  if (action.type === 'even') {
    return { ...state, even: state.even + action.payload }
  }
  return { ...state, odd: state.odd + action.payload }
}

function oddAndEvenSumReducer(acc, value) {
  if (value.type === 'even') {
    return { ...acc, even: acc.even + value.payload }
  }
  return { ...acc, odd: acc.odd + value.payload }
}

// Dry run
// 1 ; acc: { odd: 0, even: 0 }, value: { type: 'odd', payload: 1 }
// 2 ; acc: { odd: 1, even: 0 }, value: { type: 'odd', payload: 3 }
// 3 ; acc: { odd: 4, even: 0 }, value: { type: 'odd', payload: 55 }
// 4 ; acc: { odd: 59, even: 0 }, value: { type: 'even', payload: 22 }
// 5 ; acc: { odd: 59, even: 22 }, value: { type: 'even', payload: 44 }
// 6 { odd: 59, even 66 }

const state = { odd: 0, even: 0 }

numList.reduce(reducer, state)
function counterReducer(state, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: count + 1 }

    case 'DECREMENT':
      return { count: count - 1 }
  }
}

// Dry run  // + - // Define state, action pair              // state
// 1: +  state: { count: 0 }, action: { type: 'INCREMENT' } // { count: 1 }
// 2: +  state: { count: 1 }, action: { type: 'INCREMENT' } // { count: 2 }
// 3: -  state: { count: 2 }, action: { type: 'DECREMENT' } // { count: 1 }
// 4: +  state: { count: 1 }, action: { type: 'INCREMENT' } // { count: 2 }
// 5: +  state: { count: 2 }, action: { type: 'INCREMENT' } // { count: 3 }
```

COPY

challenge 3 (4m)

Now, remove if/else and use a switch statement instead.

See how we write switch in JavaScript:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch

Solution

```
const numList = [
  { type: 'odd', payload: 1 },
  { type: 'odd', payload: 3 },
  { type: 'odd', payload: 55 },
  { type: 'even', payload: 22 },
  { type: 'even', payload: 44 },
]

function oddAndEvenSumReducer(acc, value) {
  switch (value.type) {
    case 'even':
      return { ...acc, even: acc.even + value.payload }
    case 'odd':
      return { ...acc, odd: acc.odd + value.payload }
    default:
      return acc
  }
}

const initialAccumulator = { odd: 0, even: 0 }

numList.reduce(oddAndEvenSumReducer, initialAccumulator)
```

COPY

🔥 CONGRATULATIONS! You have just written your first reducer.

Now, let's just learn how to put it in React and later we'll learn how to put it in Redux.

But the basics are this and this only.

ex02 the most basic reducer (5m)

understanding

Both `useState` and `useReducer` return a current state value and a function to update the state.

However, `useState` uses a function that directly sets a new state value, while `useReducer` uses a dispatch function that takes an action and uses a reducer function to calculate and set the new state value based on the current state and the action.

When the button is clicked, the `andFunctionToSetState` function is called, which in turn dispatches an action to the `reducerFunc`.

The `reducerFunc` logs "being called..." to the console, indicating that it has been called due to the state update. This way, every time the button is clicked, the `reducerFunc` is called, and the message "being called..." is logged to the console.

challenge

Let's see how the wiring works in React.

Use `useReducer()` to `console.log` every time a button is clicked.

solution

https://codesandbox.io/s/ex02-the-most-basic-reducer-ihxfvl

class notes

1. https://codesandbox.io/s/wizardly-tristan-xrrvzw

ex03 counter with reducer

challenge 00 (5m)

Create a counter with increment + and useReducer.

https://codesandbox.io/s/ex02-the-most-basic-reducer-ihxfvl

steps

Step 1: Use `useReducer` hook with the default state.

```
const [state, dispatch] = useReducer(reducerFunc, { counter: 0 })
```

COPY

Step 2: Define the Reducer Function

```
function reducerFunc(state, action) {
  return { ...state, counter: state.counter + action.payload }
```

```
}
```

COPY

Step 3: Wire it with UI

```
<h2>{state.counter}</h2>
<button onClick={() => dispatch({ type: "increment", payload: 1 })}>
      +{" "}
</button>
```

COPY

solution

https://codesandbox.io/s/ex03-challenge-00-solution-6c6eos

understanding

Inside the `reducerFunc`, the current state is spread into a new object using the spread syntax. Then, the `action.payload` value is added to the `counter` property of the state object, which is used to calculate the new state value.

When the button is clicked, the `dispatch` function is called with the action object, and the `reducerFunc` is executed with the current state and the action.

The `counter` value in the state is incremented by the `action.payload` value, and a new state object is returned.

This triggers a re-render of the component with the updated state and the updated `state.counter` value is displayed in the UI.

```
// Internal representation of useReducer
function useReducerTanayImpl(reducerFunc, initialState) {
  function dispatch(action) {
    reducerFunc(initialState, action)
  }
  return [initialState, dispatch]
}
```

COPY

challenge 01 (5m)

Create a counter with `+` and `-` button.

Hint: You only need to pass the right `type` as there's no need to pass a `payload`

https://codesandbox.io/s/ex03-challenge-00-solution-6c6eos

steps

Step 1: Use `useReducer` hook with the default state.

```
const [state, dispatch] = useReducer(counterReducer, { count: 0 })
```

COPY

Step 2: Define a Reducer Function.

This function will handle state updates based on different action types. This function uses a switch statement and is defined with cases for "INCREMENT" and "DECREMENT" actions, which update the state by incrementing or decrementing the `count` value, respectively.

```
const counterReducer = (acc, value) => {
  switch (value.type) {
    case 'INCREMENT':
      return { ...acc, count: acc.count + 1 }
    case 'DECREMENT':
      return { ...acc, count: acc.count - 1 }
    default:
      console.log('Something went wrong')
      break
  }
  return acc
}
```

COPY

Step 3: Dispatching Actions:

When the "+" button is clicked, the `dispatch` function is called with the `{ type: "INCREMENT" }` action object. When the "-" button is clicked, the `dispatch` function is called with the `{ type: "DECREMENT" }` action object. These actions are dispatched to the `counterReducer` function, which updates the state accordingly.

```
<button onClick={() => dispatch({ type: "INCREMENT" })}>+</button>
<button onClick={() => dispatch({ type: "DECREMENT" })}>-</button>
```

COPY

Step 4: State Update

After the state is updated by the `counterReducer` function, React re-renders the component with the updated `state.count` value.

```
<p>Count : {state.count}</p>
```

COPY

solution

https://codesandbox.io/s/ex03-challenge-01-solution-uxwqmb

understanding

The **`counterReducer`**function is defined to handle state updates based on different action types. It takes two arguments - `acc` (which represents the current state) and `value` (which represents the action dispatched).

When the "+" button is clicked, the `dispatch` function is called with the `{ type: "INCREMENT" }` action object. The `counterReducer` function is executed with the current state (`acc`) and the action (`value`), and a new state object is returned with the `count` value incremented by 1.

Similarly, when the "-" button is clicked, the `dispatch` function is called with the `{ type: "DECREMENT" }` action object, and the `counterReducer` function updates the state with the `count` value decremented by 1.