**useReducer 이해하기**

-이제까지 우리는 상태를 관리할 때 useState를 사용하였다.

-하지만, useReducer를 사용하여 상태를 관리할 수도 있다.

-useReducer를 사용하면 컴포넌트의 상태 업데이트 로직을 컴포넌트에서 분리시킬 수 있다.

-상태업데이트 로직을 컴포넌트 바깥에 작성할 수도 있고, 다른 파일에 작성 후 불러서 사용할 수 있음.

-reducer : 현재 상태와 액션 객체를 파라미터로 받아와서 새로운 상태로 반환해주는 함수이다.

function reducer(state, action) {

//새로운 상태를 만드는 로직

// const nextState = …

return nextState;

}

-여기서 action은 업데이트를 위한 정보를 가지고 있음. 주로 type값을 지닌 객체 형태로 사용됨.

-useReducer는 첫번째 파라미터엔 reducer 함수를 넣고, 두번째 파라미터엔 초기상태를 넣는다

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

//state는 우리가 앞으로 사용하는 상태를 가리키고

//dispatch는 액션을 발생시키는 함수이다.

//dispatch({ type: ‘INCREMENT’ }) 로 사용함.

**useReducer vs useState**

-컴포넌트에서 관리하는 값이 적거나, 단순한 숫자, 문자열 또는 boolean 값이면 useState가 편하다

-컴포넌트에서 관리하는 값이 많거나, 그 값이 복잡하면 useReducer가 더 편하다

**Counter.js 예제**

//Counter.js

import React, { useReducer } from ‘react’;

function reducer(state, action) {

switch (action.type) {

case ‘INCREMENT’:

return state + 1;

case ‘DECREMENT’:

return state -1;

default:

return state;

}

}

function Counter() {

const [number, dispatch] = useReducer(reducer, 0);

const onIncrease = () => {

dispatch({ type: ‘INCREMENT’ });

});

const onDecrease = () => {

dispatch({ type: ‘DECREMENT’ });

});

…

}

**App.js 예제**

import React, {useRef, useReducer, useMemo, useCallback } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function countActiveUsers(users) {

console.log('활성 사용자 수를 세는 중...');

return users.filter(user => user.active).length;

}

const initialState = {

inputs: {

username:'',

email:''

},

users: [

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com',

active: true,

},

{

id: 2,

username: 'tester',

email: 'tester@example.com',

active:true,

},

{

id: 3,

username: 'liz',

email: 'liz@example.com',

active:true,

}

]

};

function reducer(state, action) {

switch (action.type) {

case 'CHANGE\_INPUT':

return {

...state,

inputs: {

...state.inputs, //reducer함수에서 새로운 상태를 만들 때에는 불변성을

[action.name]: action.value //지켜주어야하므로 spread 연산자를 사용함

}

};

case 'CREATE\_USER':

return {

inputs: initialState.inputs, //inputs은 initialState.inputs으로 초기화하고

users: state.users.concat(action.user) //users는 concat으로 추가시켜줌.

};

case 'TOGGLE\_USER':

return {

...state,

users: state.users.map(user =>

user.id === action.id ? { ...user, active: !user.active} : user

)

};

case 'REMOVE\_USER':

return {

...state,

users: state.users.filter(user => user.id !== action.id)

};

default:

return state;

}

}

function App() {

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

const nextId = useRef(4);

const { users } = state;

const { username, email } = state.inputs;

const onChange = useCallback(e=> {

const { name, value } = e.target;

dispatch({

type: 'CHANGE\_INPUT',

name,

value

});

}, []);

const onCreate = useCallback(() => {

dispatch({

type: 'CREATE\_USER',

user: {

id: nextId.current,

username,

email

}

});

nextId.current += 1;

}, [username, email]);

const onToggle = useCallback(id => {

dispatch({

type: 'TOGGLE\_USER',

id

});

}, []);

const onRemove = useCallback(id => {

dispatch({

type: 'REMOVE\_USER',

id

});

}, []);

const count = useMemo(() => countActiveUsers(users), [users]);

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} onToggle={onToggle} onRemove={onRemove}/>

<div> 활성 사용자 수 : {count}</div>

</>

);

}

export default App;