[할 일 관리] 앱 업그레이드

이번 포스트에서는 useReducer 훅을 사용하여 [할 일 관리] 앱을 업그레이드합니다.

useReducer는 상태 관리 로직을 컴포넌트 외부로 분리할 수 있어 상태가 복잡해질 때 유용합니다.

기존의 useState를 useReducer로 대체해보겠습니다.

1. useState를 useReducer로 바꾸기

먼저, App.js 파일에서 useState를 useReducer로 대체합니다.

코드 변경

import { useReducer, useRef } from "react";

*// 상태 변화 로직*

function reducer(state, action) {

switch (action.type) {

case "CREATE":

return [action.newItem, ...state];

case "UPDATE":

return state.map((it) =>

it.id === action.targetId ? { ...it, isDone: !it.isDone } : it

);

case "DELETE":

return state.filter((it) => it.id !== action.targetId);

default:

return state;

}

}

function App() {

const [todo, dispatch] = useReducer(reducer, mockTodo);

const idRef = useRef(3);

const onCreate = (content) => {

dispatch({

type: "CREATE",

newItem: {

id: idRef.current,

content,

isDone: false,

createdDate: new Date().getTime(),

},

});

idRef.current += 1;

};

const onUpdate = (targetId) => {

dispatch({ type: "UPDATE", targetId });

};

const onDelete = (targetId) => {

dispatch({ type: "DELETE", targetId });

};

return (

<div className="App">

<Header />

<TodoEditor onCreate={onCreate} />

<TodoList todo={todo} onUpdate={onUpdate} onDelete={onDelete} />

</div>

);

}

export default App;

설명

1. useReducer를 react 라이브러리에서 불러옵니다. useState 대신 useReducer를 사용할 예정이므로 기존 useState 코드는 삭제합니다.
2. reducer 함수는 상태 변화 로직을 처리합니다. 여기서는 CREATE, UPDATE, DELETE 타입에 따라 상태를 변화시킵니다.
3. useReducer를 사용하여 todo 상태와 dispatch 함수를 초기화합니다. dispatch 함수는 상태 변화를 촉발합니다.
4. onCreate, onUpdate, onDelete 함수에서 **dispatch**를 호출하여 **상태 변화를 처리**합니다.

2. Create: 할 일 아이템 추가하기

코드 변경

const onCreate = (content) => {

dispatch({

type: "CREATE",

newItem: {

id: idRef.current,

content,

isDone: false,

createdDate: new Date().getTime(),

},

});

idRef.current += 1;

};

설명

* dispatch 함수는 type을 CREATE로 설정하고, 새 할 일 아이템을 newItem 속성에 저장하여 호출합니다.
* reducer 함수에서 CREATE 타입을 처리하여 새 아이템을 기존 상태에 추가합니다.

3. Update: 할 일 아이템 수정하기

코드 변경

const onUpdate = (targetId) => {

dispatch({ type: "UPDATE", targetId });

};

설명

* dispatch 함수는 type을 UPDATE로 설정하고, targetId 속성에 수정할 아이템의 id를 저장하여 호출합니다.
* reducer 함수에서 UPDATE 타입을 처리하여 해당 아이템의 isDone 상태를 토글합니다.

4. Delete: 할 일 삭제하기

코드 변경

const onDelete = (targetId) => {

dispatch({ type: "DELETE", targetId });

};

설명

* dispatch 함수는 type을 DELETE로 설정하고, targetId 속성에 삭제할 아이템의 id를 저장하여 호출합니다.
* reducer 함수에서 DELETE 타입을 처리하여 해당 아이템을 상태에서 제거합니다.

전체 코드

import { useReducer, useRef } from "react";

import Header from "./component/Header";

import TodoEditor from "./component/TodoEditor";

import TodoList from "./component/TodoList";

import mockTodo from "./mockTodo";

function reducer(state, action) {

switch (action.type) {

case "CREATE":

return [action.newItem, ...state];

case "UPDATE":

return state.map((it) =>

it.id === action.targetId ? { ...it, isDone: !it.isDone } : it

);

case "DELETE":

return state.filter((it) => it.id !== action.targetId);

default:

return state;

}

}

function App() {

const [todo, dispatch] = useReducer(reducer, mockTodo);

const idRef = useRef(3);

const onCreate = (content) => {

dispatch({

type: "CREATE",

newItem: {

id: idRef.current,

content,

isDone: false,

createdDate: new Date().getTime(),

},

});

idRef.current += 1;

};

const onUpdate = (targetId) => {

dispatch({ type: "UPDATE", targetId });

};

const onDelete = (targetId) => {

dispatch({ type: "DELETE", targetId });

};

return (

<div className="App">

<Header />

<TodoEditor onCreate={onCreate} />

<TodoList todo={todo} onUpdate={onUpdate} onDelete={onDelete} />

</div>

);

}

export default App;

이번 포스트에서는 useReducer를 사용하여 [할 일 관리] 앱을 업그레이드했습니다.

useReducer를 이용하면 상태 변화 로직을 컴포넌트 외부로 분리할 수 있어 컴포넌트 코드가 더 간결해집니다.

앞으로 더 복잡한 상태 관리를 할 때 유용하게 사용할 수 있습니다.