**리액트 프로젝트 안에서 어떠한 값을 전역적으로 사용하는 것에 대한 챕터**

1. 기존 UserList 컴포넌트의 문제점

function UserList({ users, onRemove, onToggle }) {

return (

<div>

{users.map(user => (

<User

user={user}

key={user.id}

onRemove={onRemove}

onToggle={onToggle}

/>

))}

</div>

);

}

* UserList 는 지금 user로 onRemove와 onToggle을 전달하기 위한 중간 다리 역할만을 수행하는 중
* 지금처럼 특정 함수를 특정 컴포넌트를 거쳐서 원하는 컴포넌트에게 전달하는 작업은 리액트로 개발하다보면 자주 발생 가능.
* 한 개 정도는 상관없지만 3~4개 이상 거쳐갈 시 큰 불편함이 된다.
* context API를 사용하면 프로젝트 안에서 전역적으로 사용할 수 있는 값을 관리할 수 있다.
* “상태”가 아닌 “값”이다. 꼭 상태뿐만이 아니라 함수, 외부라이브러리, DOM를 가리킬 수도 있다.

1. ContextAPI 만드는법
   1. Context 만들기 : const UserDispatch = React.createContext(null);

createContext() 파라미터 : Context의 기본값 설정 가능. null은 따로 지정하지 않을 때

* 1. Context 안에 Provider 컴포넌트로 Context의 값 정하기. value 로 값을 설정해주면 된다.

<UserDispatch.Provider value={dispatch}>…</UserDispatch.Provider>

* + 이렇게 설정해주면 Provider에 의하여 감싸진 컴포넌트 중 어디서든지 Context의 값을 다른 곳에서 바로 조회해서 사용할 수 있다.
  + 아래 수정은, UserDispatch는 Context를 만들어서, 어디서든지 dispatch를 꺼내 쓸 수 있도록 준비를 해준 것.
  + export를 해준 이유는 나중에 다른 곳에서 사용하고 싶을 때 사용하기 쉽도록 하기 위해.

//App.js

…

export const UserDispatch = React.createContext(null);

function App() {

…

return (

<UserDispatch.Provider value={dispatch}>

..

</UserDispatch.Provider>

);

1. context 적용하기
   1. App.js에서 onToggle과 onRemove함수를 지우고, 이거와 연결된 props들을 다 지운다.
   2. UserList.js 에서도 마찬가지로 onToggle과 onRemove를 다 지운다.
   3. UserList.js 에서 useContext를 사용하여 context를 받아와서 dispatch를 사용해주면 된다.

//UserList.js

import React, { useContext } from ‘react’;

import { UserDispatch } from ‘./App’;

const User = React.memo(funtion User({ user }) {

const dispatch = useContext(UserDispatch);

return (

…

onClick={() => {

dispatch({ type: ‘TOGGLE\_USER’, id: user.id });

}}

…

onClick={() => {

dispatch({ type: ‘REMOVE\_USER’, id: user.id });

}}

…

);

…

});

**숙제 정답**

//App.js

function App() {

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

  const { users } = state;

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

  return (

    <UserDispatch.Provider value={dispatch}>

      <CreateUser />

      <UserList users={users}/>

      <div> 활성 사용자 수 : {count}</div>

    </UserDispatch.Provider>

  );

}

export default App;

//CreateUser.js

import React, { useContext, useRef } from 'react';

import useInputs from './hooks/useInputs';

import { UserDispatch } from './App';

const CreateUser = () => {

    const [{ username, email }, onChange, reset] = useInputs({

        username: '',

        email: ''

    });

    const nextId = useRef(4);

    const dispatch = useContext(UserDispatch);

    const onCreate = () => {

        dispatch({

          type: 'CREATE\_USER',

          user: {

            id: nextId.current,

            username,

            email

          }

        });

        reset();

        nextId.current += 1;

      };

    return (

        <div>

            <input

                name="username"

                placeholder="계정명"

                onChange={onChange}

                value={username}

            />

            <input

                name="email"

                placeholder="이메일"

                onChange={onChange}

                value={email}

            />

            <button onClick={onCreate}>등록</button>

        </div>

    );

}

export default React.memo(CreateUser);

* CreateUser.js에서 다 관리하도록 한다
* 매개변수를 다 지우고, CreateUser.js 안에서 다 관리하도록 useInputs을 import한다.
* 그리고 원래 App.js에서 사용했던 것처럼 username, email, onChange, reset함수를 useInputs에서 가져온다.
* onCreate함수도 가져와서 그대로 붙여넣기한다. 이때 useCallback()은 안붙여도 됨.
* dispatch를 사용하기 위해 App.js에서 UserDispatch를 import하여 useContext를 통해 조회한다.
* nextId도 createUser 내에 선언하여 사용한다.