**커스텀 Hooks 만들기**

-컴포넌트를 만들다보면 반복되는 로직이 자주 발생함.

-예를 들어 input을 관리하는 코드는 관리할 때마다 비슷한 코드가 반복됨.

-이때 커스텀 Hooks를 만들어서 반복되는 로직을 쉽게 재사용하는 방법에 대한 챕터

**예제 따라하기**

1. src 디렉터리 밑에 hooks 디렉터리 생성. 그 안에 useInputs.js 를 생성
2. 보통 커스텀 Hooks를 만들 때 이렇게 use라는 키워드로 파일을 만들고, 그 안에 함수를 작성함
3. 그 파일 안에 useState, useEffect, useReducer, useCallback 등을 이용하여 원하는 기능을 구현해주고

컴포넌트에서 사용하고 싶은 값들을 반환해주면 됨

**//useState를 사용한 방법**

import { useState, useCallback } from 'react';

function useInputs(initialForm) {

    const [form, setForm] = useState(initialForm);

    //change

    const onChange = useCallback(e=> {

        const { name, value } = e.target;

        setForm(form => ({ ...form, [name]: value }));

    }, []);

    const reset = useCallback(() => setForm(initialForm), [initialForm]);

    return [form, onChange, reset];

};

export default useInputs;

**//useReducer를 사용한 방법**

import { useReducer, useCallback } from 'react';

function reducer(state, action) {

    switch (action.type) {

        case 'CANGE\_INPUT':

            return {

                ...state,

                [action.name] : action.value     //20장에선 initial state로 user와 inputs 속성이

                                               //둘다 있어서 inputs : { } 을 따로 지정해줬지만

                                                 //여기선 initialForm에 inputs만 있어서 따로 지정해줄

                                                 //필요는 없다

            };

        case 'RESET':

            return Object.keys(state).reduce((acc, current) => {

                acc[current] = ' ';

                return acc;

            }, {});

        default:

            return state;

    }

}

function useInputs(initialForm) {

    const [form, dispatch] = useReducer(reducer, initialForm);

    //change

    const onChange = useCallback(e => {

        const { name, value } = e.target;

        dispatch({

            type: 'CANGE\_INPUT',

            name,

            value

        });

    }, []);

    const reset = useCallback(() =>

        dispatch({

            type: 'RESET',

        })

    , []);

    return [form, onChange, reset];

};

export default useInputs;

**Object.keys(state).reduce(() => { }, [deps])**

1. reduce함수 : 배열의 각 요소에 대해 callback을 실행하며 단 1개의 출력결과를 만듦
   1. 배열 원소를 입력으로 하여 1개의 수치, 문자열, 배열 또는 딕셔너리를 만드는 것.
   2. 보통 .reduce(acc, current) => { }, initialValue); 로 쓰는 거 같음
      1. 수치일 때 : 값이 빈 값에 저장되어 반환됨.
         1. .reduce((acc, current) => { }, initialValue);
      2. 배열일 때 : 값이 빈 배열에 저장되어 반환됨
         1. .reduce((acc, current) => {}. []);
      3. 딕셔너리일때 : 값이 빈 딕셔너리에 저장되어 반환됨
         1. .reduce((acc, curret) => {}. {});
   3. 앞 acc는 accumulator로 누적되어 쭉 가는 값 같고, current는 현재 값을 말하는 듯.
2. 위 예제에서는 form이 딕셔너리형태로 되어있으니

.reduce(() => {}. {}) 형태로 되고

초기화 시켜주는 거니

.reduce((acc, cur) => { acc[cur] = ‘’; return acc; }, {} )로 했다