React Hook기본 다지기#0

Use State

# 서론 :

나는 React를 학원에서 처음 배울 당시 클래스 형 컴포넌트를 이용하여, web app을 작성하였다.그 당시에도 Hook이 존재는 하였지만, CRA에서는 Hook을 지원하기 전이였고, 함수형 컴포넌트는 아직 보완할 점이 많아 React 사용자들에게는 뜨거운 감자 같은 것 이었다.

시간이 흘러 CRA에서 Hook을 공식지원하게 될 때, 나 또한 Hook에 발을 담가 보았지만, class형으로 만든 내 코드는 여전히 잘 동작 하였기에 나의 주목을 끌지 못하였다.

2020년 하반기 많은 것들이 변화하였다. Class형으로 작성한 내 코드는 규모가 커질 수록 관리하기 매우 어려웠으며, 빌드 속도가 증가하는 등 많은 이슈가 내 앞으로 다가왔다. 또한 Typescript 공부 할 수록, 함수형 컴포넌트와의 궁합이 좋겠다라는 생각이 들었기에, Hook에 대한 기초를 포스팅할 계획을 세우게 되었다.

( 이 시리즈 포스팅은 react에 대한 기초 강의가 아닙니다. )

# Use State

Class component를 학습한 인원들은 ‘state’의 존재를 알 것이다. state는 react element가 가지고 있는 상태이며, 이 state가 변경 될 때 마다, react element는 새롭게 render 메서드를 호출 한다.

/src/app.js

import logo from './logo.svg';

import './App.css';

import React, {useState} from 'react'

function App() {

const [counter,setCounter] = useState(0);

return (

<div className="App">

{counter}

</div>

);

}

export default App;

useState의 함수의 기본 형태는 다음과 같다.

function useState(value){

let \_\_state = value;

function \_\_setState(value){

if(value !== \_\_state){

//렌더링

}

return \_\_state = value;

}

return [\_\_state,\_\_setState];

}

내부 동작이 조금 더 있지만, 이해를 돕기 위해 간소화 하였다.

useState의 작동 과정은 useState에 전달인자로 기본 값을 전달하여 호출하면, 해당 기본 값을 함수 내에 저장하고, 클로저를 사용하여, \_\_state와 이를 변경 할 수 있는 \_\_setState 함수를 배열에 담아 리턴한다.

만약 setState를 사용하여, \_\_state와 setState로 들어온 값이 다르다면 해당 컴포넌트를 다시 렌더링 한다.

# 함수형 컴포넌트내에서 useState 2개 이상 사용하기

클래스형 컴포넌트에서는 state는 this.state로 관리되기 때문에, 다양한 입력폼을 받는 컴포넌트에서는 {name: “”, id : “”, pw : “”} 와 같이 단일 객체로 state를 관리 할 수 있었다. 그렇기에, state가 굳이 변경 되지 않아도 되는 몇몇의 값에 대해서 렌더를 막기 위해 ‘shouldComponentUpdate’ 메서드를 오버라이드 하고 if문으로 렌더링이 변경되지 않게 작업 하였다.

(이번 포스팅에서는 ‘shouldComponentUpdate’를 대체할 ‘useMemo’ 함수에 대해서는 설명 하지 않도록 하겠다. )

결국 문제는 클래스 컴포넌트에서 state를 한개의 단일 객체로 관리하고 새롭게 객체를 만드는 과정에서 브라우저가 다시 객체에 기존 프로퍼티를 할당하면서 발생하는 ‘시간’ 이다.

useState는 다음과 같이 함수 내에서 여럿 호출이 가능하다.

function App() {

const [counter,setCounter] = useState(0);

const [name,setName] = useState('hokeys');

return (

<div className="App">

{counter}

{name}

</div>

);

}

또한 한개의 state를 다음과 같이, 객체로 관리하는 것 또한 가능하다.

function App() {

const [counter,setCounter] = useState(0);

const [name,setName] = useState('hokeys');

const [obj,setobj] = useState({id : "hokidoki", pw : "0000"});

return (

<div className="App">

{counter}

{name}

{obj.id}

{obj.pw}

</div>

);

}

# setState

useState로 전달된 배열의 두번째 함수인 setState 함수는 state를 변경하는 용도의 함수이다. 이 함수를 호출 할 때는 state가 새롭게 변경될 값을 전달 하면 된다.

function App() {

const [counter,setCounter] = useState(0);

const [name,setName] = useState('hokeys');

const [obj,setobj] = useState({id : "hokidoki", pw : "0000"});

return (

<div className="App">

{counter}

<button onClick={()=> setCounter(counter + 1 )}>+</button>

</div>

);

}

useState는 두번째 전달인자로 callBack을 지원하지 않는다. State가 변경되어 사이드 이펙트를 발생시키려면 useEffect함수를 사용하면 된다.

# useState 식별자 규칙

위에서 작성한 것과 같이, useState에서 리턴된 배열의 식별자는 [<stateId>,set<stateId>] 처럼 사용하는 것이 일반적이다.

const [name,setName] = useState('hokeys');

const [obj,setobj] = useState({id : "hokidoki", pw : "0000"});