Hook 기본 다지기#2

useEffect#1

# 서론

“Side Effect”를 사용하기 위해서는 useEffect 함수로 의존성 배열을 통해 발생 시킨다고 지난번 포스팅에서 언급하였다. 그러나 이전에 작성한 함수형 컴포넌트를 내부를 보면 useEffect를 내부에서 사용함에 따라, sideEffect를 발생 시키는 useEffect 함수를 재 사용 하기 어렵고, 간결함이 장점인 함수형 컴포넌트가 useEffect함수로 오염되어 버렸다.

function App() {

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

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

const [show,setShow] = useState(false);

useEffect(()=>{

console.log("component Did Mount !!!")

},[])

useEffect(()=>{

console.log(counter)

console.log("counter is Update")

},[counter,name])

return (

<div className="App">

{counter}

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

<input value={name} onChange={(e) => setName(e.target.value)}></input>

<button onClick={()=> setShow(!show)}>쇼 !</button>

{show ? <ShowName name={name}/> : null}

</div>

);

}

…

# Custom Hook

React에서 제공하는 hook을 이용하여, Custom hook을 작성 할 수 있다. Custom hook을 사용하면, 가독성이 상승하고, 로직을 쉽게 분리하여 재사용 및 관리할 수 있다.

먼저 componentDidMount 역할을 하던 useEffect를 따로 분리하여 ./src/hooks/lifecycle.js에 업로드 하도록 하자 .

/src/hooks/lifecycle.js

import { useState, useEffect} from 'react';

export const useCDM = () => {

const [mount, setMount] = useState(false);

useEffect(()=>{

setMount(true);

},[])

return mount;

}

/src/App.js

\

function App() {

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

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

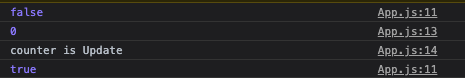
const [show,setShow] = useState(false);

const isMount = useCDM();

console.log(isMount)

…

이 결과는 다음과 같다.



(로그)

처음 App 컴포넌트가 마운트 되기 전에는 hook에 의해 false가 출력된다. 이후 커스텀 hook에서 useEffect가 setMount를 통하여, 변경 하였기 때문에 true가 출력된다.

이 useCDM이라는 함수를 응용하면, 다음과 같이 효율적으로 사용 가능 하다.

/src/hooks/lifecycle.js

import { useState, useEffect} from 'react';

export const useCDM = (callBack) => {

const [mount, setMount] = useState(false);

useEffect(()=>{

setMount(true);

callBack();

},[])

return mount;

}

/src/api/api.js

export const getInitData = (good) => {

return new Promise((resolve,reject)=>{

setTimeout(()=>{

resolve(good)

},1000)

})

}

/src/app.js

import { getInitData } from './api/api';

function App() {

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

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

const [show,setShow] = useState(false);

const appInitialize = () => {

getInitData("blue").then((initData)=>{

console.log(initData);

setName(initData);

}).catch((error)=>{

console.log(error);

})

}

const isMount = useCDM(appInitialize);

…

}

이와 같이 useCDM의 매개변수를 콜백으로 변경하고, 커스텀 훅이 마운트 될 때 콜백이 실행 되게 끔 하면, 코드가 useEffect로 떡칠 되는 상황을 방지 할 수 있다.