Hook 기본 다지기#3

useMemo

# 서론

프론트엔드 개발자가 가장 신경 쓰는 것은 무엇일까 ? UI/UX ? , 모델링의 최적화 ? 화려한 애니메이션? 물론 다 맞는 말이지만, 옆에 이 같은 일을 같이 해줄 동료가 존재한다면, 그렇게 신경 쓰지 않아도, 괜찮을 듯 하다. 그렇다면 리액트 개발자가 가장 신경 쓰는 것은 무엇일까 ? 적어도 나는 state와 props의 변경으로 인하여 컴포넌트에서 발생하는 함수들의 재계산으로 인한 자원 소모와 쓸 대 없는 렌더링의 방지 일 것이다. 그래서 오늘은 함수들의 재 계산을 방지하는 useMemo 함수를 살펴 볼 예정이다.

# State와 Props의 변경으로 인한 렌더링

우리 모두 컴포넌트에서 state와 props가 변경되면, 렌더링이 발생된다는 것을 익히 알고 있을 것이다. 특히 함수형 컴포넌트는 그저 jsx 문법을 리턴하는 함수이기 때문에, state와 props가 변경이 된다면, 함수 내부에서 발생하는 함수들이 호출 되게 된다. 그 예를 보도록 하자.

/src/App.js

import {useState,useEffect} from 'react';

function App(){

const [myName, setMyName ] = useState("");

const myNameLength = nameLength(myName);

return (

<div className="App">

<input value={myName} onChange={(e)=> setMyName(e.target.value)}></input>

{myNameLength}

</div>

);

}

function nameLength(name){

console.log("이름 길이 계산중");

return name.length;

}

export default App

이 코드를 실행하여, input을 입력하면, input을 입력한대로, nameLength 함수가 실행되는 것을 볼 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

(결과)

그렇다면, App함수를 다음과 같이 변경 해보자

/src/App.js

function App(){

const [myName, setMyName ] = useState("");

const [myAddress,setMyAddress] = useState("");

const myNameLength = nameLength(myName);

return (

<div className="App">

<div>

<input value={myName} onChange={(e)=> setMyName(e.target.value)}></input>

{myNameLength}

</div>

<div>

<input value={myAddress} onChange={(e)=> setMyAddress(e.target.value)}></input>

</div>

</div>

);

}

이후, 다음과 같이 화면이 등장할 것이고, 아래의 input에 입력을 해보자.

텍스트이(가) 표시된 사진

자동 생성된 설명

(결과)

myName state는 변경되지 않았으나, myAddress가 변경 될 때 마다, nameLength함수 또한 호출 되고 있다. 현재는 작은 데이터 연산만 하기 때문에 성능 이슈가 적다. 그러나, nameLength함수가 복잡하고 CPU연산이 많아 시간이 많이 소모된다면, 연산과 관련 없는 값이 변경될 때도 불필요한 자원을 소모하게 된다.

# useMemo

위와 같은 상황을 인지한 hook은 컴포넌트에서 특정 함수와 관련 있는 값이 변경될 때만 함수를 실행하고, 관련 없는 state나 props가 변경 될 때는 기존 return 한 값을 그대로 return 하는 memorization api를 제공한다. 이 api의 함수 명이 useMemo 이다.

useMemo의 형태는 다음과 같다. useMemo(callBack, dependencyArray) useMemo는 dependency Array에 있는 값들이 변경 될 때만, callBack으로 전달된 함수를 호출 한다.

이제 App.js 에서 useMemo를 사용해보도록 하자.

/src/App.js

import {useState,useMemo} from 'react';

function App(){

const [myName, setMyName ] = useState("");

const [myAddress,setMyAddress] = useState("");

const myNameLength = useMemo(()=>nameLength(myName),[myName]);

return (

<div className="App">

<div>

<input value={myName} onChange={(e)=> setMyName(e.target.value)}></input>

{myNameLength}

</div>

<div>

<input value={myAddress} onChange={(e)=> setMyAddress(e.target.value)}></input>

</div>

</div>

);

}

결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

(결과)

아무리, myAddress가 변경되어도, 함수는 호출 되지 않는다. 이처럼 useMemo API는 함수 호출을 제한 하면서, 불 필요한 cpu연산과 실행 시간을 줄이도록 지원한다.

# 주의 사항