8253jang@daum.net
장희정

React Hooks

React Hook은 리액트 16.8버전에서 새롭게 등장한 개념으로 **React 함수형 컴포넌트에서 상태(state)나 생명주기(lifecycle) 기능을 사용할 수 있도록 해주는 JavaScript 함수이다.** 클래스형 컴포넌트를 사용할 때 필요했던 this.state, this.setState, componentDidMount 등의 기능을 함수형 컴포넌트에서도 사용할 수 있게 해준다.

공식문서 :https://react.dev/reference/react/hooks

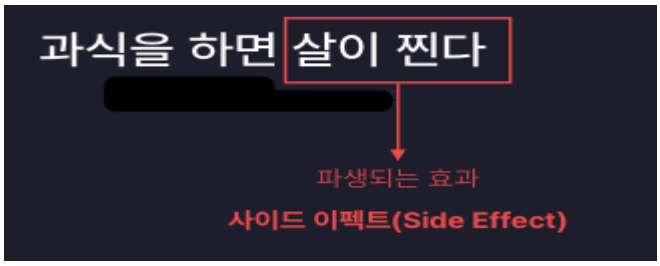
☞Built-in React Hooks

State Hooks	useState
	useReducer
Context Hooks	useContext
Ref Hooks	useRef
Effect Hooks	useEffect
Performance Hooks	useMemo
	useCallback

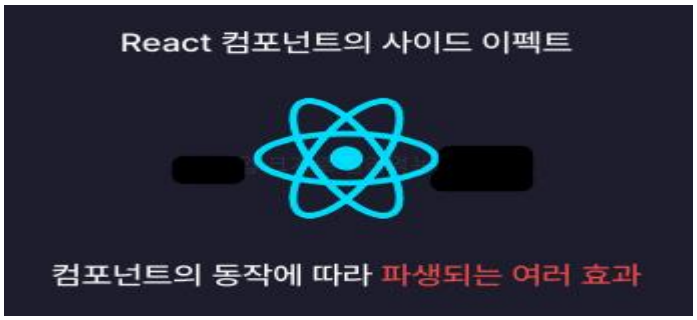
useEffect

React 컴포넌트의 사이드 이펙트를 제어하는 Reack Hook

사이드 이펙트(Side Effect)란?
우리말로는 "부작용" 이라는 뜻!
리엑트에서는**"부수적인 효과","파생되는 효과"** 로 해석한다.

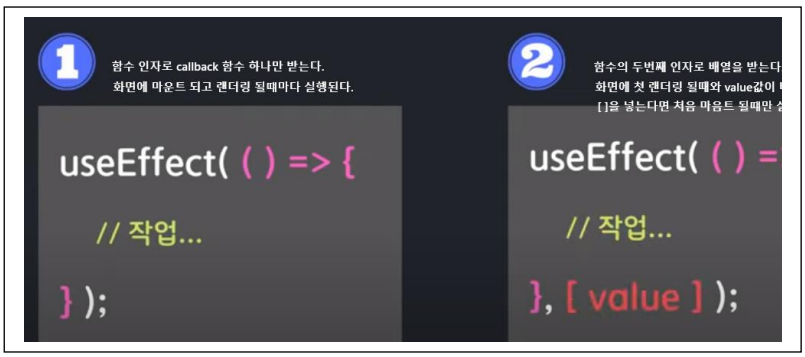


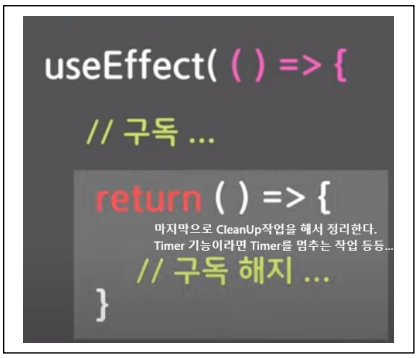
8253jang@daum.net
장희정



React 컴포넌트 라이프 사이클

8253jang@daum.net
장희정





● Step1.useEffect ... **최초 랜더링 (마운트)**

```
import React, { useEffect } from 'react';
function App() {
  //실행시점 : 1)App() 함수가 최초로 실행될때(마운트될때)
  // 마우스로 눌러보면 문법을 확인(callback, dependency List)
  useEffect(()=>{
    console.log("App() 실행됨...useEffect");
  });
  return (
    <div>
      Hello World
    </div>
  );
}
```

Hello World

App() 실행됨...useEffect

```
function App() {

  //2. State 사용
  const [num, setNum] = useState(0);
  useEffect(()=>{
    console.log("App() 실행됨...useEffect");
  });
  return (
    <div>
      <h1>Number : {num}</h1>
    </div>
  );
}
```

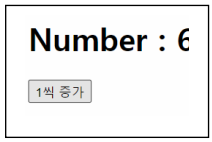
Number : 0

1번 실행됨을 확인

● **Step2 버튼 추가하고 값 변경**

8253jang@daum.net
장희정

```
//실행시점 :
//1)App() 그림이 최초 그려질때
//2)상태변수가 변경될때
useEffect(()=>{
  console.log("App() 실행됨...useEffect");
});
return (
 <div>
   <h1>Number : {num}</h1>
   <button onClick={()=>{
          setNum(num+1)
          }}>
           1씩 증가
   </button>
 </div>
);
```

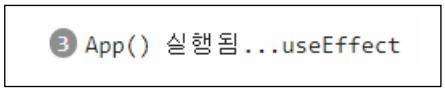


버튼 누를 때 마다useEffect가 실행된다.

상태변수가 실행될때 App이 실행되고 return이 동작하면서 다시 랜더링된다.

● Step3 Dependency List

```
function App() {

 const [num, setNum] = useState(0);
 //서버측에서 데이타(5) 전달받았다 치고
 const download = ()=>{
   let downloadNumber = 5;
   setNum(downloadNumber);
 }
 useEffect(()=>{
   console.log("App() 실행됨...useEffect");
   //useEffect가 최초 실행될때 download()함수 실행
   download();
 });
 return (
```



▶ App 실행(useEffect실행됨)-->그안에 있는 download() 함수 실행됨 -->상태값을 5로 변경

--> 화면에 5가 뜬다.//

▶ 버튼을 클릭하면 5에서 6으로 변경되어야 하는데 6으로 변경되자마자 다시 5로 뜬다. 왜?

-->setNum(num+1) 에서 순간적으로 6이 되지만 상태값이 변경되었기에 다시 useEffect가 실행되고 다시 5로 값을 되돌려 놓기 때문이다.

▶ 처음 랜더링 될 때useEffect는 실행되어야 하지만상태값이변경될때마다 호출되지 않도록 한다.

-->useEffect()함수의 두번째 인자값을 사용해야 한다.

```
useEffect(()=>{
  console.log("App() 실행됨...useEffect");
  download();
},[num]);
```

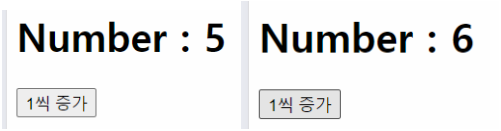
상태변수 num이 변경될때마다useEffect실행

8253jang@daum.net
장희정

```
useEffect(()=>{
  console.log("App() 실행됨...useEffect");
  download();
},[]);
```

[ ]빈 배열은 어디에도 의존하지 않는다는 뜻

최초에 랜더링 될 때 만 useEffect()함수 실행



5에서 1씩 증가해서 다음 값은 6을 출력한다

5) 또다른 상태값을 추가

```
const [search, setSearch] = useState(0);
const download = ()=>{
  let downloadNumber = 5;
  setNum(downloadNumber);
}
useEffect(()=>{
  console.log("App() 실행됨...useEffect");
  download();
},[search]);
return (
 <div>
  {/* 원래는 입력받은 데이타라 생각하고 2를 그냥 넣음 */}
   <button onClick={()=>{
          setSearch(2);
          }}>
           검색하기
   </button>
   <h1>Number : {num}</h1>
```

[search] search상태값에 의존적으로 작성

useEffect() 함수가 실행된다. 이때 출력숫자가

다시 5로 바뀌는것을 주목

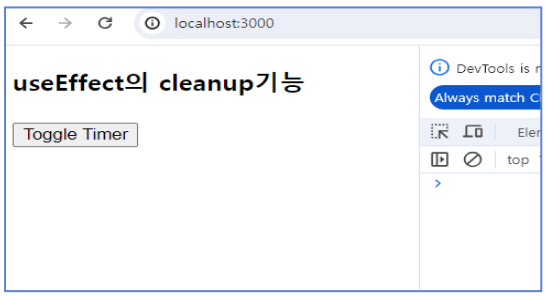
● **Step4useEffect 정리작업**

8253jang@daum.net
장희정

```jsx
import React, { useState } from 'react';
import Timer from './component/Timer';

//useEffect의 cleanUp에 대해서 알아보는 예제
const App02_1 = () => {
   const [showTimer, setShowTimer] = useState(false);
    return (
        <div>
           {/* <Timer/> */}
            <h3>useEffect의 cleanup기능 </h3>
            {showTimer && <Timer/> }
            <button onClick={()=>{setShowTimer(!showTimer)}}>Toggle Timer</button>
        </div>
    );
};

export default App02_1;
```

```jsx
import React, { useEffect } from 'react';

const Timer = () => {
    useEffect(()=>{
        const timer = setInterval(()=>{
            console.log("타이머 돌아가는중");
        }, 1000);

    },[]);//최초에 한번만 실행

    return (
        <div>
            <span>타이머를 시작합니다. 콘솔을 보세요.</span>
        </div>
    );
};

export default Timer;
```

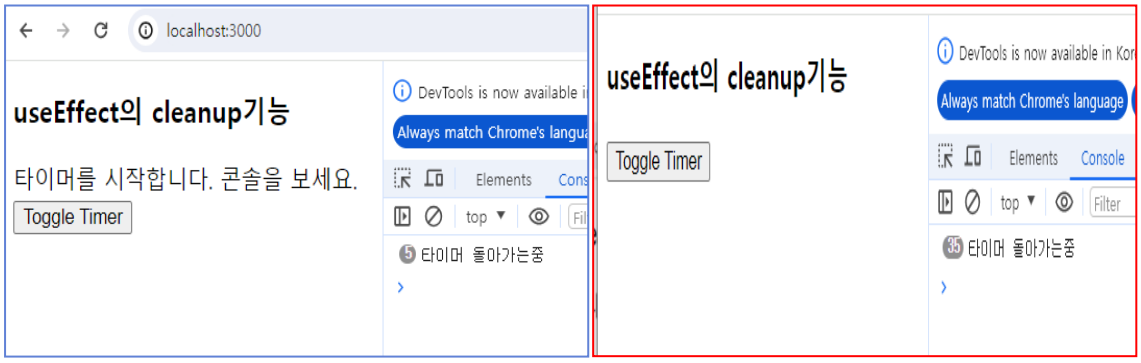
실행화면



"Toggle Timer" 버튼을 클릭하면 <Timer>가 mount되고 , 다시 클릭하면 unmount된다.

그러나,

unmount되어도 돌고 있는 setInterval은 계속 실행중이다. 이것을 정리 해줘야 한다.



useEffect 함수안에서 return 함수를 작성하여 정리작업을 한다.

```
3   const Timer = () => {
4       useEffect(()=>{
5           const timer = setInterval(()=>{
6               console.log("타이머 돌아가는중");
7           }, 1000);
8
9           //정리
10          return ()=>{
11              clearInterval(timer);
12              console.log("타이머 종료합니다.")
13          }
14
15      },[]);//최초에 한번만 실행
16
17      return (
18          <div>
19              <span>타이머를 시작합니다. 콘솔을 보세요.</span>
20          </div>
21      );
```

useMemo

useMemo의 Memo는 Memoization(메모이제이션)을 뜻하는데 기존에 수행한 연산의 결과값을 어딘가에 저장해 두고 동일한 입력이 들어오면 저장해둔 값을 재활용하는 프로그래밍 기법이다.(캐싱기법)

8253jang@daum.net
장희정

React에서 성능 최적화를 위해 사용하는 훅으로 주로 **값 계산**에 드는 비용이 클 때, 해당 값이 **변경되지 않으면 다시 계산하지 않고 이전 값을 재사용**하도록 한다. 주로 **리렌더링 성능을 최적화**하려는 목적으로 사용된다.

● Step1.

1) list 상태값 출력

```
function App() {

  const [list, setList] = useState([1,2,3,4]);
  return (
   <div>
      <div>
        {list.map(i=>(
        <h1>{i}</h1>
        ))};
      </div>
   </div>
  );
}
```

map() 함수는 일종의 for문으로 배열을 리턴한다.

1

2

3

4

2) sum 상태값 콘솔 출력

```
function App() {
  const [list, setList] = useState([1,2,3,4]);
  const getAddResult=()=>{
    let sum = 0;
    list.forEach((i)=>(sum = sum +i));//한줄일때 { }생략
    console.log('sum ',sum);
  };
  return (
   <div>
      <div>
        {list.map(i=>(
        <h4>{i} </h4>
        ))};
      </div>
      <div>합계 : {getAddResult()}</div>
   </div>
  );
```

1

2

3

4

;

합계 :

sum 10

콘솔창에는 10이라고 출력, 화면에도 동일한 값이 출력되려면 getAddResult() 함수가 리턴해야함

3)화면에 sum이 출력됨

8253jang@daum.net
장희정

```
const getAddResult=()=>{
  let sum = 0;
  list.forEach((i)=>(sum = sum +i));//
  console.log('sum ',sum);
  return sum;
};
return (
 <div>
    <div>
      {list.map(i=>(
      <h4>{i} </h4>
      ))};
    </div>
    <div>합계 : {getAddResult()}</div>
 </div>
```

1
2
3
4
;
합계 : 10

4) 버튼 클릭하면 list 상태값에 값 하나 추가

가장쉬운 전개연산자를 사용한다.

```
return (
 <div>
    <button
      onClick={()=>{
      setList([...list, 10]);
      }}
    >리스트 값 추가
    </button>
    <div>
      {list.map(i=>(
```

리스트 값 추가
1
2
3
4
;
합계 : 10

리스트 값 추가
1
2
3
4
10
;
합계 : 20

리스트 값 추가
1
2
3
4
10
10
10
;
합계 : 40

버튼 클릭하면 상태값 list가 변경됨으로 return이 다시 실행(재랜더링)...getAddResult() 함수가 다시 실행된 결과이다.

● Step2.

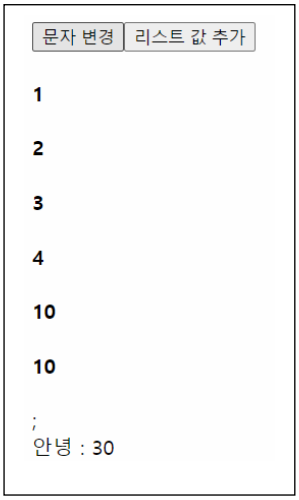
1) 문자변경 버튼을 추가하고 실행결과 확인

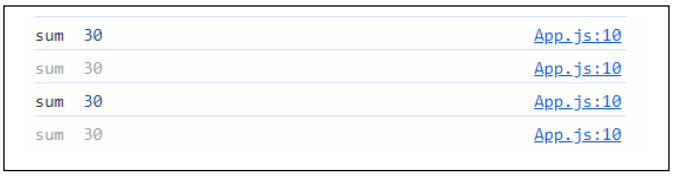
```
function App() {
  const [list, setList] = useState([1,2,3,4]);
  const [str, setStr] = useState("합계");
```

```
  <div>{str} : {getAddResult()}</div>
```

```
return (
 <div>
    <button
      onClick={()=>{
      setStr("안녕");
      }}
    >문자 변경
    </button>
```

8253jang@daum.net
장희정





문자변경 버튼을 클릭하면 "안녕"으로 상태값이 변경됨으로 다시 return부분이 재랜더링된다 -->getAddResult() 함수가 다시 실행됨으로 필요하지 않은 연산작업을 수행한다!!

리스트 추가버튼을 클릭했을 때 getAddResult() 함수가 실행.

문자변경 버튼을 클릭했을때는getAddResult() 함수 실행을 막아둠.

2) useMemo() 함수 사용법

```
const memoizedValue = useMemo(() => {
// 값 계산
return 값;
}, [의존성 배열]);
```

첫 번째 인자: 값을 계산하는 함수. 이 함수는 의존성 배열이 변경될 때만 실행된다.
두 번째 인자: 의존성 배열. 배열 안에 있는 값이 변경될 때만 useMemo 안의 계산 함수가 실행된다. 의존성 배열이 빈 배열([])이면, useMemo는 컴포넌트가 처음 렌더링될 때만 값을 계산하고, 이후에는 재계산하지 않는다.

```
  return sum;
};
//마우스로 눌러서 문법을 잘 본다 . 순사가 getAddResult() 뒤에 와야한다.
//()=>any(뭘 기억할지),언제만 any가 실행될지를 지정함
const addResult = useMemo(()=>getAddResult(),[list]);
```

```
<div>{str} : {addResult}</div>
```

문자변경 버튼을 클릭하면 getAddResult() 함수가 실행되지 않는다.

8253jang@daum.net
장희정

useRef

useRef는 React에서 값을 참조하거나 DOM 요소에 직접 접근할 수 있도록 해주는 훅이다. useRef는 렌더링 사이에 값을 유지할 수 있는 방법을 제공하며, 컴포넌트의 렌더링과 관계없이 값이 유지되므로 렌더링을 유발하지 않기 때문에 성능상 이점이 있다.

사용법

const ref = useRef(initialValue);

initialValue : useRef로 생성할 참조의 초기 값이다. initialValue는 기본값이 될 수 있으며, DOM 요소를 참조하는 경우에는 null로 초기화하는 경우가 많다.

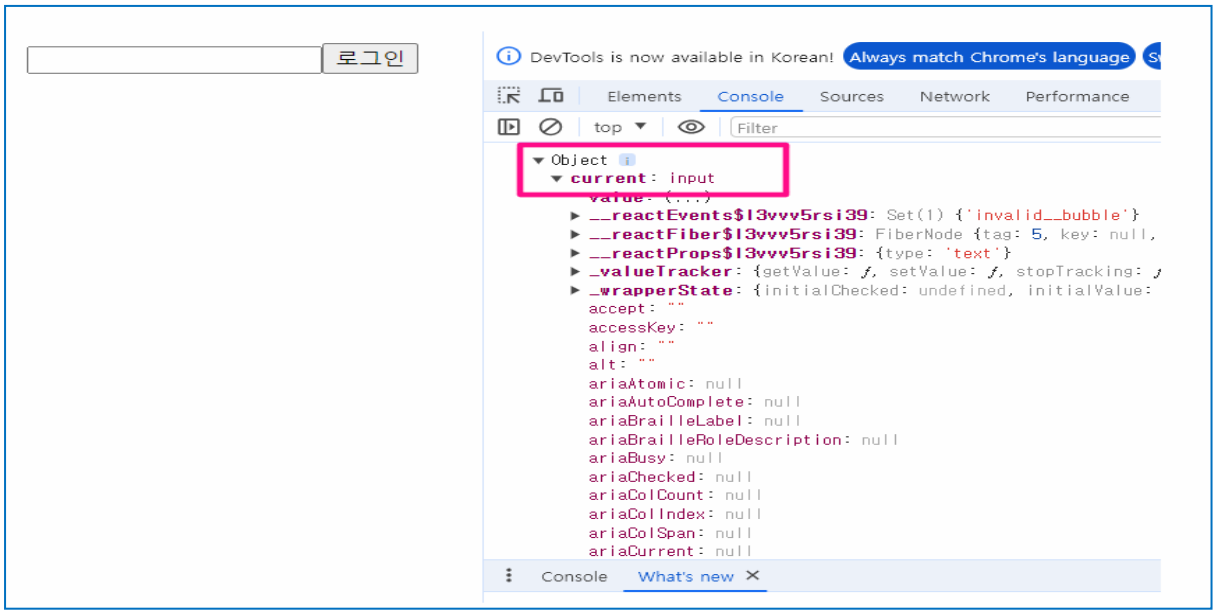
useRef 는 { current : T } 형태의 객체를 반환한다. T 는 useRef 에 전달한 initialValue 의 타입이고 ref.current 를 통해 값을 읽거나 수정할 수 있다.

text박스에 커서 놓기 - useRef를 이용하면 DOM요소에 쉽게 접근 할 수 있다.

```
1   import React, { useEffect, useRef } from 'react';
2
3   const App04_5 = () => {
4       const inputRef = useRef();
5
6       //처음 로딩될 때 커서 놓기
7       useEffect(()=>{
8           console.log(inputRef);
9
10          inputRef.current.focus();
11      },[]);
12      return (
13          <div>
14              <input type="text" ref={inputRef}/>
15           <button>로그인</button>
16
17      </div>
18       );
19  };
20
21  export default App04_5;
```

실행결과

8253jang@daum.net
장희정



버튼 클릭시 alert() 출력

```jsx
import React, { useEffect, useRef } from 'react';

const App04_5 = () => {
    const inputRef = useRef();

    //처음 로딩될 때 커서 놓기
    useEffect(()=>{
        console.log(inputRef);

        inputRef.current.focus();
    },[]);

    const login = ()=>{
        alert(`로그인되었습니다. ${inputRef.current.value}님`);
        inputRef.current.focus();
    }
    return (
        <div>
            <input type="text" ref={inputRef}/>
        <button onClick={login}>로그인</button>

    </div>
    );
```

8253jang@daum.net
장희정

2) 저장공간

- state의 변화가 생기면 다시 랜더링이 되고 컴포넌트 내부 변수들이 모두 초기화 되지만 **ref는 ref의 변화가 생겨도 랜더링이 되지 않고 변수들의 값이 유지가 된다.**

결론적으로,

useRef는 변화는 감지해야 하지만 그 변화가 랜더링을 감지하면 안 되는 경우에 사용하면 좋다.

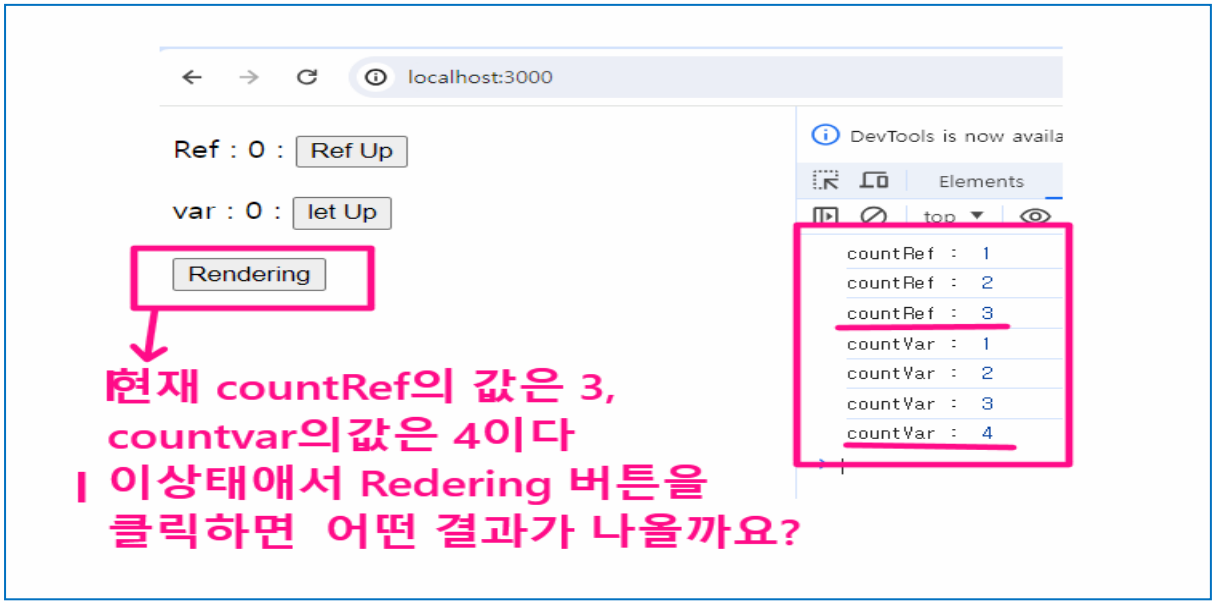
```
1  import React, { useRef, useState } from 'react';
2
3  const App04_2 = () => {
4      console.log("랜더링....");
5      const [count, setCount] = useState(0);
6      const countRef = useRef(0);
7
8      const stateUp = () =>{
9          setCount(count+1);
10     }
11
12     const refUp = ()=>{
13         countRef.current = countRef.current+1;
14         console.log("ref : " , countRef)
15     }
16     return (
17         <div>
18         <p>State : {count} :  <button onClick={stateUp}>state Up</button> </p>
19         <p>State : {countRef.current} :  <button onClick={refUp}>ref Up</button></p>
20         </div>
21     );
22 };
23
24 export default App04_2;
```

stateUp을 클릭하면 계속 랜더링 출력
refUp을 클릭하면 랜더링 출력안됨.

☞ **Ref와 let의 차이**

```
3  const Ref03 = () => {
4      console.log("리 랜더링중....")
5      const countRef = useRef(0);
6      let countLet = 0;
7
8      //화면갱신(리랜더링)을 위한 변수
9      const [render, setRender] = useState(0);
10
11     const refUp=()=>{
12         countRef.current = countRef.current +1;
13         console.log("countRef.current = " , countRef.current)
14     }
15
16     const letUp=()=>{
17         countLet = countLet +1;
18         console.log("countLet = " , countLet)
19     }
20
21     return (
22         <div>
23           <p>Ref :{countRef.current} <button onClick={refUp}>Ref up</button></p>
24           <p>let :{countLet} <button onClick={letUp}>let up</button></p>
25
26         <button onClick={()=>{setRender(render+1)}}>reRendering</button>
27         </div>
28     );
29 };
```

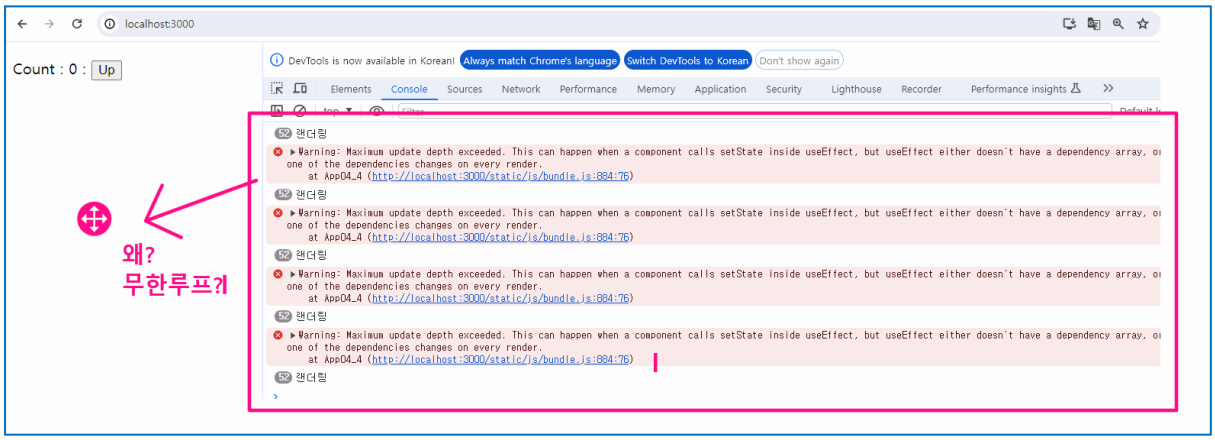
실행결과



8253jang@daum.net
장희정

총 랜더링이 된 횟수를 계산하고 싶다면?

```jsx
import React, { useEffect, useRef, useState } from 'react';

const App04_4 = () => {
    const [count, setCount] = useState(0);
    const [rederCount,  setRenderCount] = useState(1);
   // const rederCount = useRef(1);

    //랜더링이 된 횟수를 계산하고 싶다?
    useEffect(()=>{
        console.log("랜더링");
        setRenderCount(rederCount+1);
    });
    return (
        <div>
            <p> Count : {count} : <button onClick={()=>{setCount(count+1)}}>Up</button></p>

        </div>
    );
};

export default App04_4;
```



☞useState를 useRef로 변경해보자!

```jsx
const Ref04 = () => {
    const [count ,setCount] = useState(0);
    const countRef = useRef(0);

    //컴포넌트가 총 몇번 랜더링 되었는제 회수를 계산
    useEffect(()=>{
        console.log("랜더링....")
        //setCount(count+1)//무한루프(state가변경되면 컴포넌트함수가 다시 호출된다. -> useEffect실행->호출...)
        countRef.current = countRef.current+1;//리랜더링 안된다!!
        console.log("countRef.current = " + countRef.current);
    });
    return (
        <div>
           <p>Count = {count}  / countRef.current = {countRef.current} <br/>
            <button onClick={()=>{setCount(count+1)}}>Up</button></p>
        </div>
    );
};

export default Ref04;
```

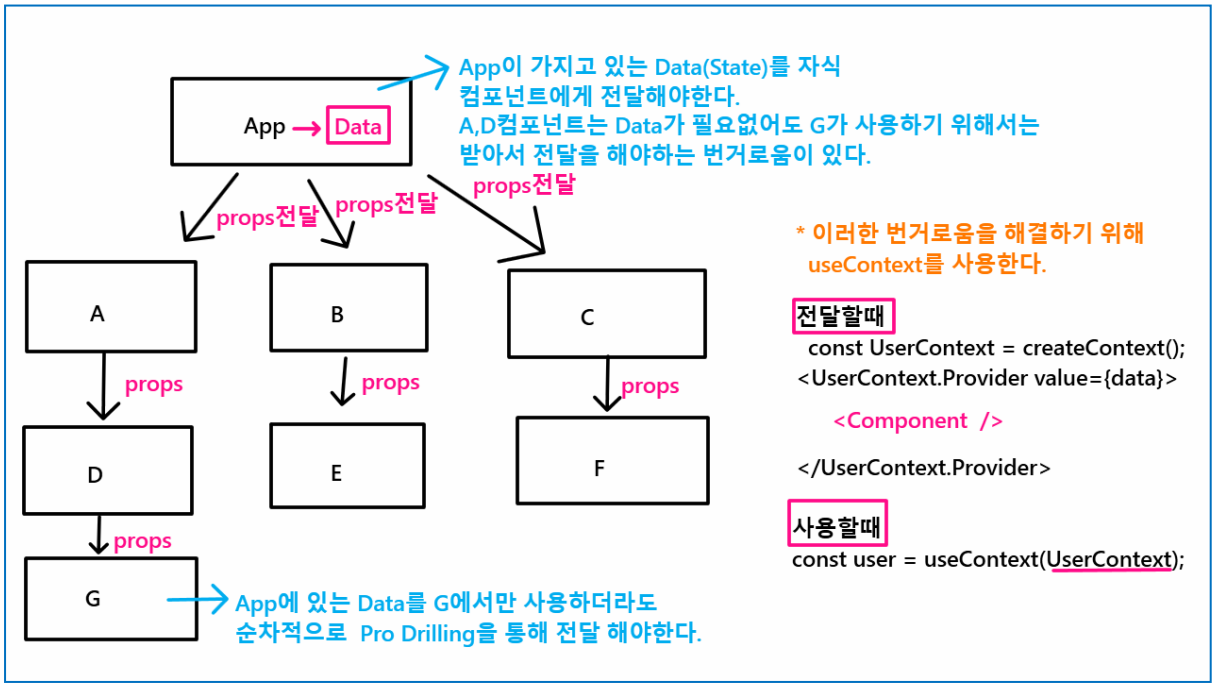
useContext

https://ko.legacy.reactjs.org/docs/context.html

글로벌하게 상태를 관리하는 방법

useState만 사용하는 것보다 깊게 중첩된 컴포넌트 간의 상태를 더 쉽게 공유할 수 있다.

Context를 사용하면 컴포넌트를 재사용하기 어려워 질 수 있다.



8253jang@daum.net
장희정

1) 컴포넌들 간에 Props를 통해서 데이타를 전달받는 코드

```
my-app > src > JS App.js > ...
1   import React, { useState} from "react";
2
3   function App() {
4     const [user, setUser] = useState("James");
5     return (
6       <>
7         <h1>Hello {user}!</h1>
8         <Component1 user={user} />
9       </>
10    );
11  }
12
13  function Component1({user}){
14    return (
15      <>
16        <h1>Component 1</h1>
17        <Component2 user={user} />
18      </>
19    )
20  }
```

```
21  function Component2({user}){
22    return (
23      <>
24        <h1>Component 2</h1>
25        <Component3 user={user} />
26      </>
27    )
28  }
29  function Component3({user}){
30    return (
31      <>
32        <h1>Component 3</h1>
33        <h2>Hello {user} Again~~~!!</h2
34      </>
35    )
36  }
37  export default App;
```

결과 확인

Hello James!

Component 1

Component 2

Component 3

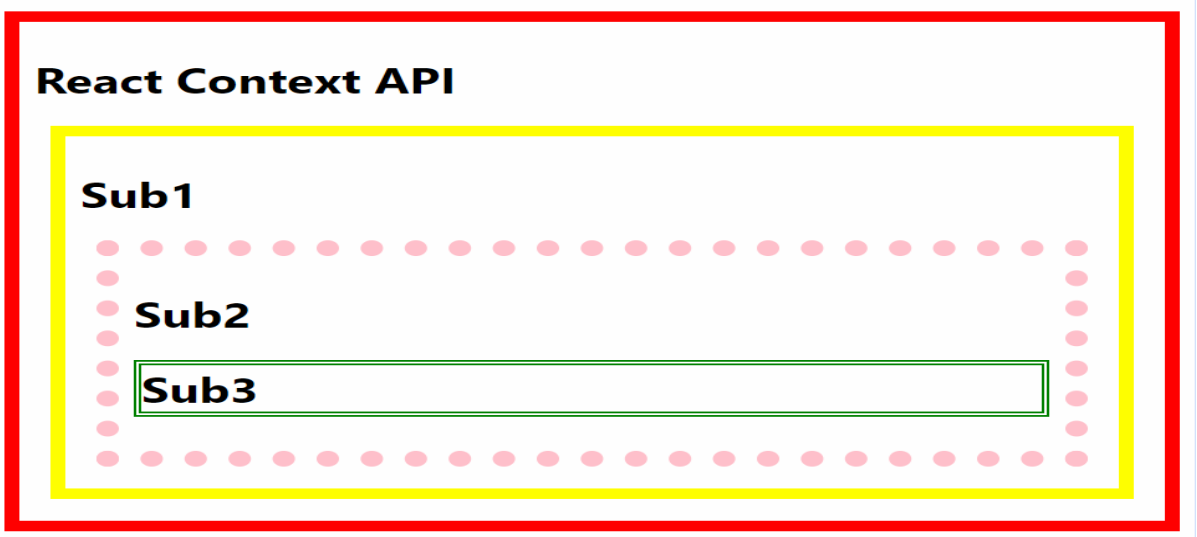
Hello James Again~~~!!

2) Context 사용해서 코드 작성

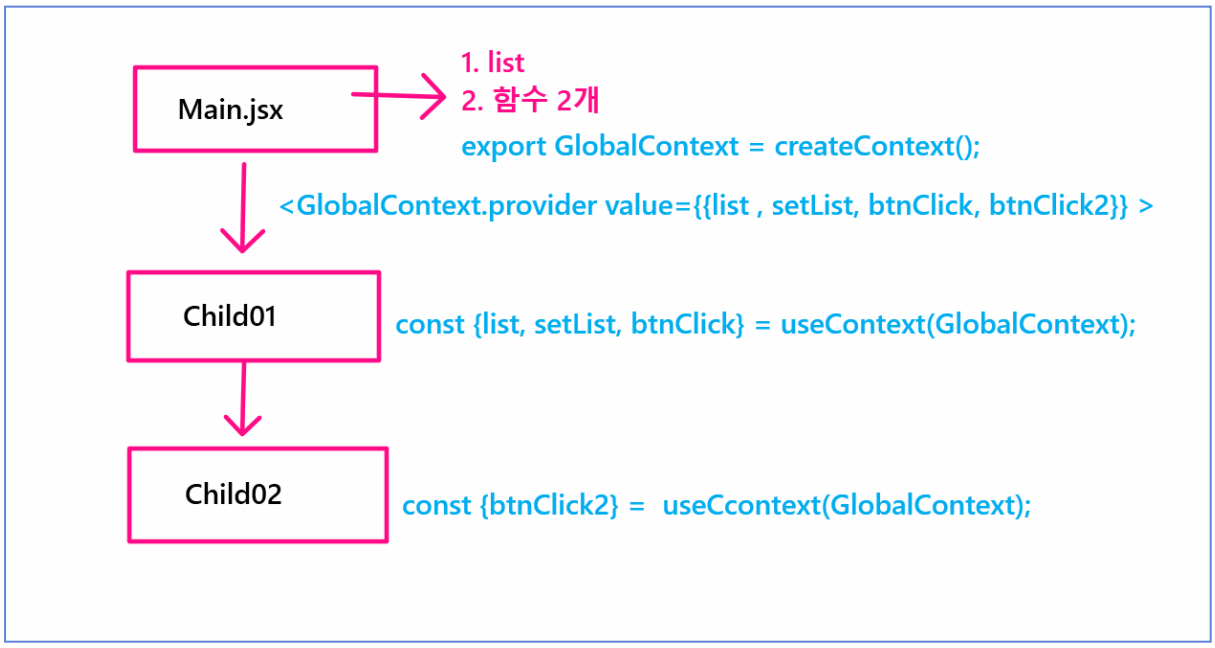
```
import React, { useState, createContext,useContext  } from "react";

const UserContext = createContext();

function App() {
  const [user, setUser] = useState("James");
  return (
    <>
      <UserContext.Provider value={user}>
        <h1>Hello {user}!</h1>
        <Component2 />
      </UserContext.Provider>
    </>
  );
}
```

```
function Component1(){
  return (
    <>
      <h1>Component 1</h1>
      <Component2 />
    </>
  )
}
function Component2(){
  return (
    <>
      <h1>Component 2</h1>
      <Component3/>
    </>
  )
}
function Component3(){
  const user = useContext(UserContext);
  return (
    <>
      <h1>Component 3</h1>
      <h2>Hello {user} Again~~~!!</h2>
    </>
```

8253jang@daum.net
장희정

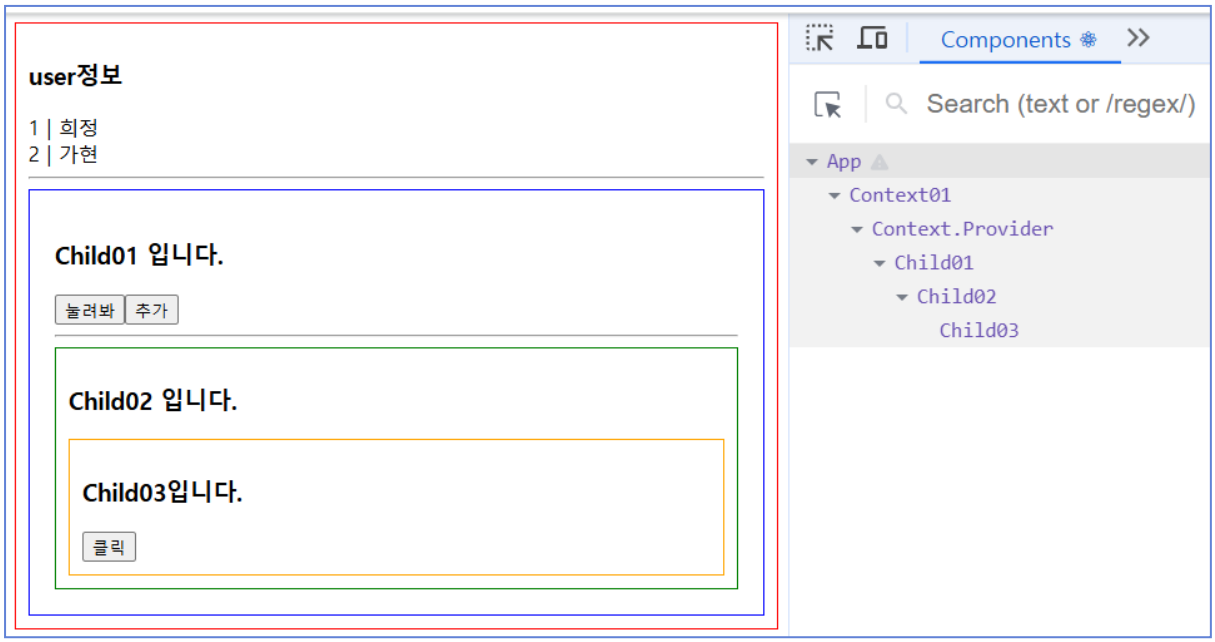


실습



실습화면

8253jang@daum.net
장희정



Context01.jsx

```jsx
import React from 'react';
import { useState } from 'react';

import { createContext } from 'react';
import Child01 from '../components/Child01';

export const GlobalContext = createContext();

const Context01 = () => {
    const [list, setList] = useState([{id:1, name:"희정"},{id:2, name:"가현"}]);

    const btnClick01 = ()=>{
        console.log("btnClick01 호출됨...")
    }
    const btnClick02 = ()=>{
        console.log("btnClick02 호출됨...")
    }
    return (
        <GlobalContext.Provider value={{list , setList , btnClick01 , btnClick02}}>
            <div style={{border:"1px red solid" , padding:"10px"}}>
                <h3 onClick={btnClick01}>user정보</h3>
                {list.map((user, index)=><div key={index}>{user.id} | {user.name}</div>)}
                <hr/>
                <Child01/>
            </div>
        </GlobalContext.Provider>
    );
};

export default Context01;
```

8253jang@daum.net
장희정

Child01.jsx

```jsx
import { useContext } from 'react';
import { GlobalContext } from '../src/Context01';
import Child02 from './Child02';

const Child01 = () => {
    const {list,setList, btnClick01} = useContext(GlobalContext);
    const addList = ()=>{
        setList([...list, {id:3, name:"나영"}]);
    }

    return (
        <div style={{border:"1px blue solid" , padding:"20px"}}>
            <h3>Child01 입니다.</h3>
            <button onClick={btnClick01}>눌려봐</button>
            <button onClick={addList}>추가</button>
            <hr/>
            <Child02/>
        </div>
    );
};

export default Child01;
```

Child02.jsx

```jsx
import Child03 from './Child03';

const Child02 = () => {
    return (
        <div style={{border:"1px green solid" , padding:"10px"}}>
            <h3>Child02 입니다.</h3>
            <Child03/>
        </div>
    );
};

export default Child02;
```

8253jang@daum.net
장희정

Child03.jsx

```jsx
import React from 'react';
import { useContext } from 'react';
import { GlobalContext } from '../src/Context01';

const Child03 = () => {
    const {btnClick02} = useContext(GlobalContext);
    return (
        <div style={{border:"1px orange solid" , padding:"10px"}}>
          <h3>Child03입니다.</h3>
          <button onClick={btnClick02}>클릭</button>
        </div>
    );
};

export default Child03;
```