

## 8. HOOK

작성일시: 2022년 5월 9일 오후 8:25

### 8.0 Hook이란?

함수 컴포넌트에서도 상태 관리를 할 수 있는 `useState`, 렌더링 직후 설정하는 `useEffect` 기능을 제공하여 다양한 작업을 할 수 있게 해준다.

### 8.1 `useState`

`useState`는 가장 기본적인 Hook이며, 함수 컴포넌트에서도 가변적인 상태를 지닐 수 있게 한다.

```
import React, { useState } from 'react'

const Counter = () => {
  const [value, setValue] = useState(0);
  return (
    <div>
      <p>
        현재 카운터 값은 <b>{value}</b>입니다.
      </p>
      <button onClick={() => setValue(value + 1)}>+1</button>
      <button onClick={() => setValue(value - 1)}>-1</button>
    </div>
  );
};

export default Counter;
```

```
01_src_app > src > COUNTER.js > default
1  import React, { useState } from 'react'
2
3  const Counter = () => {
4    const [value, setValue] = useState(0);
5    return (
6      <div>
7        <p>
8          현재 카운터 값은 <b>{value}</b>입니다.
9        </p>
10       <button onClick={() => setValue(value + 1)}>+1</button>
11       <button onClick={() => setValue(value - 1)}>-1</button>
12     </div>
13   )
14 }
15
16 export default Counter
```

현재 카운터 값은 -3입니다.

+1 -1

```
import React from 'react';
import Counter from './Counner';

const App = () => {
  return (
    <Counter/>
  );
};
export default App;
```

```
01_src_app > src > App.js > App
1  import React from 'react';
2  import Counter from './Counner';    "Counner": Unknown word.
3
4
5  const App = () => {
6
7      return (
8          <Counter/>
9      );
10 };
11 export default App;
```

현재 카운터 값은 2입니다.

### 8.1.1 useState를 여러 번 사용하기

Info.js

```
import React, { useState } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');

  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName}/>
        <input value={nickname} onChange={onChangeNickname}/>
      </div>
    </div>
  )
}
```

```

        <b>이름:</b>{name}
      </div>
    </div>
    <div>
      <b>닉네임:</b>{nickname}
    </div>
  </div>
);
};

export default Info;

```

```

src_app > src > Info.js > Info > onChangeNickname
1  import React, { useState } from 'react'
2
3  const Info = () => {
4    const [name, setName]=useState('');
5    const [nickname, setNickname] = useState('');
6
7    const onChangeName = e =>{
8      setName(e.target.value);
9    };
10   const onChangeNickname = e =>{
11     setNickname(e.target.value);
12   };
13   return (
14     <div>
15       <div>
16         <input value={name} onChange={onChangeName}/>
17         <input value={nickname} onChange={onChangeNickname}/>
18       </div>
19       <div>
20         <div>
21           <b>이름:</b>{name}
22         </div>
23         <div>
24           <b>닉네임:</b>{nickname}
25         </div>
26       </div>
27     </div>
28   );
29 };
30
31 export default Info;

```

김민준 velopert

이름:김민준  
닉네임:velopert

## App.js

```

import React from 'react';
import Info from './Info';

const App = ()=>{

  return (
    <Info/>
  );
};

export default App;

```

```

01_src_app > src > App.js > ...
1  import React from 'react';
2  import Info from './Info';
3
4
5  const App = () => {
6
7      return (
8          <Info />
9      );
10 };
11 export default App;

```

김민준

velopert

이름:김민준

닉네임:velopert

## 8.2 useEffect

useEffect는 리액트 컴포넌트가 렌더링(화면을 그린다)될 때 마다 특정 작업을 수행하도록 설정했다.

Info.js

```

import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');
  useEffect(()=>{
    console.log('렌더링이 완료되었습니다. ');
    console.log({
      name,
      nickname
    });
  });

  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName}/>
        <input value={nickname} onChange={onChangeNickname}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>

```

```

    </div>
  </div>
);
};

export default Info;

```

```

01_src_app > src > Info.js > (90) Info
1  import React, { useState, useEffect } from 'react'
2
3  const Info = () => {
4    const [name, setName]=useState('');
5    const [nickname, setNickname] = useState('');
6    useEffect(()=>{
7      console.log('렌더링이 완료되었습니다. ');
8      console.log({
9        name,
10       nickname
11      });
12    });
13
14    const onChangeName = e =>{
15      setName(e.target.value);
16    };
17    const onChangeNickname = e =>{
18      setNickname(e.target.value);
19    };
20    return (
21      <div>
22        <div>
23          <input value={name} onChange={onChangeName}/>
24          <input value={nickname} onChange={onChangeNickname}/>
25        </div>
26        <div>
27          <div>
28            <b>이름:</b>{name}
29          </div>
30          <div>
31            <b>닉네임:</b>{nickname}
32          </div>
33        </div>
34      </div>
35    );
36  };
37

```

김민준      감사합니다

이름:김민준  
닉네임:감사합니다



### 8.2.1 마운트될 때만 실행하고 싶을 때

useEffect에서 설정한 함수를 컴포넌트가 화면에 맨 처음 렌더링될 때만 실행하고, 업데이트 될때는 실행하지 않으려면, 함수의 두번째 파라미터를 빈 배열로 처리해주면 된다.

Info.js-useEffect

```

import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');
  useEffect(()=>{
    console.log('마운트가 될 때만 실행됩니다.')
  }, []);

  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>

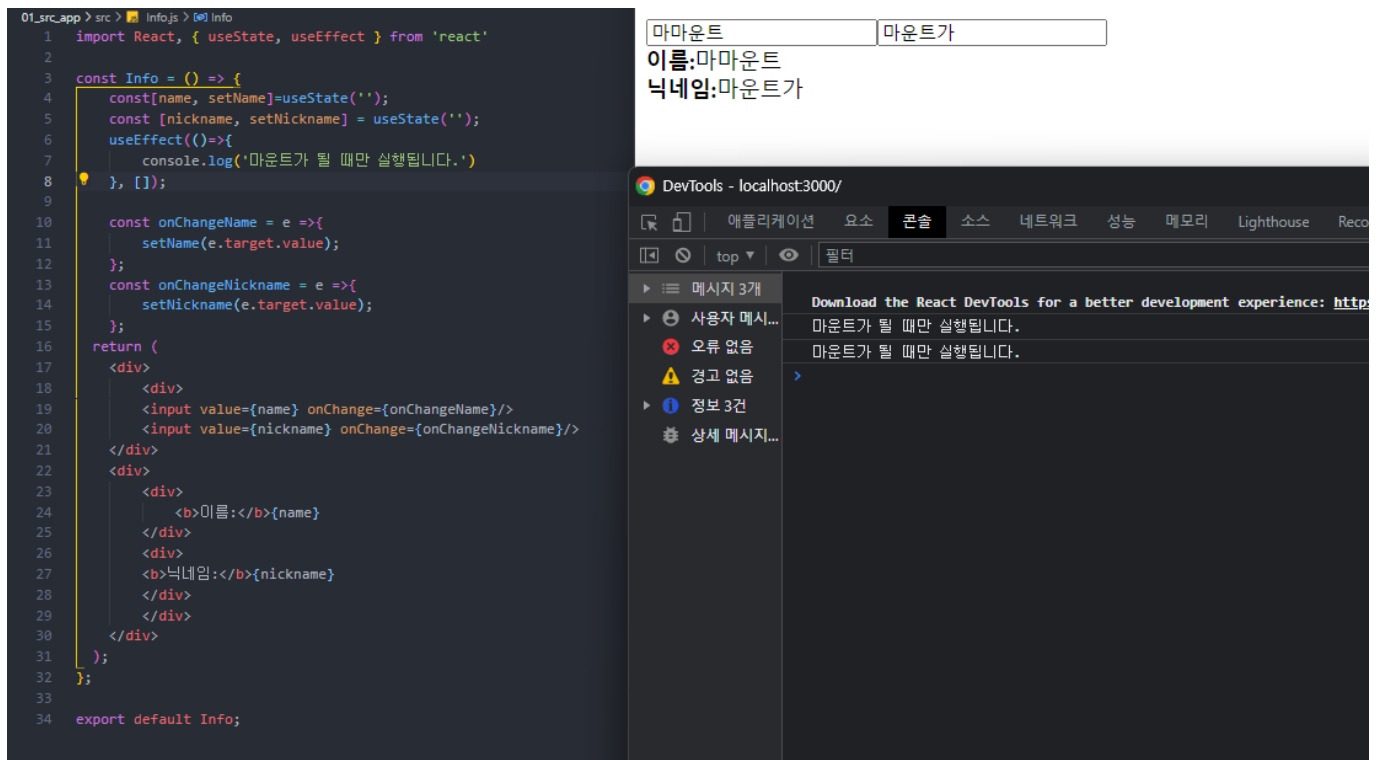
```

```

    <div>
      <input value={name} onChange={onChangeName}/>
      <input value={nickname} onChange={onChangeNickname}/>
    </div>
    <div>
      <div>
        <b>이름:</b>{name}
      </div>
      <div>
        <b>닉네임:</b>{nickname}
      </div>
    </div>
  </div>
);
};

export default Info;

```



## 8.2.2 특정 값이 업데이트 될 때만 실행하고 싶을 때

useEffect의 두 번째 파라미터로 전달되는 배열 안에 검사하고 싶은 값을 넣어주면 된다.

Info.js-useEffect

```

import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');
  useEffect(()=>{

```

```

    console.log(name)
  }, [name]));

  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName}/>
        <input value={nickname} onChange={onChangeNickname}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>
      </div>
    </div>
  );
};

export default Info;

```

```

import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');
  useEffect(()=>{
    console.log(name)
  }, [name]);

  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName}/>
        <input value={nickname} onChange={onChangeNickname}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>
      </div>
    </div>
  );
};

export default Info;

```

마운트

이름:마운트

닉네임:

DevTools - localhost3000/

애플리케이션 요소 콘솔 소스 네트워크 성능 메모리 Light

메시지 11개

사용자 메시...

오류 없음

경고 없음

정보 11건

상세 메시지...

Download the React DevTools for a better development experience

마

망

마

마우

마운

마운트

마운트

### 8.2.3 뒷정리하기

컴포넌트가 언마운트 되기 전이나 업데이트 되지 직전에 어떠한 작업을 수행하고 싶다면, `useEffect`에서 뒷정리 (`cleanup`) 함수를 반환해줘야 한다.

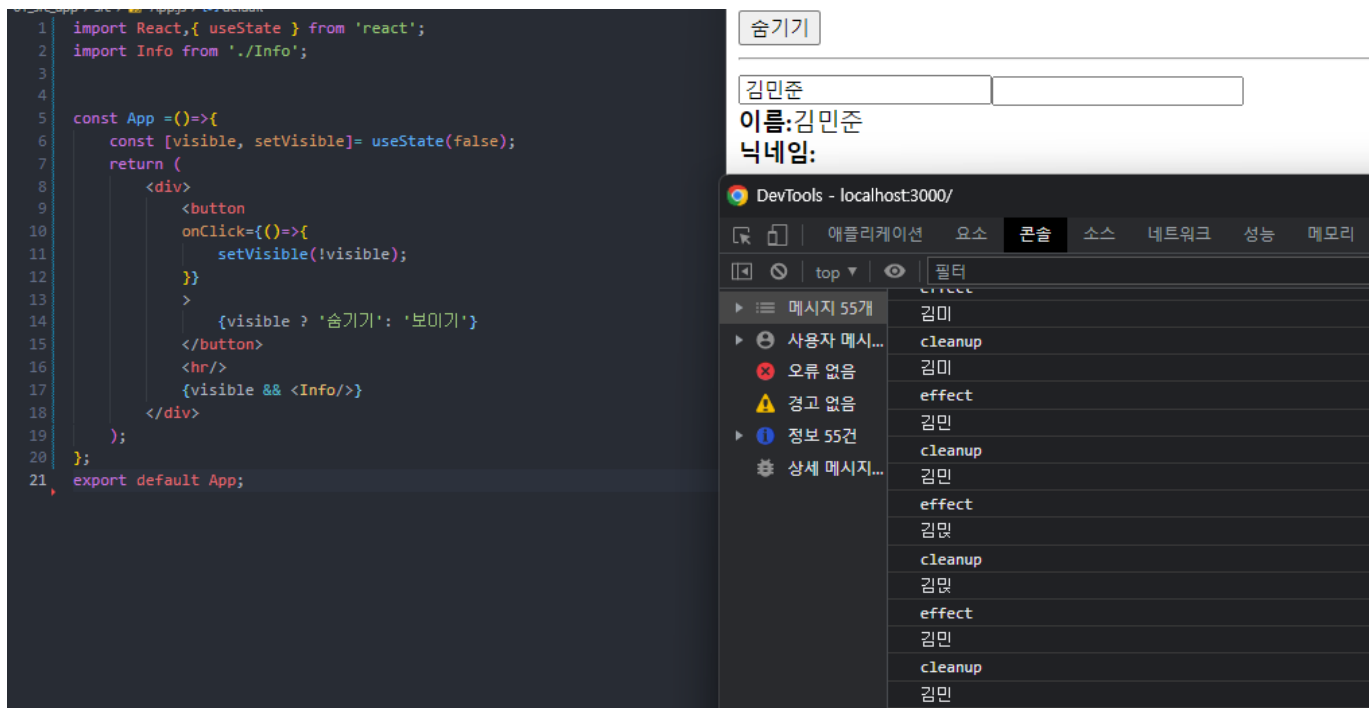
Info.js-useEffect

```
import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName]=useState('');
  const [nickname, setNickname] = useState('');
  useEffect(()=>{
    console.log('effect');
    console.log(name);
    return ()=>{
      console.log('cleanup');
      console.log(name);
    }
  }, [name])
  const onChangeName = e =>{
    setName(e.target.value);
  };
  const onChangeNickname = e =>{
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName}/>
        <input value={nickname} onChange={onChangeNickname}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>
      </div>
    </div>
  );
};

export default Info;
```

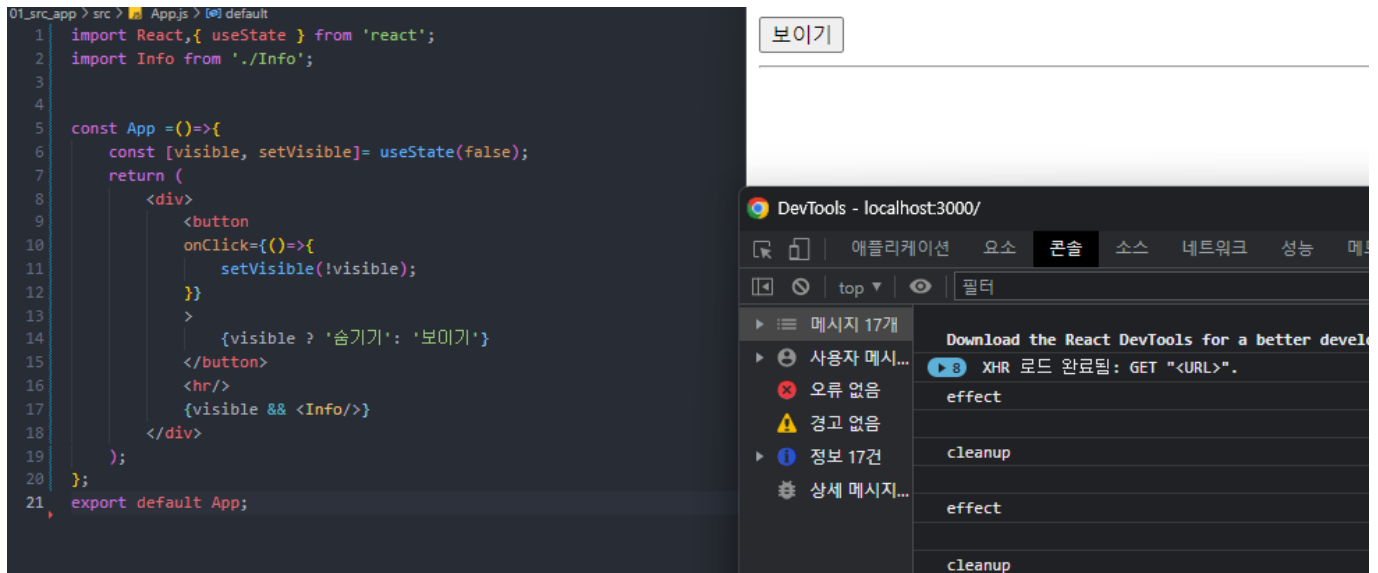




## App.js

```
import React,{ useState } from 'react';
import Info from './Info';

const App =()=>{
  const [visible, setVisible]= useState(false);
  return (
    <div>
      <button
        onClick={()=>{
          setVisible(!visible);
        }}
      >
        {visible ? '숨기기': '보이기'}
      </button>
      <hr/>
      {visible && <Info/>}
    </div>
  );
};
export default App;
```



## 언마운트 시에만 뒷정리 함수를 호출

Info.js

```

import React, { useState, useEffect } from 'react'

const Info = () => {
  const [name, setName] = useState('');
  const [nickname, setNickname] = useState('');
  useEffect(() => {
    console.log('effect');
    return () => {
      console.log('unmount');
    }
  }, [name])
  const onChangeName = e => {
    setName(e.target.value);
  };
  const onChangeNickname = e => {
    setNickname(e.target.value);
  };
  return (
    <div>
      <div>
        <input value={name} onChange={onChangeName} />
        <input value={nickname} onChange={onChangeNickname} />
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>
      </div>
    </div>
  )
}

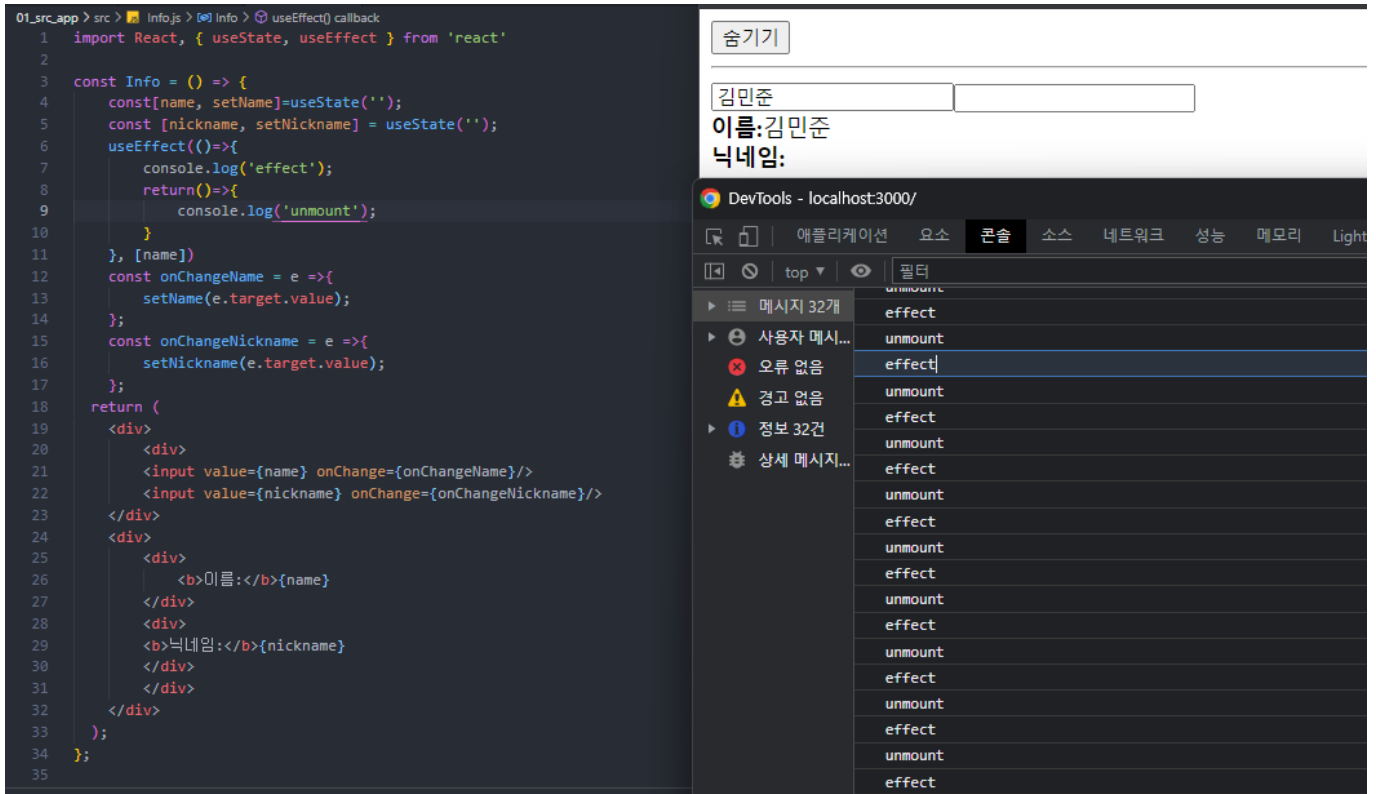
```

```

    );
  };

  export default Info;

```



## 8.3 useReducer

useReducer은 useState 보다 더 다양한 컴포넌트 상황에 따라 다양한 상태를 다른 값으로 업데이트를 할 수 있는 Hook이다.

Reducer은 현재 상태, 그리고 업데이트를 위해 필요한 정보를 담은 액션(action) 값을 전달받아 새로운 상태를 반환하는 함수이다. Reducer함수에선 새로운 상태를 만들 때는 반드시 불변성을 지켜줘야한다.

### 불변성이란?

상태 값에 대한 복사본을 생성한 후, 활용하여 상태 값을 갱신한다.

```

function reducer(state, action){
  return {...}; // 불변성을 지키며 업데이트한 새로운 상태를 반환한다.
}
//액션 값은 주로 다음과 같은 형태로 이루어짐
{
  type: 'INCREMENT',
  // 다른 값들이 필요하다면 추가로 들어감
}

```

### 8.3.1 카운터 구현하기

## Counner.js

```
import React, {useReducer} from 'react'

function reducer(state, action) {
  // state={value=0}, action={type:'INCREMENT'} state와 action은 JSON객체이다.
  // action.type에 따라 다른 작업 수행
  switch(action.type){
    case 'INCREMENT':
      return{ value: state.value +1 };
    case 'DECRMENT':
      return { value: state.value -1};
    default:
      // 아무 것도 해당되지 않을 경우
      return state;
  }
}

const Counter = () => {
  const [state, dispatch] = useReducer(reducer,{value:0});
  return (
    <div>
      <p>
        현재 카운터 값은 <b>{state.value}</b>입니다.
      </p>
      <button onClick={() => dispatch({type: 'INCREMENT'})}>>+1</button>
      <button onClick={() => dispatch({type: 'DECREMENT'})}>>-1</button>
    </div>
  );
};

export default Counter;
```

```

01_src_app > src > Counterjs > ...
1  import React, {useReducer} from 'react'
2
3  function reducer(state, action) { // state={value=0}, action={type: 'INCR
4    // action.type에 따라 다른 작업 수행
5    switch(action.type){
6      case 'INCREMENT':
7        return{ value: state.value +1 };
8      case 'DECREMENT': "DECRMENT": Unknown word.
9        return { value: state.value -1};
10     default:
11       // 아무 것도 해달라지 않을 경우
12       return state;
13   }
14 }
15
16 const Counter = () => {
17   const [state, dispatch] = useReducer(reducer,{value:0});
18   return (
19     <div>
20       <p>
21         현재 카운터 값은 <b>{state.value}</b>입니다.
22       </p>
23       <button onClick={()=> dispatch({type: 'INCREMENT'})}>+1</button>
24       <button onClick={()=> dispatch({type: 'DECREMENT'})}>-1</button>
25     </div>
26   );
27 }
28 export default Counter;

```

현재 카운터 값은 2입니다.

+1 -1

## App.js

```

import React from 'react';
import Counter from './Counter';

const App = () => {
  return <Counter/>
}

export default App

```

## 8.3.2 인풋 상태 관리하기

### Info.js

```

import React, { useReducer } from 'react'

function reducer(state, action){
  //action 값은 <input name="name">, <input name = "nickname"> 중 이벤트가 발생한 하나.
  return{
    ...state, // 기본 값 그대로 복사
    [action.name]:action.value // action.value는 input 태그 입력값
  }
  // action에 들어오는 name에 따라서 JSON 객체가 변경됨
  // JSON의 key를 동적으로 명시할 수 있음
  /*=> ex) const key ="b"
      const data={
        [key]: 100
      }
  */
}

```


```

    }; => {b:100}

    */
  };
}
const Info = () => {
  const [state, dispatch] = useReducer(reducer,{
    name:'',
    nickname: ''
  });
  const {name, nickname} = state;
  const onChange = e =>{
    dispatch(e.target);
  };
  return (
    <div>
      <div>
        <input name="name" value={name} onChange={onChange}/>
        <input name="nickname" value={nickname} onChange={onChange}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>
            {nickname}
        </div>
      </div>
    </div>
  );
};

export default Info;

```



The screenshot shows a code editor with the following code on the left and its rendered output on the right:

```

01_src_app > src > Info.js > Info > onChange
9  const Info = () => {
10   const [state, dispatch] = useReducer(reducer,{
11     name:'',
12     nickname: ''
13   });
14   const {name, nickname} = state;
15   const onChange = e =>{
16     dispatch(e.target);
17   };
18   return (
19     <div>
20       <div>
21         <input name="name" value={name} onChange={onChange}/>
22         <input name="nickname" value={nickname} onChange={onChange}/>
23       </div>
24       <div>
25         <div>
26           <b>이름:</b>{name}
27         </div>
28         <div>
29           <b>닉네임:</b>
30             {nickname}
31         </div>
32       </div>
33     </div>
34   );
35 };
36
37 export default Info;

```

The rendered output on the right is:

이승아	seung lee
-----	-----------

이름:이승아  
닉네임:seung lee

```
import React from 'react';
import Info from './Info';

const App = () => {
  return <Info/>
};

export default App;
```

## 8.4 useMemo

useMemo를 사용하면 함수 컴포넌트 내부에서 발생하는 연산을 최적화 할 수 있다.

Average.js

```
import React,{useState} from 'react'

const getAverage = numbers =>{
  console.log('평균값 계산 중...');
  if(numbers.length === 0) return 0;
  const sum = numbers.reduce((a,b)=>a+b);
  return sum/ numbers.length;
};

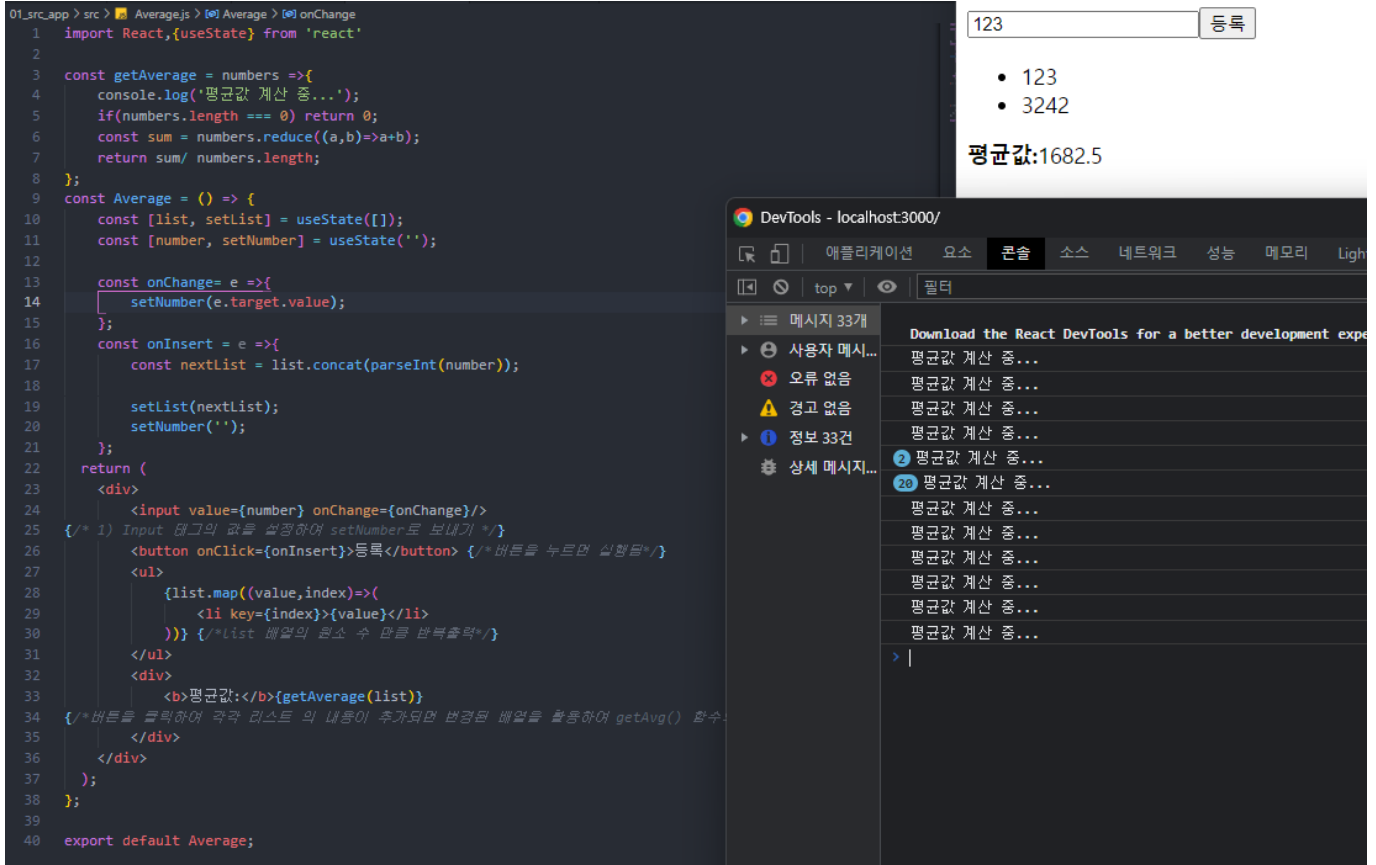
const Average = () => {
  const [list, setList] = useState([]);
  const [number, setNumber] = useState('');

  const onChange= e =>{
    setNumber(e.target.value);
  //2) const setNumber 값을 변경해줌
  };
  const onInsert = e =>{
    const nextList = list.concat(parseInt(number));
    //기존의 list 배열에 number의 값을 추가한 새로운 복사본 을 생성함
    setList(nextList); //새롭게 생성된 복사본을 list에 적용함
    setNumber('');
  };
  return (
    <div>
      <input value={number} onChange={onChange}/>
      { /* 1) Input 태그의 값을 설정하여 setNumber로 보내기 */}
      <button onClick={onInsert}>등록</button> { /*버튼을 누르면 실행됨*/}
      <ul>
        {list.map((value,index)=>(
          <li key={index}>{value}</li>
        ))} { /*list 배열의 원소 수 만큼 반복출력*/}
      </ul>
      <div>
        <b>평균값:</b>{getAverage(list)}
      </div>
    </div>
  )
};

{ /*버튼을 클릭하여 각각 리스트 의 내용이 추가되면 변경된 배열을 활용하여 getAvg() 함수의
```

```
리턴값을 출력함. -> list 배열의 변경에 따라 실시간 저장됨*/}
    </div>
  </div>
);
};

export default Average;
```



## App.js

```
import React from 'react';
import Average from './Average';

const App = () => {
  return <Average/>
};

export default App;
```

**useMemo Hook**을 이용하여 특정 값이 변경시 연산을 실행

## Average.js

```
import React,{useState, useMemo} from 'react'
```



```

const getAverage = numbers =>{
  console.log('평균값 계산 중...');
  if(numbers.length === 0) return 0;
  const sum = numbers.reduce((a,b)=>a+b);
  return sum/ numbers.length;
};

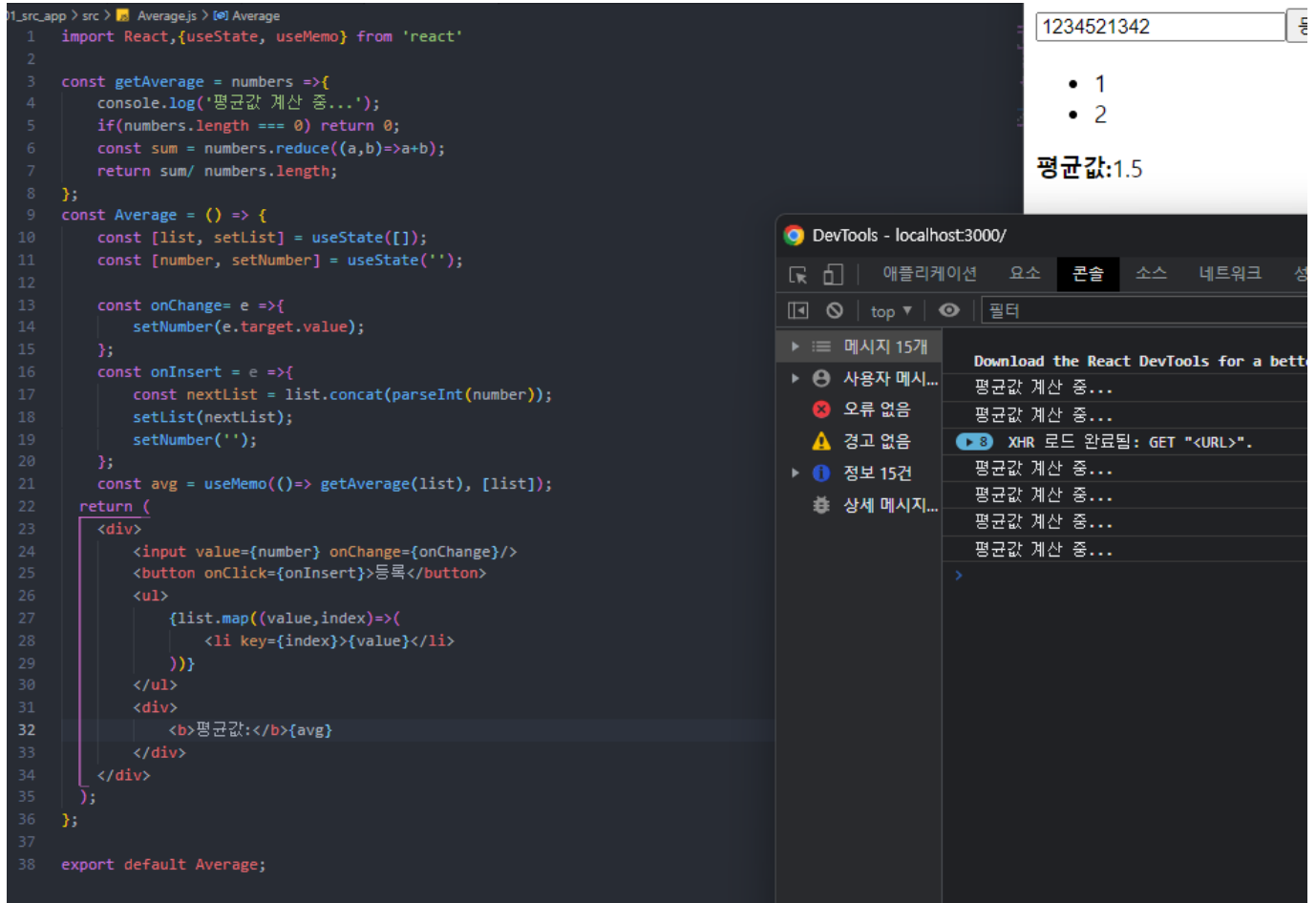
const Average = () => {
  const [list, setList] = useState([]);
  const [number, setNumber] = useState('');

  const onChange= e =>{
    setNumber(e.target.value);
  };
  const onInsert = e =>{
    const nextList = list.concat(parseInt(number));
    setList(nextList);
    setNumber('');
  };

  const avg = useMemo(()=> getAverage(list), [list]);
  //getAverage 값은 avg에 리턴된다.
  // 상태값이 list에 변경되었을 때만, getAverage()를 호출한다.
  return (
    <div>
      <input value={number} onChange={onChange}/>
      <button onClick={onInsert}>등록</button>
      <ul>
        {list.map((value,index)=>(
          <li key={index}>{value}</li>
        ))}
      </ul>
      <div>
        <b>평균값:</b>{avg}
      </div>
    </div>
  );
};

export default Average;

```



## 8.5 useCallback

useCallback은 useMemo와 상당히 비슷한 함수이다. 주로 렌더링 성능을 최적화해야 하는 상황에서 사용. 이 Hook을 사용하면 함수를 재사용할 수 있다.

Average.js

```

import React, {useState, useMemo, useCallback} from 'react'

const getAverage = numbers => {
  console.log('평균값 계산 중...');
  if(numbers.length === 0) return 0;
  const sum = numbers.reduce((a,b)=>a+b);
  return sum / numbers.length;
};

const Average = () => {
  const [list, setList] = useState([]);
  const [number, setNumber] = useState('');

  const onChange = useCallback(e => {
    setNumber(e.target.value);
  }, []); // 컴포넌트가 처음 렌더링될 때만 함수 생성
  const onInsert = useCallback(() => {
    const nextList = list.concat(parseInt(number));
    setList(nextList);
    setNumber('');
  }, [list]);

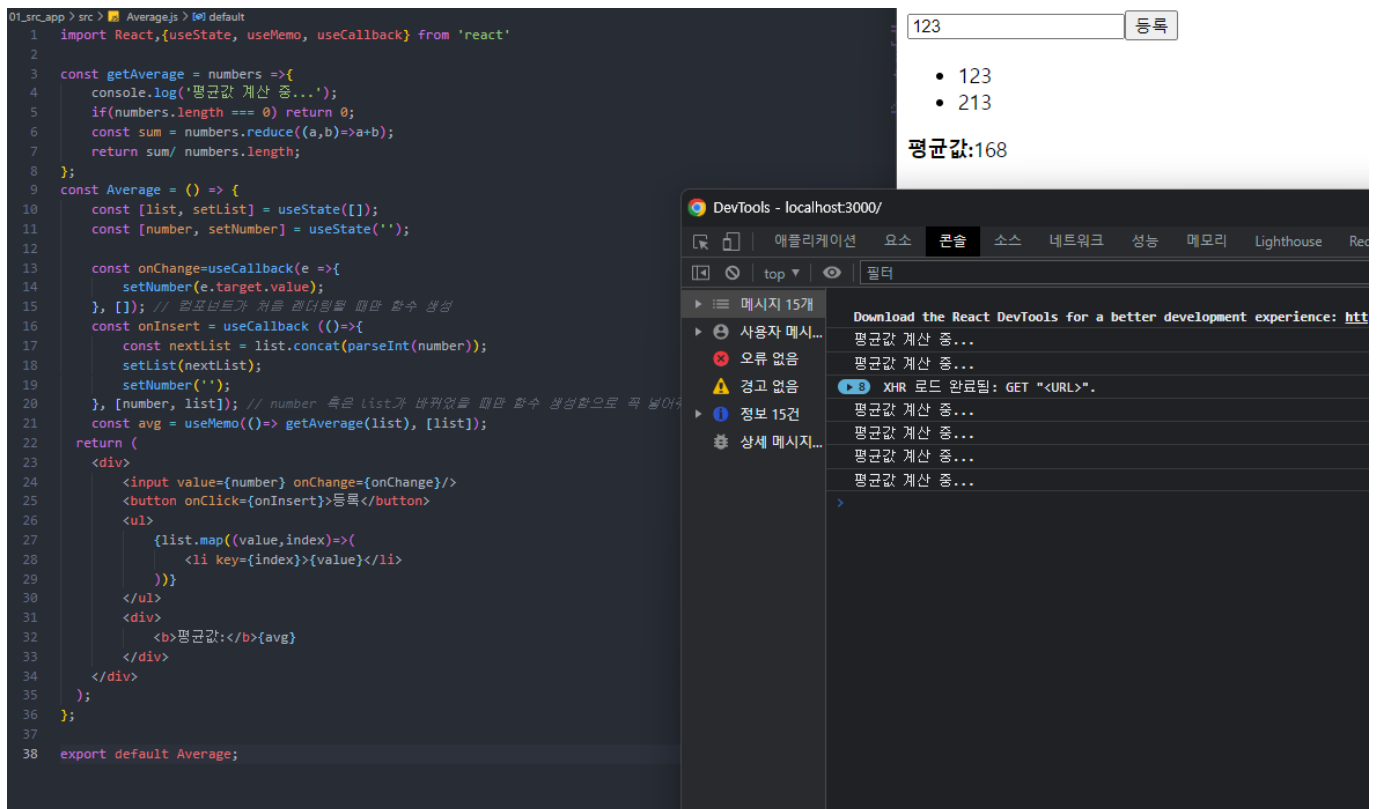
```

```

    }, [number, list]); // number 혹은 list가 바뀌었을 때만 함수 생성함으로 꼭 넣어줘
    야함
    const avg = useMemo(() => getAverage(list), [list]);
    return (
      <div>
        <input value={number} onChange={onChange}/>
        <button onClick={onInsert}>등록</button>
        <ul>
          {list.map((value, index) => (
            <li key={index}>{value}</li>
          ))}
        </ul>
        <div>
          <b>평균값:</b>{avg}
        </div>
      </div>
    );
  };

  export default Average;

```



첫 번째 파라미터에는 생성하고 싶은 함수를 넣고, 두 번째 파라미터에는 배열을 넣으면된다. 이 배열에는 어떤 값이 바뀌었을 때 함수를 새로 생성해야 하는지 명시해야한다. onChange처럼 비어있는 배열을 넣으면 컴포넌트가 렌더링 될 때 만들었던 함수를 재사용한다. onInsert 처럼 배열 안에 number와 list를 넣게 되면 인풋 내용이 바뀌거나 새로운 항목이 추가될 때 새로 만들어진 함수를 사용한다.

## 8.6 useRef

Average.js

```

import React,{useState, useMemo, useCallback, useRef} from 'react'

const getAverage = numbers =>{
  console.log('평균값 계산 중...');
  if(numbers.length === 0) return 0;
  const sum = numbers.reduce((a,b)=>a+b);
  return sum/ numbers.length;
};

const Average = () => {
  const [list, setList] = useState([]);
  const [number, setNumber] = useState('');
  const inputE1 = useRef(null);

  const onChange=useCallback(e =>{
    setNumber(e.target.value);
  }, []); // 컴포넌트가 처음 렌더링될 때만 함수 생성
  const onInsert = useCallback (()=>{
    const nextList = list.concat(parseInt(number));
    setList(nextList);
    setNumber('');
  }, [number, list]); // number 혹은 list가 바뀌었을 때만 함수 생성
  const avg = useMemo(()=> getAverage(list), [list]);
  return (
    <div>
      <input value={number} onChange={onChange} ref={inputE1}/>
      <button onClick={onInsert}>등록</button>
      <ul>
        {list.map((value,index)=>(
          <li key={index}>{value}</li>
        ))}
      </ul>
      <div>
        <b>평균값:</b>{avg}
      </div>
    </div>
  );
};

export default Average;

```

```

01_src_app > src > Average.js > Average
1  import React,{useState, useMmo, useCallback, useRef} from 'react'
2
3  const getAverage = numbers =>{
4    console.log('평균값 계산 중...');
5    if(numbers.length === 0) return 0;
6    const sum = numbers.reduce((a,b)=>a+b);
7    return sum/ numbers.length;
8  };
9  const Average = () => {
10   const [list, setList] = useState([]);
11   const [number, setNumber] = useState('');
12   const inputE1 = useRef(null);
13
14   const onChange=useCallback(e =>{
15     setNumber(e.target.value);
16   }, []); // 컴포넌트가 처음 렌더링될 때만 함수 생성
17   const onInsert = useCallback (()=>{
18     const nextList = list.concat(parseInt(number));
19     setList(nextList);
20     setNumber('');
21   }, [number, list]); // number 혹은 list가 바뀌었을 때만 함수 생성
22   const avg = useMmo(()=> getAverage(list), [list]);
23   return (
24     <div>
25       <input value={number} onChange={onChange} ref={inputE1}/>
26       <button onClick={onInsert}>등록</button>
27       <ul>
28         {list.map((value,index)=>{
29           <li key={index}>{value}</li>
30         })}
31       </ul>
32       <div>
33         <b>평균값:</b>{avg}
34       </div>
35     </div>
36   );
37 };
38
39 export default Average;

```

등록

- 123
- 123123

평균값:61623

### 8.6.1 로컬 변수 사용하기

#### 예시코드

```

import { useRef } from "react";

const RefSample = () => {
  const id = useRef(1);
  const setId = (n)=>{
    id.current=n;
  }
  const printId =()=>{
    console.log(id.current);
  }
  return(
    <div>
      refsample
    </div>
  );
};

export default RefSample;

```

```

01_src_app > src > Exjs > ...
1  import { useRef } from "react";
2
3  const RefSample = () => {
4    const id = useRef(1);
5    const setId = (n)=>{ 'setId' is assigned a value but never used.
6      id.current=n;
7    }
8    const printId =()=>{ 'printId' is assigned a value but never used.
9      console.log(id.current);
10   }
11   return(
12     <div>
13       refsample "refsample": Unknown word.
14     </div>
15   );
16 };
17
18 export default RefSample;
19

```

## 8.7 커스텀 Hooks 만들기

여러 컴포넌트에서 비슷한 기능을 공유할 경우, 이를 자신만의 Hook으로 작성하여 로직을 재사용할 수 있다.

UseInput.js

```

import {useReducer} from 'react';

function reduce(state, action){
  return{
    ...state,
    [action.name]: action.value //function을 통해 리턴되는 값은 state의 상태값이
    됨
  };
} //dispatch 값이 실행되기 때문에 여기를 통해 값이 갱신됨

export default function UseInput (initialForm) {
  const [state, dispatch] = useReducer(reduce, initialForm);

  //initialForm은 action에들어감
  const onChange= e =>{
    dispatch(e.target); // 1) 이벤트가 발생한 주체로 (Reducer을 통해)initialForm
    리턴
  };
  return [state,onChange]; // 여기가 실행되면 const onChage 부분이 실행됨
  // state는 값, onChange는 함수를 배열로 묶어서 리턴해줌
}

```

```
01_src_app > src > UseInputs > UseInput
1 import {useReducer} from 'react';
2
3 function reduce(state, action){
4   return{
5     ...state,
6     [action.name]: action.value
7   };
8 }
9
10 export default function UseInput (initialForm) {
11   const [state, dispatch] = useReducer(reduce, initialForm);
12   const onChange= e =>{
13     dispatch(e.target);
14   };
15   return [state,onChange];
16 }
17
```

이름:  
닉네임:

## Info.js

```
import UseInput from './UseInput';

const Info = () => {
  const [state, onChange] = UseInput({
    name:'',
    nickname:''
  });
  const { name, nickname } = state;
  return (
    <div>
      <div>
        <input name="name" value={name} onChange={onChange}/>
        { /* 1) input 태그 내용이 변경되면 return
[state,onChange]가 실행됨 */}
        <input name="nickname" value={nickname} onChange={onChange}/>
      </div>
      <div>
        <div>
          <b>이름:</b>{name}
        </div>
        <div>
          <b>닉네임:</b>{nickname}
        </div>
      </div>
    </div>
  );
};

export default Info;
```

```
01_src_app > src > Info.js > 001 Info
1 import UseInput from './UseInput';
2
3 const Info = () => {
4   const [state, onChange] = UseInput({
5     name: '',
6     nickname: ''
7   });
8   const { name, nickname } = state;
9   return (
10    <div>
11      <div>
12        <input name="name" value={name} onChange={onChange}/>
13        <input name="nickname" value={nickname} onChange={onChange}/>
14      </div>
15      <div>
16        <div>
17          <b>이름:</b>{name}
18        </div>
19        <div>
20          <b>닉네임:</b>{nickname}
21        </div>
22      </div>
23    </div>
24  );
25 };
26
27 export default Info;
```

이름:  
닉네임: