React

◆ Hooks - 1

정수아

Contents

Hooks

Hooks

❖ Hooks란?

- 리액트 버전 16.8에 새로 도입된 기능
- 함수형 컴포넌트에서 상태 관리를 할 수 있는 기능 제공

- 종류
  - useState
  - useEffect
  - useReducer
  - useMemo
  - useCallback
  - useRef

02

useState()

useState()

❖ useState()

- 가장 기본적인 Hook
- 함수형 컴포넌트가 가변적인 상태를 지닐 수 있도록 해 줌

- 형태

```
const [state, setState] = useState(초기값);
```

- 예시

```
const [number, setNumber] = useState(3);
```

  - 현재 number 값 = 3
  - number 값 변경 = setNumber(6);

[실습] Counter

❖ App.js

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

function App() {
  return (
    <>
      <Counter/>
    </>
  );
};

export default App;
```

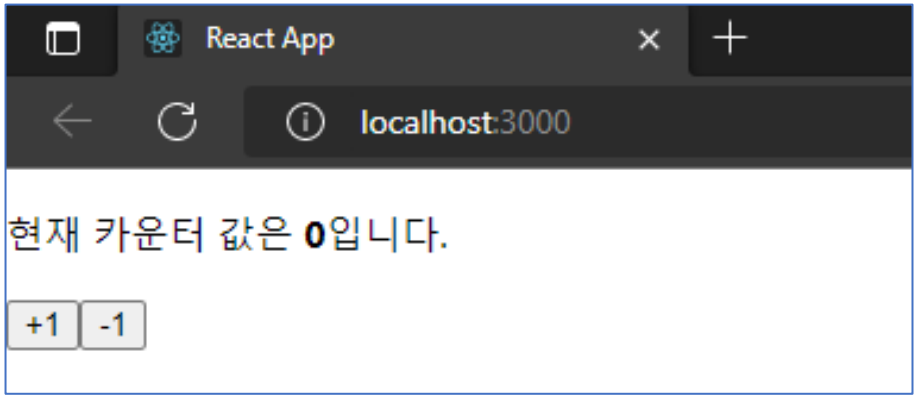
[실습] Counter

❖ Counter.js

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

function Counter(){
    const [value, setValue] = useState(0);

    function numUp(){
        setValue(value+1);
    }

    function numDown(){
        setValue(value-1);
    }

    return (
        <div>
            <p>현재 카운터 값은 <b>{value}</b>입니다.</p>
            <button onClick={numUp}>+1</button>
            <button onClick={numDown}>-1</button>
        </div>
    );
};

export default Counter;
```

[실습] Counter

❖ 실행 결과



[실습] AddName

❖ App.js

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

function App() {
  return (
    <>
      <AddName/>
    </>
  );
};

export default App;
```

[실습] AddName

❖ AddName.js

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

function AddName() {
    const [names, setNames] = useState(['정수아', '리액트’]);
    const [input, setInput] = useState('');

    function InputChange() { }
    function uploadInput() { }

    return (
        <div>
            <input type='text' onChange={InputChange}/>
            <button onClick={uploadInput}>추가</button>
            <div>
              { names.map((name, idx) => (<p key={idx}>{name}</p>)) }
            </div>
        </div>
    );
};

export default AddName;
```

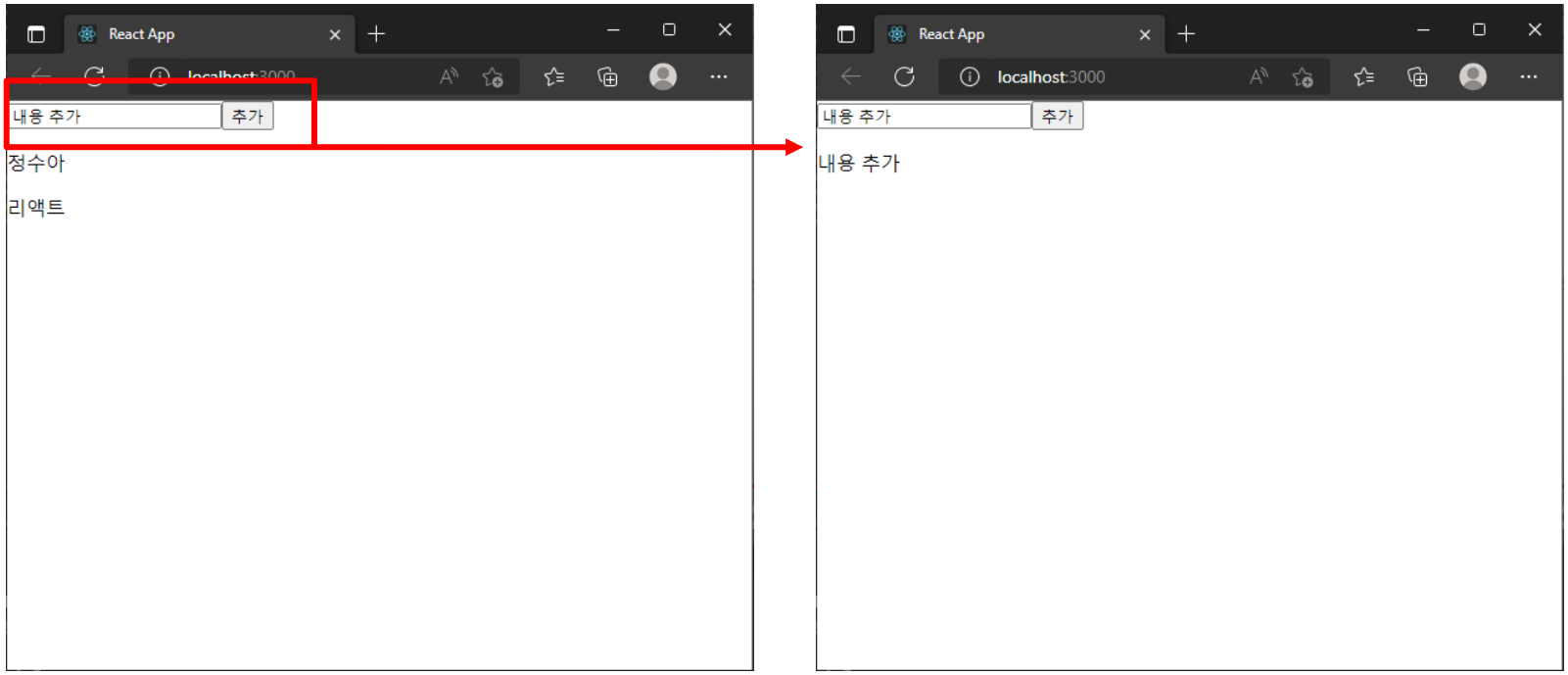
[실습] AddName

❖ AddName.js

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

function AddName() {
    const [names, setNames] = useState(['정수아', '리액트’]);
    const [input, setInput] = useState('');

    function InputChange(e) {
        setInput(e.target.value);
    }

    function uploadInput() {
        setNames([input])
    }


    // 생략
};

export default AddName;
```

[실습] AddName

❖ 실행 결과



[실습] AddName

❖ **uploadInput() 함수 수정**

- state의 기존 값을 유지하면서 새로운 값을 추가하려면 setState()의 콜백 함수에 prevState 값을 전달해서 유지해야 함

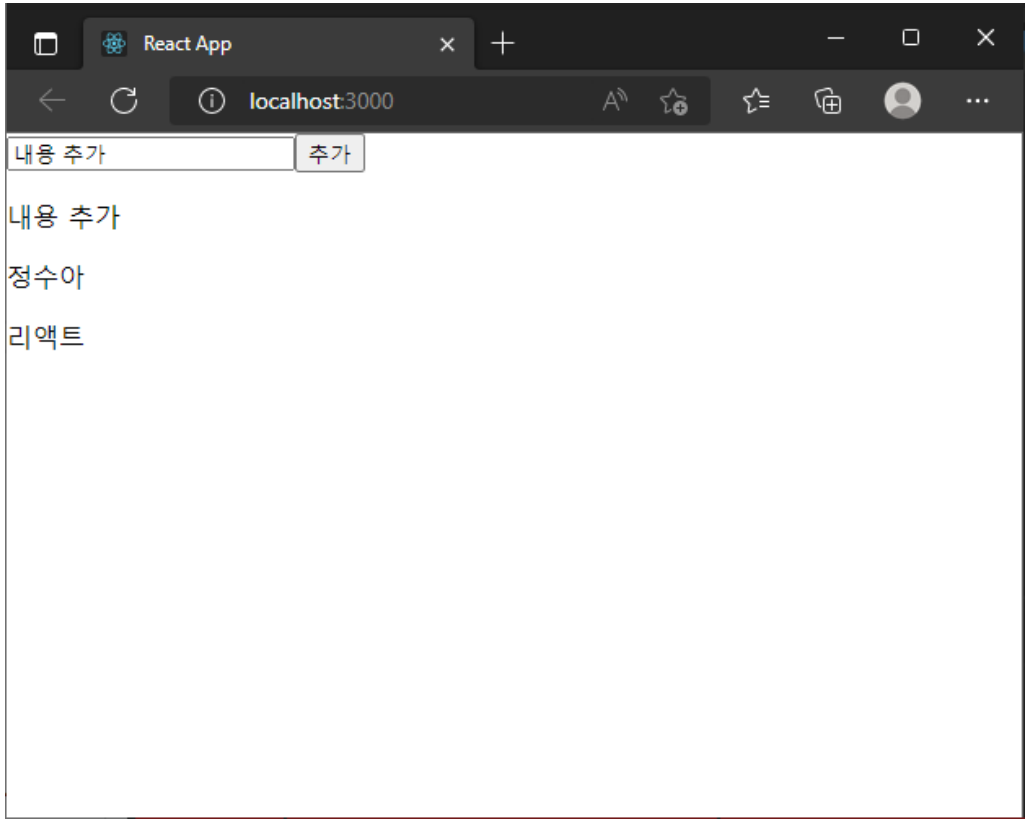
```
function uploadInput() {
    setNames((prevState) => [input, ...prevState]);
}
```



```
function uploadInput() {
    setNames(function(prevState) {
                    return [input, ...prevState]
            });
}
```

[실습] AddName

❖ 실행 결과



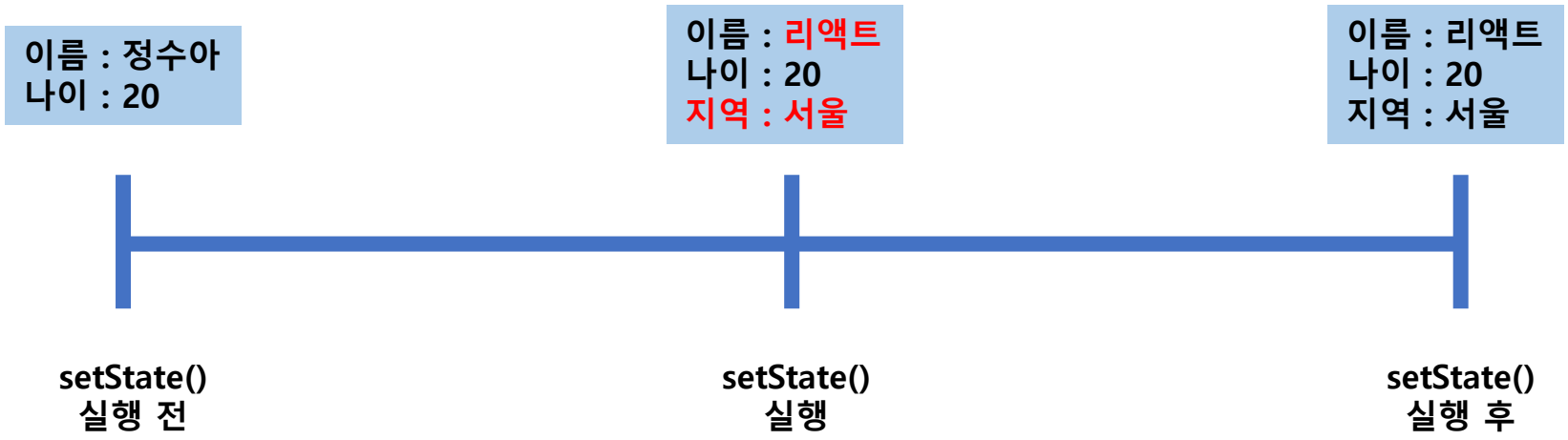
setState() 함수

❖ setState() 함수 동작 과정

setState() 함수

❖ **setState() 함수의 인자로 state를 전달 시 동작 과정**

- 이전 state와 새로운 state를 비교하여 바뀐 데이터만 업데이트
- 변경되지 않은 값은 그대로 유지함



useState() 성능 최적화

❖ **성능 최적화**

- useState() 함수의 인자에 초기값을 지정한 경우
  - state 값이 업데이트 될 때마다 초기값이 계속해서 호출됨
  - 만약 초기값에 복잡한 계산식이 있다면 성능 저하 문제 발생

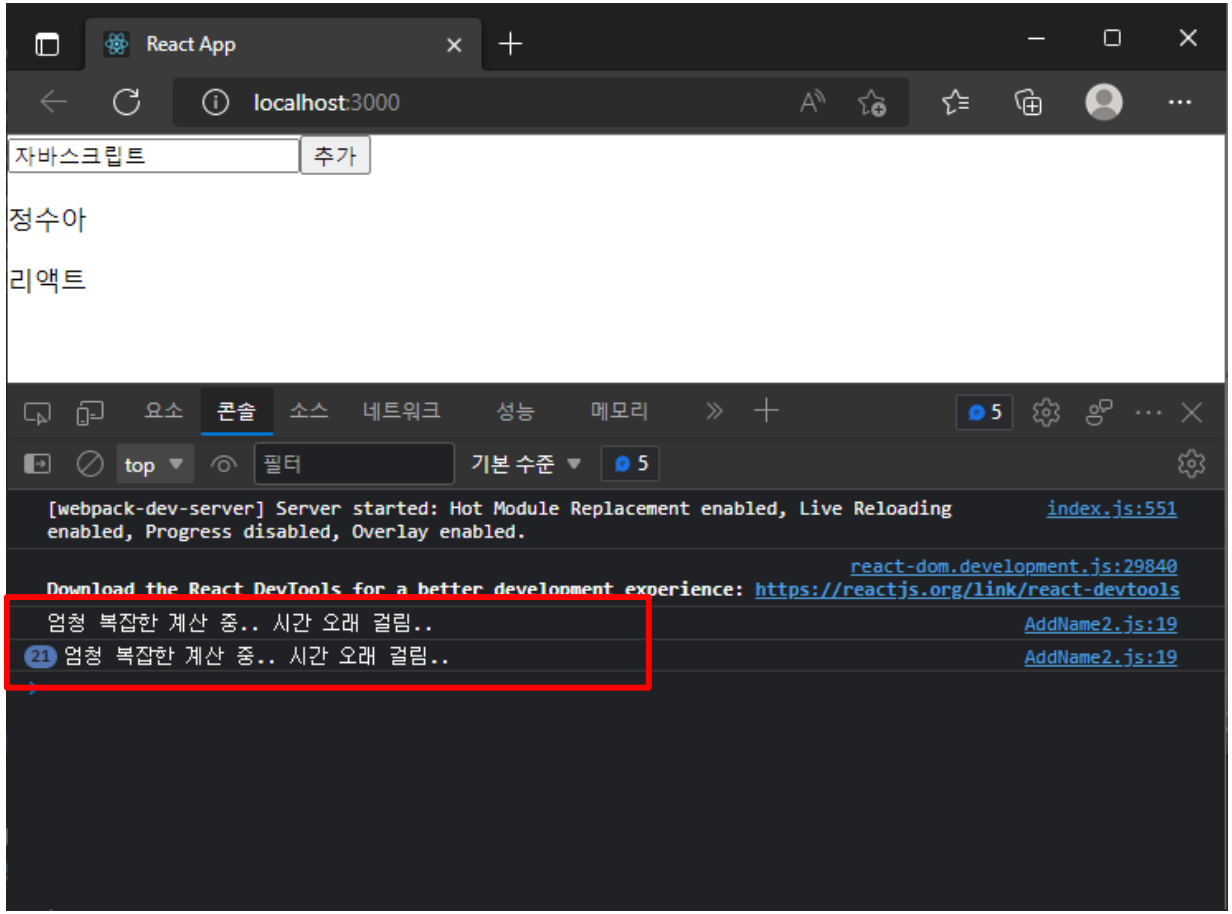
useState() 성능 최적화

❖ **AddName.js 수정**

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

function AddName() {
    const [names, setNames] = useState(heavyWork());

    function heavyWork() {
       for (let i = 0; i < 1000; i++) {
          console.log("엄청 복잡한 계산 중.. 시간 오래 걸림..");
       }

       return ["정수아", "리액트"];
    }

    // 생략
};

export default AddName;
```

useState() 성능 최적화

❖ 실행 결과



useState() 성능 최적화

❖ **문제점**
- state가 업데이트 될 때마다 heavyWork()가 계속 호출
- 초기값은 최초 한번만 호출되도록 수정해야 함

❖ **해결 방법**
- useState()함수의 인자로 콜백 함수를 넣어줌

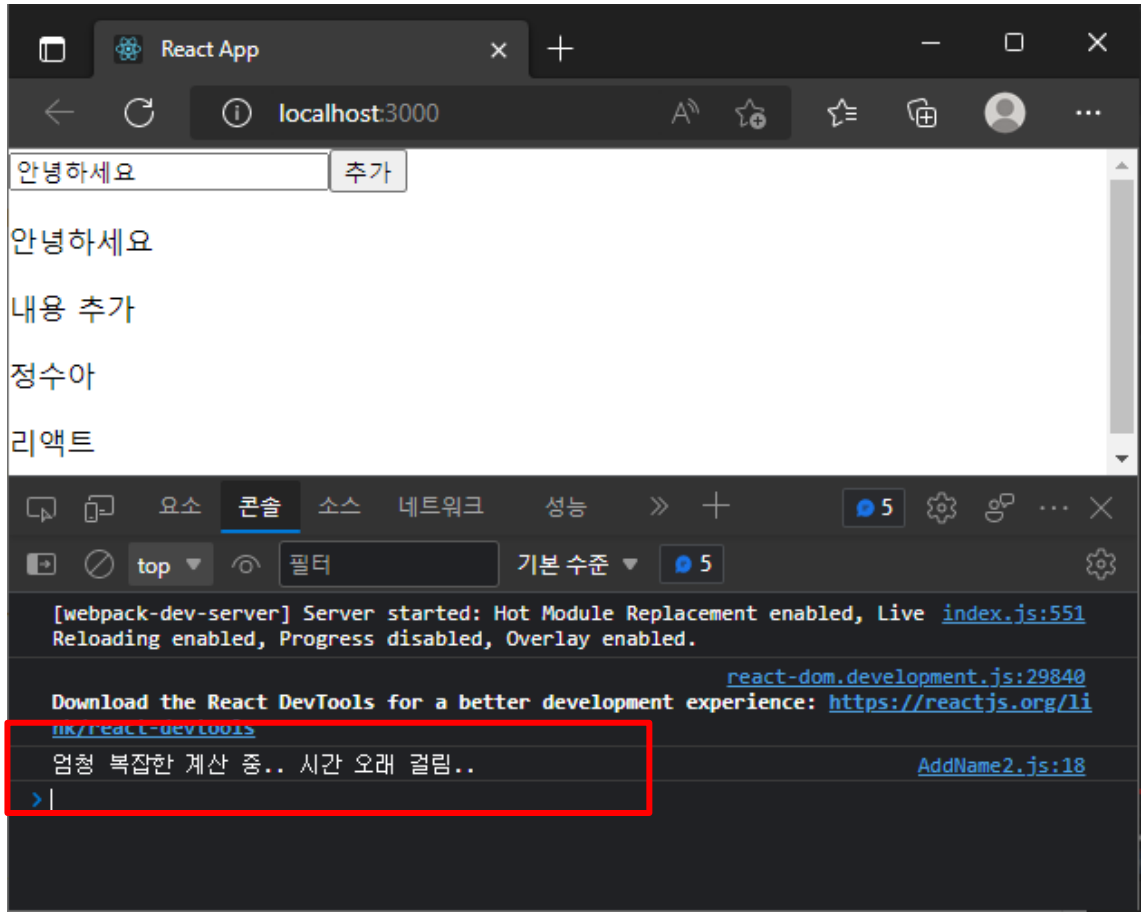
useState() 성능 최적화

❖ **AddName.js 수정**

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

function AddName() {
    const [names, setNames] = useState(() => heavyWork());

    function heavyWork() {
       for (let i = 0; i < 1000; i++) {
          console.log("엄청 복잡한 계산 중.. 시간 오래 걸림..");
       }

       return ["정수아", "리액트"];
    }

    // 생략
};

export default AddName;
```

useState() 성능 최적화

❖ 실행 결과



03

useEffect()

useEffect()

❖ **useEffect()**

- 리액트 컴포넌트가 렌더링 될 때마다 특정 작업을 수행하도록 설정해주는 Hook
- 최초에 한번 실행하게 하고 싶은 작업을 작성할 때 주로 사용
  - 예) fetch()를 이용한 네트워크 통신 연결

useEffect()

❖ useEffect()

useEffect()

❖ useEffect() 구조

- useEffect()의 매개변수에 콜백함수만 있는 경우
  - 컴포넌트가 렌더링 될 때마다 실행 됨

```
useEffect(() => {
    // 작업
});
```

- useEffect()의 매개변수에 콜백함수, 배열이 있는 경우
  - 컴포넌트가 처음 렌더링 될 때 실행
  - value 값이 변경 되었을 때 실행

```
useEffect(() => {
    // 작업
}, [value]);
```

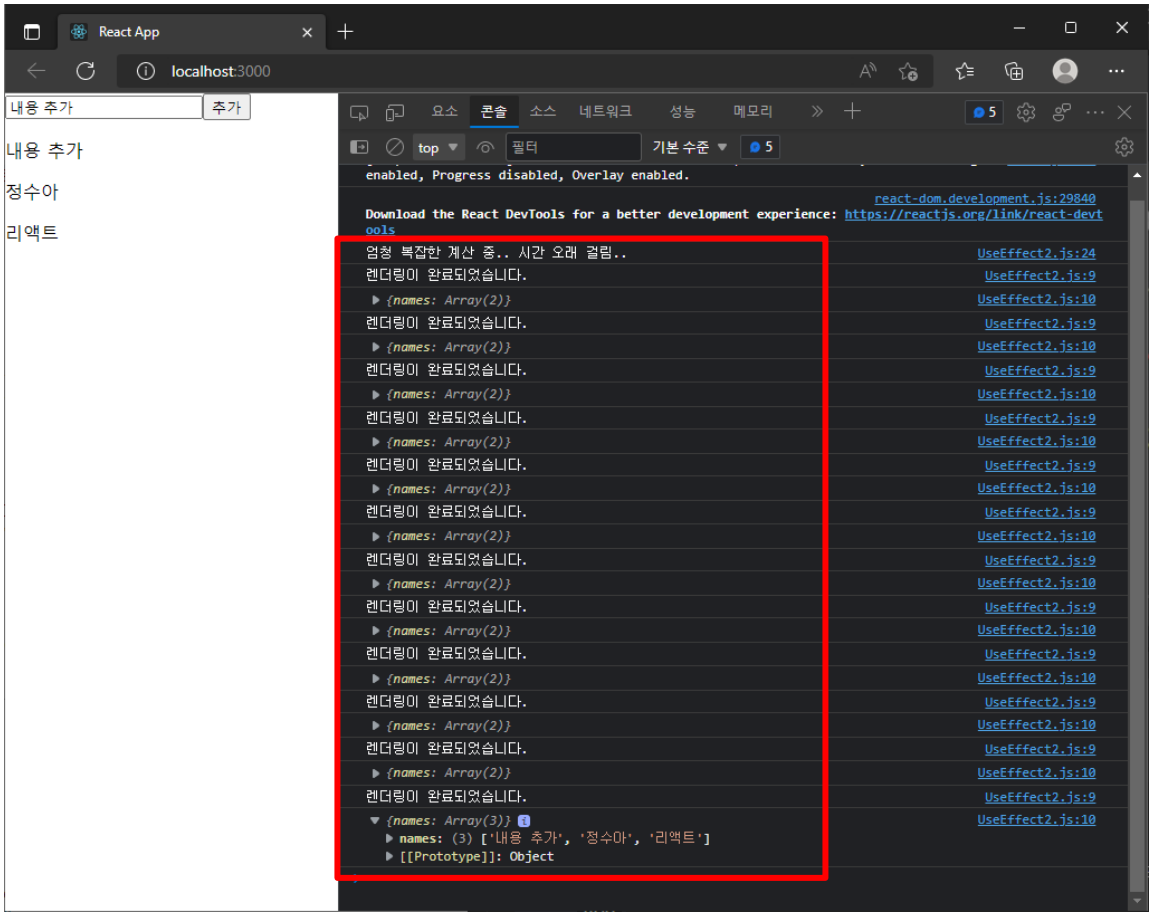
[실습] useEffect()

❖ AddName.js 수정

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

function AddName() {
    const [names, setNames] = useState(() => heavyWork());
    const [input, setInput] = useState('');

    // 생략

    useEffect(()=>{
        console.log("렌더링이 완료되었습니다.");
        console.log({names});
    })

    return (
         // 생략
    );
};

export default AddName;
```

[실습] useEffect()

❖ 실행 결과



[실습] useEffect()

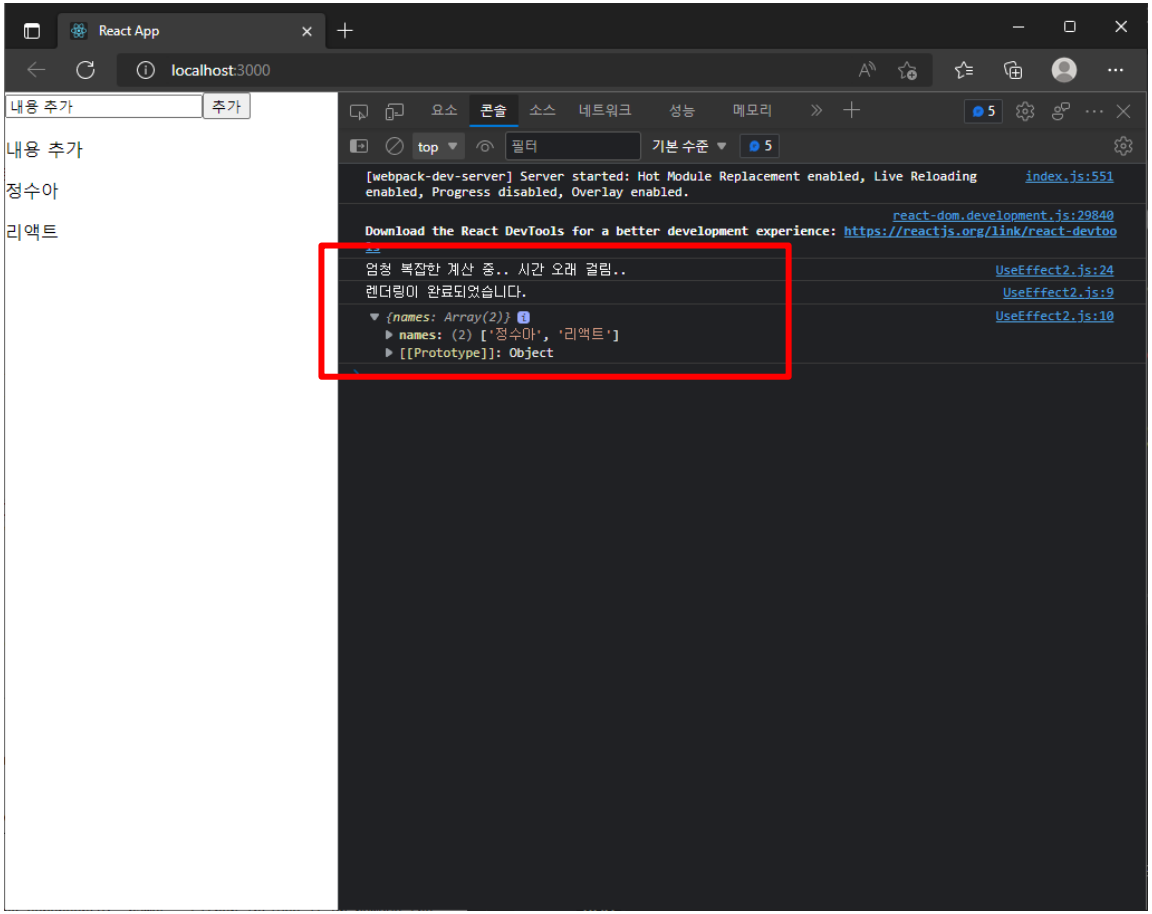
❖ **mount될 때만 실행하고 싶을 때**

- 함수의 두 번째 매개변수에 빈 배열을 넣음

```
useEffect(()=>{
                    console.log("렌더링이 완료되었습니다.");
                    console.log({names});
}, [])
```

[실습] useEffect()

❖ 실행 결과



[실습] useEffect()

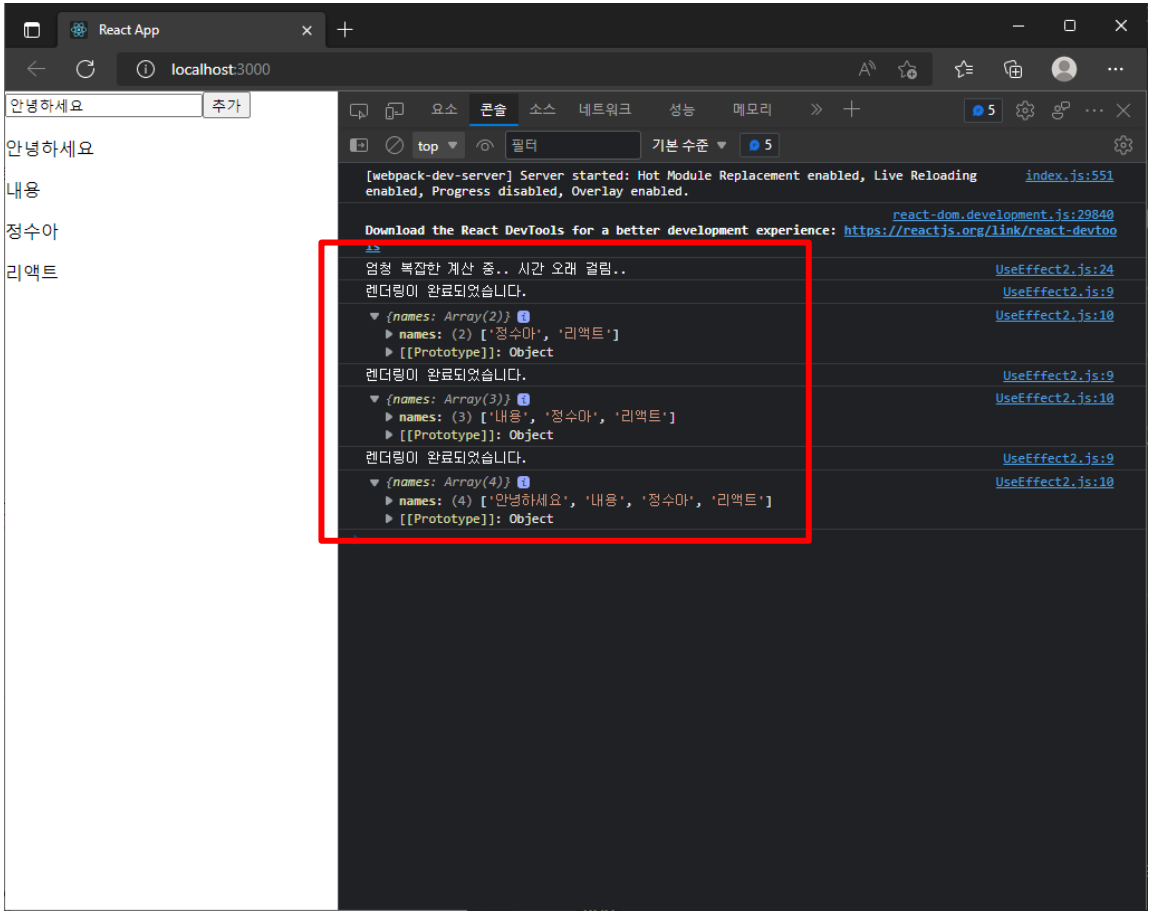
❖ **특정 값이 update될 때만 실행하고 싶을 때**

- 함수의 두 번째 매개변수 배열에 검사하고 싶은 값을 넣어 줌

```
useEffect(()=>{
                    console.log("렌더링이 완료되었습니다.");
                    console.log({names});
}, [names])
```

[실습] useEffect()

❖ 실행 결과



뒷정리하기

❖ 뒷정리하기 - cleanup

- 컴포넌트가 update되기 직전 또는 unmount되기 직전에 어떠한 작업을 수행하고 싶다면 뒷정리(cleanup) 작업을 해줘야 함

- 해결 방법
  - useEffect() 함수 내부에서 return 함수를 반환하면 됨

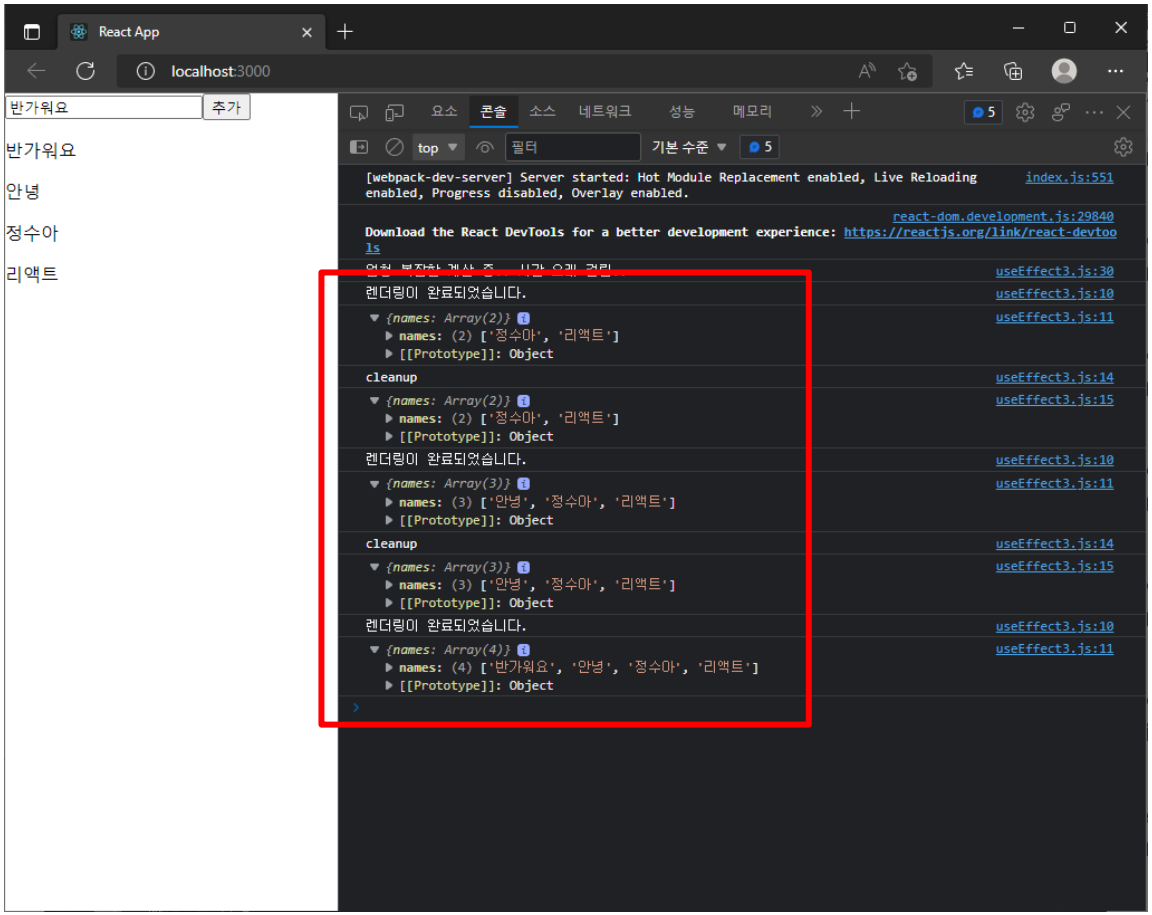
[실습] cleanup

❖ AddName.js 수정

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

function AddName() {
    // 생략

    useEffect(()=>{
                        console.log("렌더링이 완료되었습니다.");
                        console.log({names});

                        return () => {
                                        console.log("cleanup");
                                        console.log({names});
                                    }
    }, [names])

    return (
          // 생략
    );
};

export default AddName;
```

[실습] cleanup

❖ 실행 결과



04 useRef()

useRef()

❖ **useRef()**
- 컴포넌트 내부에서 사용되는 변수를 저장하는 Hook

❖ **특징**
- 컴포넌트가 재렌더링되어도 저장된 변수 값을 유지
- 불필요한 렌더링을 방지할 수 있음
- 특정 DOM 요소에 접근 가능

useRef()

❖ 사용 방법

```
const ref = useRef(value)
```

- useRef() 함수는 value 값으로 초기화된 ref 객체를 반환

- ref 객체

```
{ current : value }
```

- ref 객체 값 변경

```
ref.current = "hello"
```

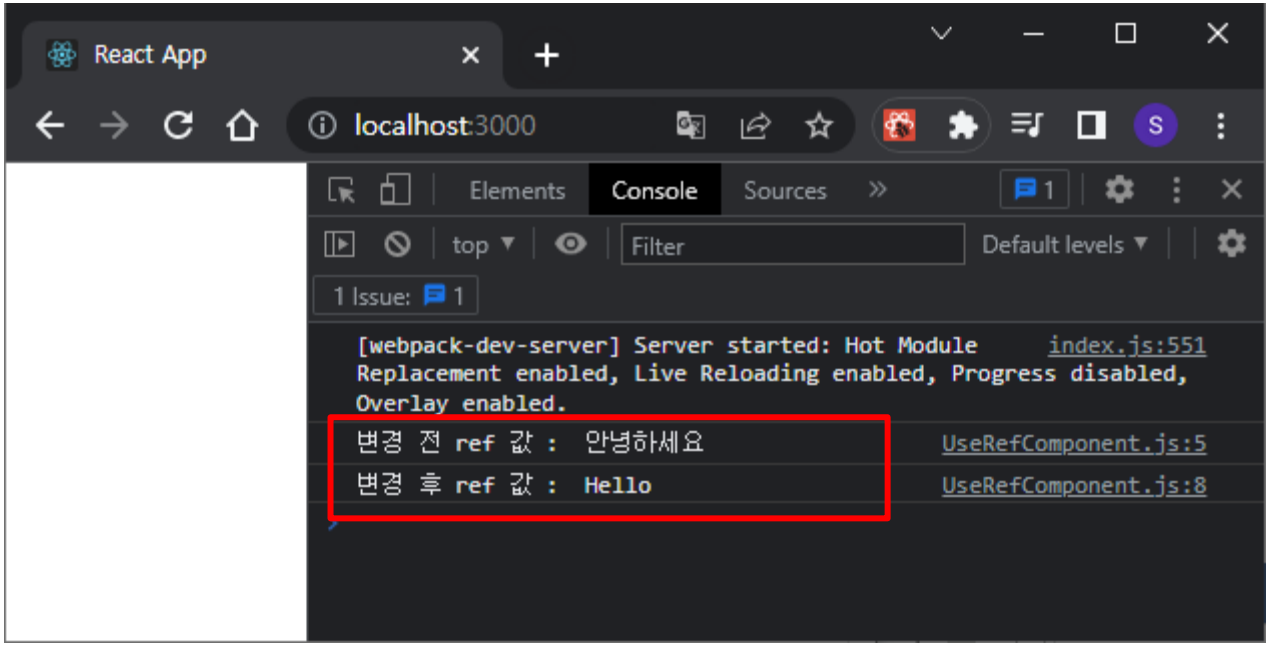
[실습] useRef() 사용하여 값 저장하기

❖ **UseRefComponent1.js**

```
import React, { useRef } from "react";

const UseRefComponent1 = () => {
  const ref = useRef("안녕하세요");
  console.log("변경 전 ref 값 : ", ref.current);

  ref.current = "Hello";
  console.log("변경 후 ref 값 : ", ref.current);

  return <div></div>;
};

export default UseRefComponent1;
```

[실습] useRef() 사용하여 값 저장하기

❖ 실행 결과



[실습] State vs useRef()

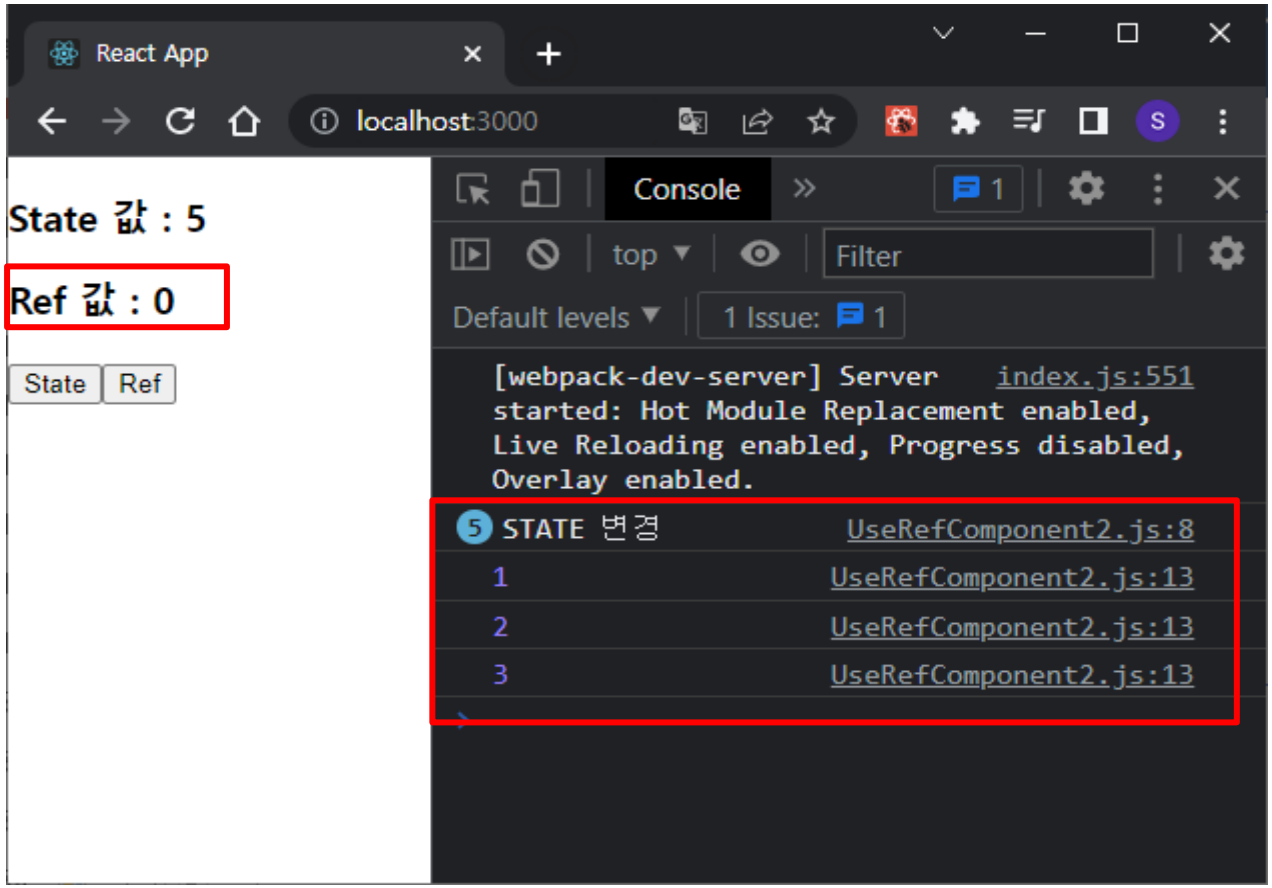
❖ **UseRefComponent2.js**

```
const UseRefComponent2 = () => {
  const [count, setCount] = useState(0);
  const countRef = useRef(0);

  function addStateHandler() {
    console.log("STATE 변경");
    setCount(count + 1);
  }

  function addRefHandler() {
    countRef.current = countRef.current + 1;
    console.log(countRef.current);
  }

  return (
    <div>
      <h3>State 값 : {count}</h3>
      <h3>Ref 값 : {countRef.current}</h3>
      <button onClick={addStateHandler}>State</button>
      <button onClick={addRefHandler}>Ref</button>
    </div>
  );
};
```

[실습] State vs useRef()

❖ 실행 결과

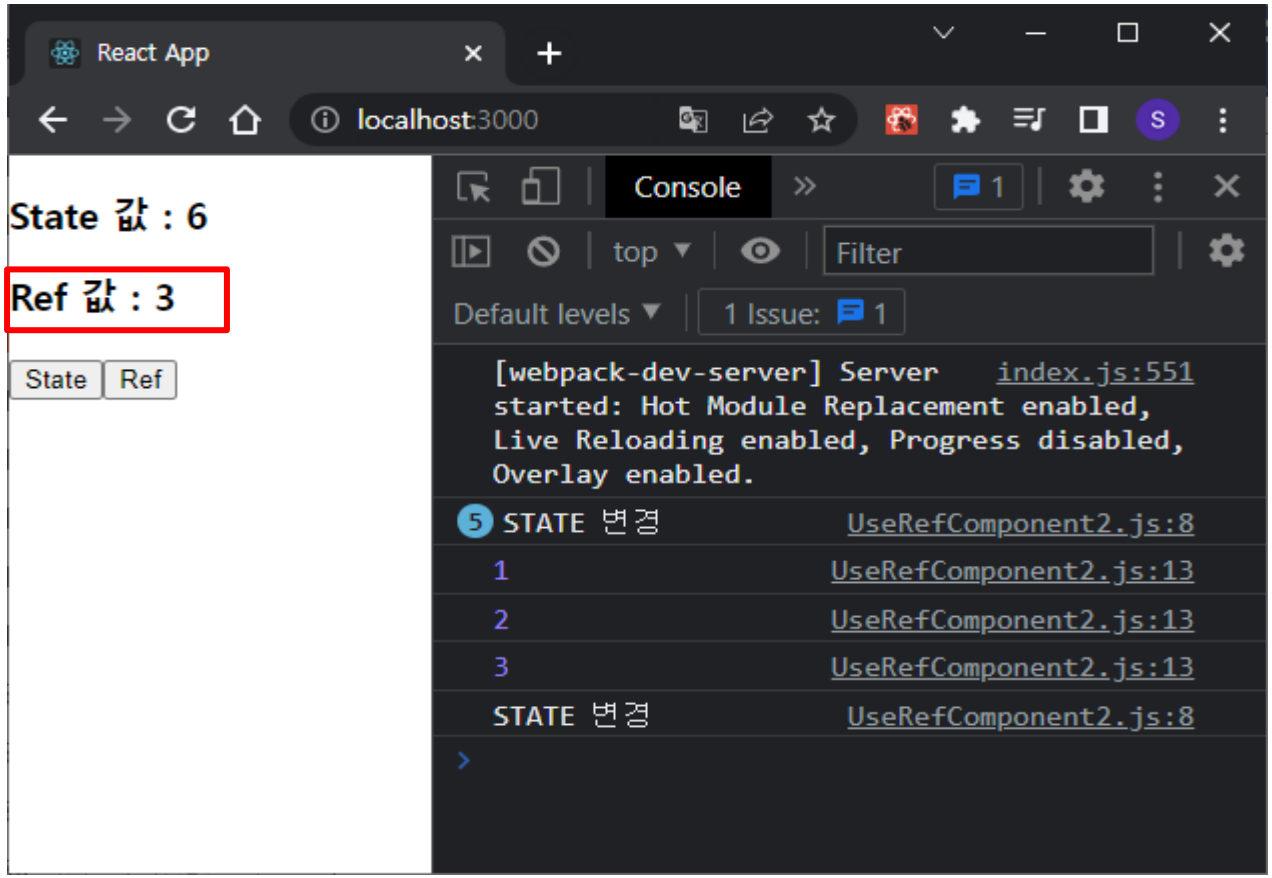
- State 버튼 5번 클릭, Ref 버튼 3번 클릭



[실습] State vs useRef()

❖ 실행 결과

- State 버튼 1번 클릭



[실습] useRef() vs 일반 변수

❖ **UseRefComponent3.js**

```
const UseRefComponent3 = () => {
  const [refresh, setRefresh] = useState();
  const countRef = useRef(0);
  let currentVar = 0;

  function refreshHandler() {
    setRefresh(refresh + 1);
  }

  function addRefHandler() {
    countRef.current = countRef.current + 1;
    console.log("ref : ", countRef.current);
  }

  function addVarHandler() {
    currentVar = currentVar + 1;
    console.log("var : ", currentVar);
  }
}
```

[실습] useRef() vs 일반 변수

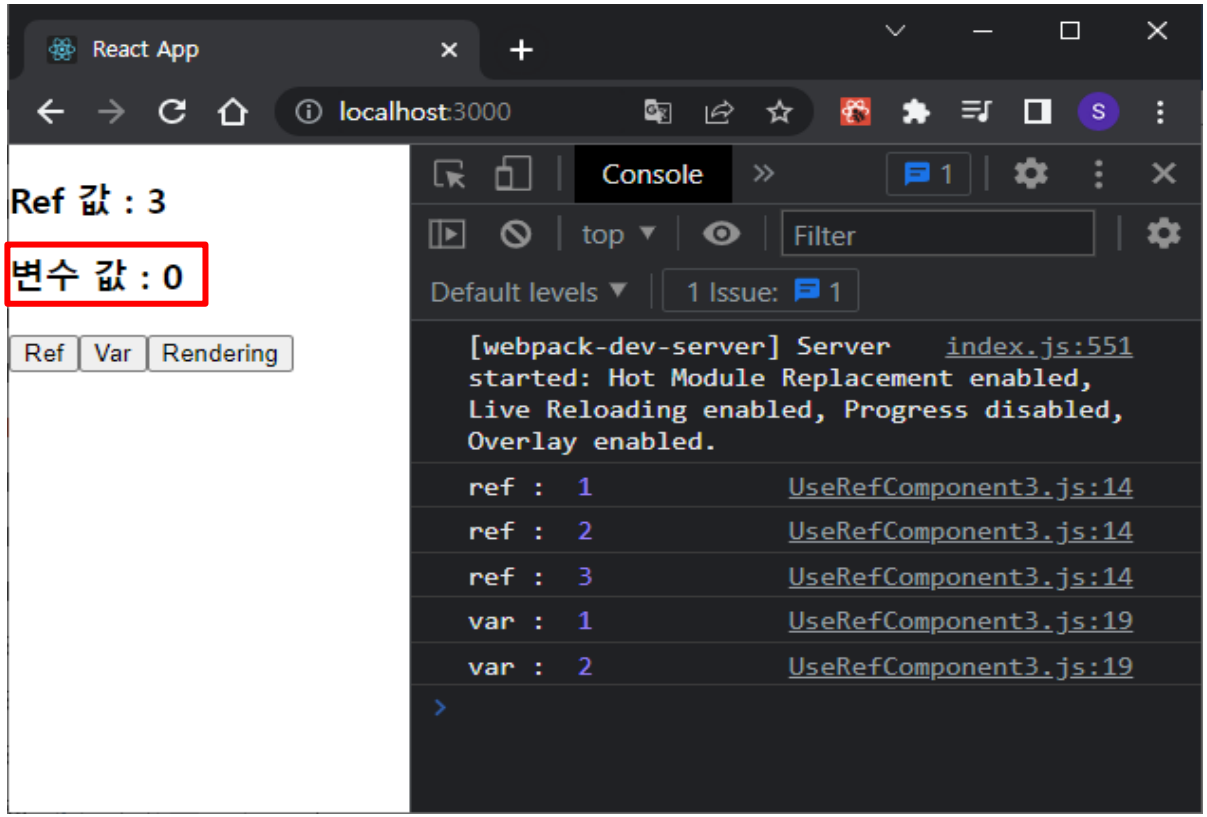
❖ **UseRefComponent3.js**

```
const UseRefComponent3 = () => {
  const [refresh, setRefresh] = useState();
  const countRef = useRef(0);
  let currentVar = 0;

  // 생략

  return (
    <div>
      <h3>Ref 값 : {countRef.current}</h3>
      <h3>변수 값 : {currentVar}</h3>
      <button onClick={addRefHandler}>Ref</button>
      <button onClick={addVarHandler}>Var</button>
      <button onClick={refreshHandler}>Rendering</button>
    </div>
  );
}
```

[실습] useRef() vs 일반 변수

❖ 실행 결과

- Ref 버튼 3번 클릭, Var 버튼 2번 클릭, Rendering 버튼 1번 클릭



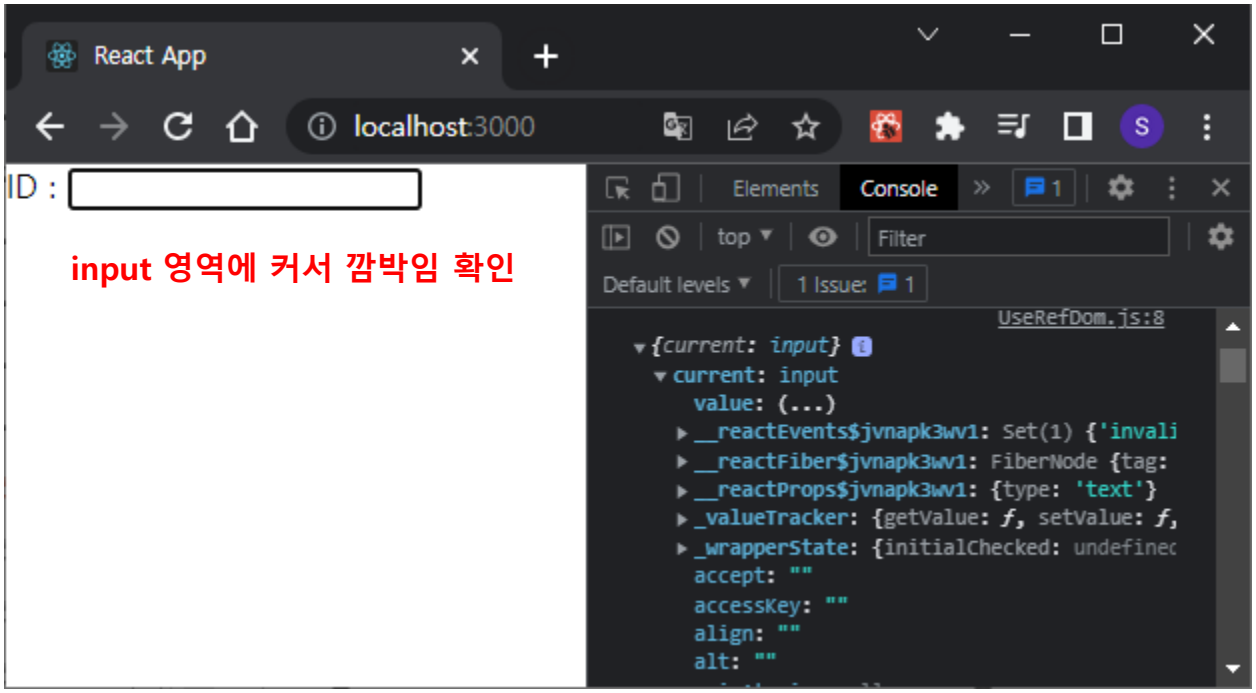
[실습] useRef()로 DOM에 접근하기

❖ **UseRefDom.js**

```
import { useEffect, useRef } from "react";

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

  useEffect(() => {
    console.log(inputRef);

    // input 태그에 focus 설정
    inputRef.current.focus();
  }, []);

  return (
    <div>
      ID : <input type="text" ref={inputRef} />
    </div>
  );
};

export default UseRefDom;
```

[실습] useRef()로 DOM에 접근하기

❖ 실행 결과




THANK YOU