5. ref: DOM에 이름달기

- 리액트 프로젝트 내부에서 DOM에 이름을 다는 방법
- 리액트에서는 id사용을 권하지 않음(SPW)

5.1 ref는 어떤 상황에서 사용해야 할까?

- DOM을 직접적으로 건드려야 할때

5.1.1

```
const ValidationSample =() =>{
    const [password, setPassword] = React.useState('');
    const [clicked, setClicked] = React.useState(false);
    const [validated, setValidate] = React.useState(false);
}
const handleChange = (e)=>{
    setPassword(e.target.value);
};
const handleButtonClick=()=>{
    setClicked(true);
    setValidated(password === 0000);
}
```

5.1.3 DOM을 꼭 사용해야하는 상황

state만으로 해결할수 없는 기능, Dom에 직접적으로 접근 => ref 사용

- 특정 input에 포커스 주기
- 스크롤 박스 조작하기
- canvas 요소에 그림그리기 등

5.2 ref 사용

React.useRef()

리액트에 내장된 훅을 사용

```
const RefSample=()=>{
    const input = React.useRef();

    const handleFocus = ()=>{
    input.current.focus();
}
}
```

5.2.2

- 컴포넌트 내부에서 변수 로 React.useRef()를 담아주어야 합니다 해당 변수를 ref를 달고자 하는 요소에 ref props로 넣어 주면 ref 설정이 완료됩니다.
- 설정한 뒤 나중에 ref를 설정해준 DOM에 접근하여면 input.current를 조회하면 됩니다
- 콜백함수를 사용할 때와 다른점은 .current를 넣어 주어야 함

5.2.3.2 버튼 onClick이벤트 코드 수정

```
const handleButtonClick=()=>{
    setClicked(true);
    setValidated(password === 0000);
    input.current.focus();
}
```

> ref 참조변수.current는 일반 자바스크립트 HTMLElement 객체와 동일 => document.querySelector("...")

> src/components/MyBox.js

```
import React from 'react';

//부모로부터 전달받은 ref 참조변수를 받기위해 React.forwardRef hook에 대한 콜백으로 컴포넌트를 구현한다.
//이렇게 구현된 컴포넌트는 props와 부모로부터 전달받은 ref 참조변수를 파라미터로 주입받는다

const MyBox = React.forwardRef((props, ref) => {
  const containerStyle = {
    border: '1px solid black',
    height: '100px',
    width: '100px',
  };

  //부모로부터 전달받은 ref 참조변수를 div에 연결한다
  //이 참조변수를 통해 부모 컴포넌트가 ref참조변수가 연결된 div의 DOM에 접근할 수 있다.
  return <div style={containerStyle} ref={ref}></div>;
});

export default MyBox;
```

> src/pages/MyRef.js

```
import React from 'react';
import MyBox from '../components/MyBox';

/**
 * React 에서 document.getElementById(...) 에 해당하는 기능을 사용하는 방법
 */

const MyRef = () => {
  //HTML 태그를 react안에서 참조할 수  있는 변수를 생성
  const myDname = React.useRef();
  const myLoc = React.useRef();
  const myResult = React.useRef();

  //컴포넌트에 설정하기 위한 ref
  const myBoxRef = React.useRef();

  return (
    <div>
      <h2>MyRef</h2>
      <h3>ref 기본 사용 방법</h3>
      {/**미리 준비한 컴포넌트 참조변수와 HTML태그를 연결 */}
      <div>
        <label htmlFor="dname">학과명</label>
        <input type="text" ref={myDname} id="dname" />
      </div>
      <div>
        <label htmlFor="dname">학과위치</label>
        <input type="text" ref={myLoc} id="loc" />
      </div>
      <h3>
        입력값 확인: <span ref={myResult}></span>
      </h3>
      <button
        onClick={(e) => {
          /**컴포넌트 참조변수를 사용해서 다른 HTML태그에 접근 가능
           * -->'참조변수.current' 해당 HTML을 의미하는 javascript DOM객체
           * --> myDname.current와 document.querySelector(...),
document.getElementById(...) 등으로 생성한 객체가 동일한 DOM객체이다
           */
          console.log(myDname);
          console.log(myLoc);

          const dname = myDname.current.value;
          const loc = myLoc.current.value;

          myResult.current.innerHTML = dname + ', ' + loc;
        }}
      >
        클릭
      </button>
      <hr />
      <h3>컴포넌트에 ref 적용하기</h3>
      {/**ref 참조변수를 컴포넌트에 전달한다 */}
```

```
        <MyBox ref={myBoxRef} />
        <button
          type="button"
          onClick={() => {
            //<MyBox>를 통해 myBoxRef를 주입받은 DOM에 접근하여 제어함.
            myBoxRef.current.style.backgroundColor = '#f00';
          }}
        >
          red
        </button>

        <button
          type="button"
          onClick={() => {
            //<MyBox>를 통해 myBoxRef를 주입받은 DOM에 접근하여 제어함.
            myBoxRef.current.style.backgroundColor = '#00f';
          }}
        >
          blue
        </button>
      </div>
    );
};

export default MyRef;
```

07-hook-event

MyState | DateRange1 | MyEffect | MyRef | MyReducer | DateRange2

MyRef

ref 기본 사용 방법

학과명 asdf
학과위치 1234

입력값 확인: asdf, 1234

클릭

컴포넌트에 ref 적용하기

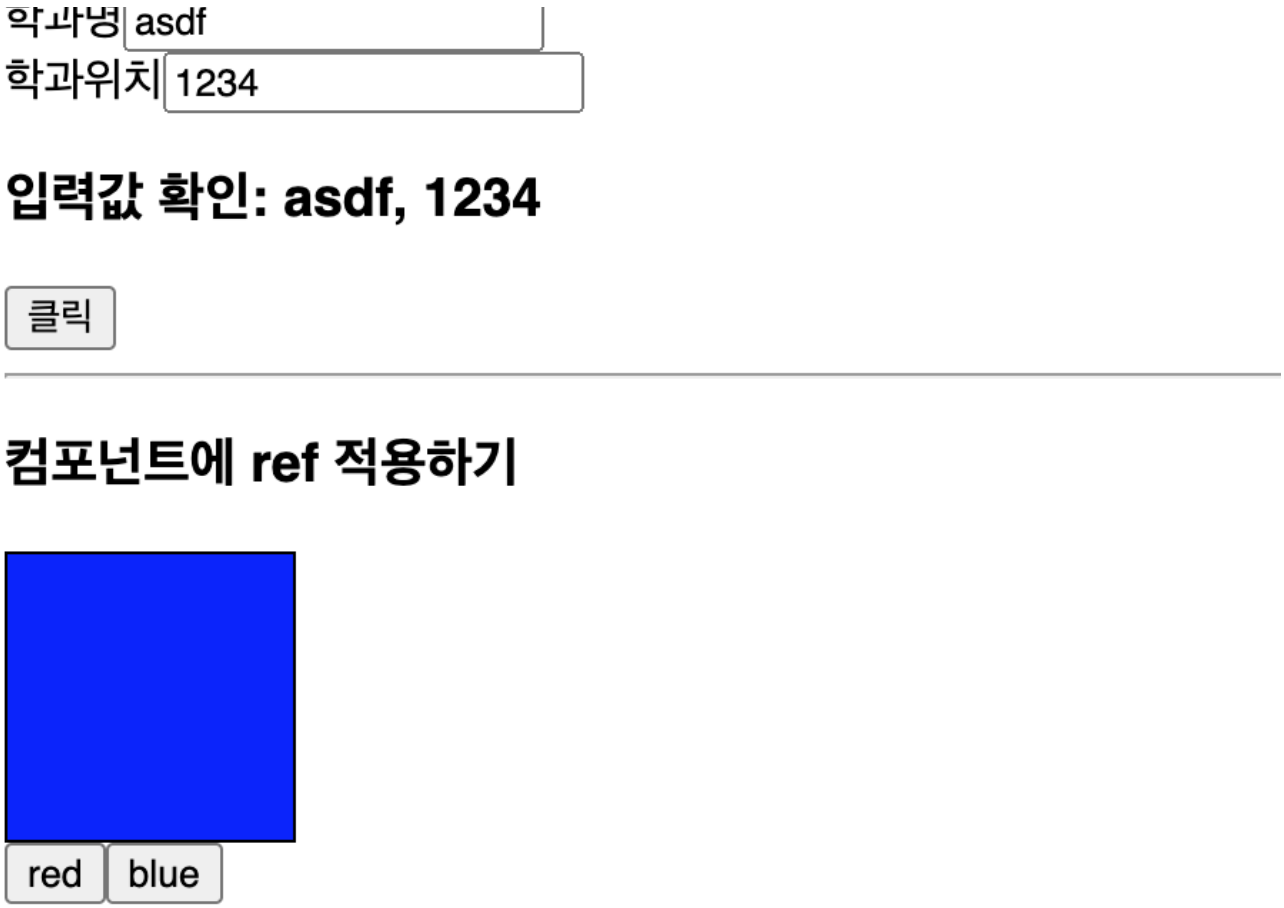
red blue

07-hook-event

MyState | DateRange1 | MyEffect | MyRef | MyRe

MyRef

ref 기본 사용 방법



5.3 컴포넌트에 ref 달기

컴포넌트에도 ref를 달수 있다 컴포넌트 내부에 있는 DOM을 컴포넌트 외부에서 사용할때

5.3.1 사용법

참조변수 이름 명시 메서드와 멤버변수가 클래스인 경우에만 성립되므로 함수형 컴포넌트에서는 성립안됨

```
const ScrollBox = React.forwardRef((props, ref)=>[
    ...
    ...
    return (
        <div
        ref={ref}>
        </div>
    )
])
```

5.3.4 컴포넌트에 ref달고 내부 매서드 사용

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

```
const App = () =>{
    const srollBoxRef = React.useRef();

    return(
        <div>
            <ScrollBox ref={scrollBoxRef}>/>
            <button onClick={()=>{
                const {scroolHeight, clientHeight} =
scrollBoxRef.current;//부모 div
                scrollBoxRef.current.scrollTop = scrollHeight -
clientHeight;//스크롤바 맨끝이동
            }}>맨밑으로</button>
        </div>
    );
};

export default App;
```

5.4 정리

- 컴포넌트 내부에서 DOM에 직접 접근해야 할때는 ref를 사용합니다 먼저 ref를 사용하지 않고도 원하는 기능을 구현할 수 잇는지 반드시 고려한 후에 활용
- 서로 다른 컴포넌트끼리 데이터를 교류할때 ref를 사용한다면 이는 잘못된것
- 컴포넌트끼리 데이터 교류할때 언제나 데이터를 부모<->자식 흐름으로 교류
- 나중에 리덕스 효율적으로 교류하는 방법

---

6. 컴포넌트 반복

자바스크립트 배열 객체의 내장함수인 map 함수를 사용하여 반복되는 컴포넌트 렌더링

6.1 자바스크립트 배열의 map()함수

6.2 데이터 배열을 컴포넌트 배열로 변환

```
const IterationSample = () =>{
    const names = [...];
    const nameList = names.map(name=> <li>{name}</li>);
    return <ul>{nameList}</ul>;
}
```

- ...

jsx코드로 된 배열을 새로 생성한수 nameList에 담음 map 함수에서 JSX를 작성할 때는 앞서 다른 예제처럼 DOM요소를 작성해도 되고, 컴포넌트를 사용해도 됨

6.3 key

key는 컴포넌트 배열을 렌더링 했을 때 어떤 원소에 변동이 있었는지 알아내려고 사용

6.3.1 key설정

- key값을 설정할 때는 map함수의 인자로 전달되는 함수 내부에서 컴포넌트 props를 설정하듯이 설정
- key값은 언제나 유일
- 데이터가 가진 고윳값을 key로 설정
- 게시판의 게시물을 렌더링한다면 게시물 번호를 key값으로 설정

6.4 응용

```
 const onClick = () =>{ //names 배열에 원소가 추가된 복사본을 nextNames로 반환함
(push는 원보에 직접)
     const nexNames = names.concat({
         id: nextId // nextId값을 id로 설정하고
         text: inputText
     })
     setNextId(nextId + 1); // nextId값에 1을 더해 준다
     setNames(nextNames); // names값을 업데이트 한다
     setinputText(''); //inputText를 비운다
 }
```

배열에 새 항목을 추가할때 배열의 push 함수를 사용하지 않고 concat을 사용 push는 기존 배열자체를 변경 concat은 새로운 배열을 생성 리액트에서 상태를 업데이트할때는 기존상태를 그대로 두면서 새로운 값을 상태로 설정해야함 => 불변성유지

6.5 정리

컴포넌트 배열을 렌더링 할때는 key값 설정에 항상주의 key값은 언제나 유일 key값이 중복된다면 렌더링 과정에서 오류 상태 안에서 배열을 변형할 때는 배열에 직접 접근하여 수정하는 것이 아니라 concat, filter등의 배열의 배열 내장함수를 사용하여 새로운 배열을 만든후 새로운 상태로 설정해주어야

---

8. Hooks

8.1 useState

함수형 컴포넌트에서도 가변적인 상태를 지닐수 잇께 해줌

```
const [상태변수, 상태변수의 값을 갱신하기 위한 함수] = useState(상태변수의 초기값);
return(
<>{상태값갱신을 위한 함수를 통해 갱신된 값이 자동으로 실시간 반영}</>
)
```

- useState 함수의 파라미터에는 상태의 기본값을 넣어줍니다
- 배열을 반환함
- 첫번째요소는 상태값 두번째요소는 상태를 설정하는 함수

- 이 함수에 파라미터를 넣어서 호출하면 전달받은 파라미터로 값이 바뀌고 컴포넌트가 정상적으로 리렌더링

8.1.1 useState여러번 사용하기

하나의 useState함수는 하나의 상태값만 관리 컴포넌트에서 관리해야할 상태가 여러개라면 useState를 여러번 사용

8.2 useEffect

useEffect는 리액트 컴포넌트가 랜더링(화면에 그림) 될때마다 특정작업을 수행하도록 설정

8.2.1 마운트 될때만 실행하고 싶을때

useEffect에서 설정한 함수를 컴포넌트가 화면에 맨 처음 랜더링 될때만 실행학고 업데이트될 때는 실행하지 않으려면 함수의 두번째 파라미터로 비어 있는 배열을 넣어주면됨

8.2.2 특정값이 업데이트될때만 실행하고 싶은때

useEffect의 두번째 파라미터로 전달되는 배열안에 검사하고 싶은 값을 넣어주면 됨

> 배열안에서 useState를 통해 관리하고 있는 상태를 넣어주어도 되고 props로 전달받은 값을 넣어주어도 됩니다

8.2.3 뒷정리하기

- 컴포넌트가 언마운트 되기 전이나 업데이트 되기 직전에 어떠한 작업을 수행하고 싶다면 useEffect에서 뒷정리 함수(함수를 리턴)를 반환해 주어야함
- 오직 언마운트 될때만 뒷정리 함수를 호출하고 싶다면 useEffect함수의 두번째 파라미터에 비어있는 배열을 넣으면 됨

8.3 useReducer

- useReducer는 userState보다 더 다양한 컴포넌트 상황에 따라 다양한 상태를 다른값으로 업데이트해주고 싶을때 사용
- 리듀서는 현재상태 그리고 업데이트를 위해 핑요한 정보를 담은 action값을 전달받아 새로운 상태를 반환하는 함수
    - 리듀서함수에서 새로운 상태를 만들때는 받드시 불변성을 지켜야함
    - 상태값에 대한 복사본을 생성한후 복사본을 활용하여 상태값을 갱신해야한다
- useReducer의 첫번째 파라미터에는 리듀서 함수를 넣고 두번째 파라미터에는 해당 리듀서의 기분값을 넣음
- state는 현재 가리키고 있는 상태
- dispatch는 action을 발생시키는 함수
- dispatch(action)과 같은 형태로 함수안에 파라미터로 액션값을 넣어주면 리듀서 함수가 호출되는 구조
- useReducer를 사용했을때의 컴포넌트 업데이트 로직을 컴포넌트 바깥으로 빼낼수있는 장점

8.3.2 input상태 관리하기

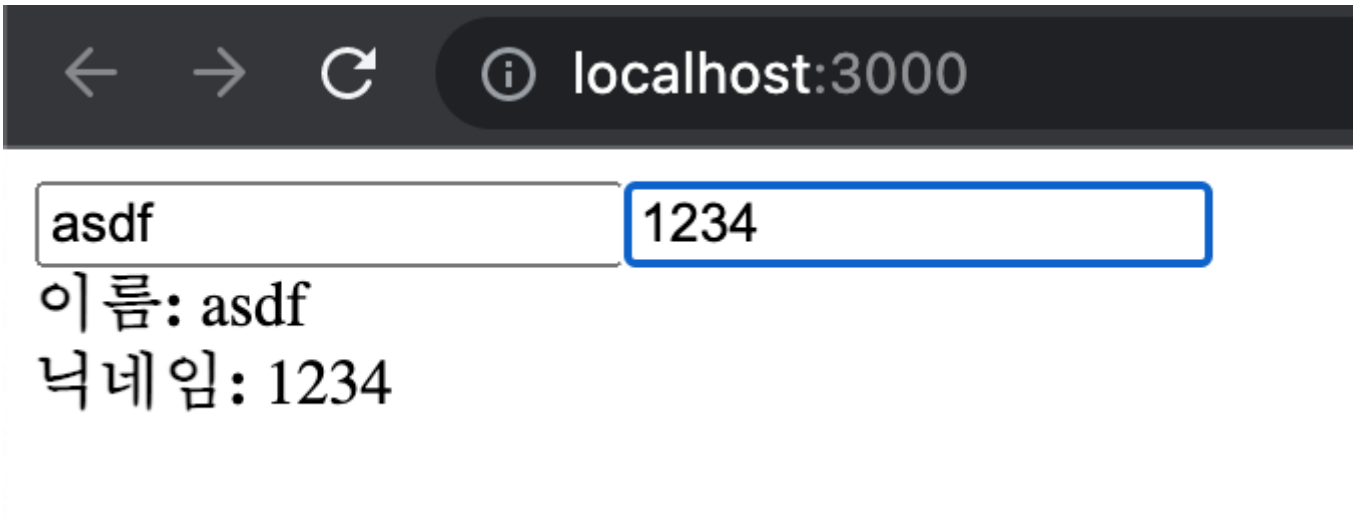
> json key 동적 명시

```
const key ="b";
const data = {
    [key]: 100
};
->{b: 100 }
```

useReducer의 action은 어떤 값도 사용가능

src/Info.js

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

function reducer(state, action) {
  return {
    ...state,
    [action.name]: action.value,
  };
}

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;
```



8.4 useMemo

- 함수형 컴포넌트 내부에서 발생하는 연산의 최적화
- 숫자를 등록할 때 분만아니라 인풋내용이 수정될 때도 getAverage함수가 호출되는 렌더링 계산낭비를 막기위해 useMemo를 사용
- 렌더링 과정에서 특정값이 바뀌었을때만 연산을 실행하고 원하는 값이 바뀌지 않았다면 이전결과 다시사용

8.5 useCallback

- 렌더링 성능을 최적화해야 하는 상황에서 사용
- 이벤트 핸들러 함수를 필요할 때만 생성할수 잇다
- 컴포넌트 랜더링이 자주 발생하거나 랜더링해야할 컴포넌트의 개수가 많아지면 이부분을 최적화 해주는 것이 좋습니다
- useCallback 의 첫번째 파라미터에는 생성하고 싶은 함수를 넣고
- 두번째파라미터에는 배열을 넣는다
- 이 배열의 어떤값이 바뀌었을때 함수를 새오 생성해야하는지 명시
- 비여있는 배열을 넣으면 컴포넌트가 랜더링될때 단 한번만 함수가 생성
- 비여있지 않은 배열을 넣으면 내용이 바뀌거나 새로운 항목이 추가 될때마다 함수가 생성
- 함수내부에서 상태값에 의존해야할때 그값을 반드시 두번째 파라미터안에 포함시켜야한다

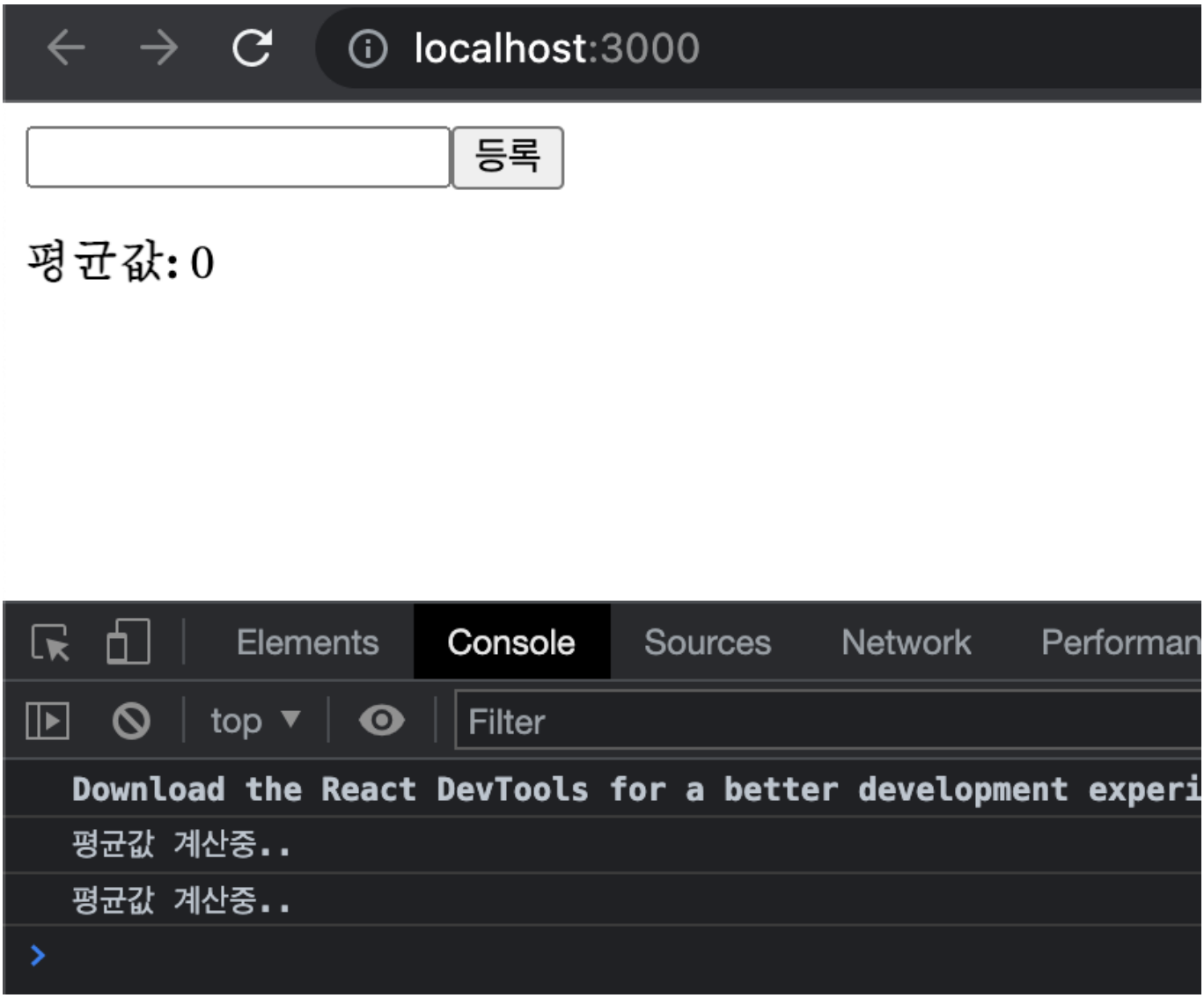
> 숫자 문자열 객체처럼 일반값을 재사용하려면 useMemo 함수는 재사용하려면 useCallback

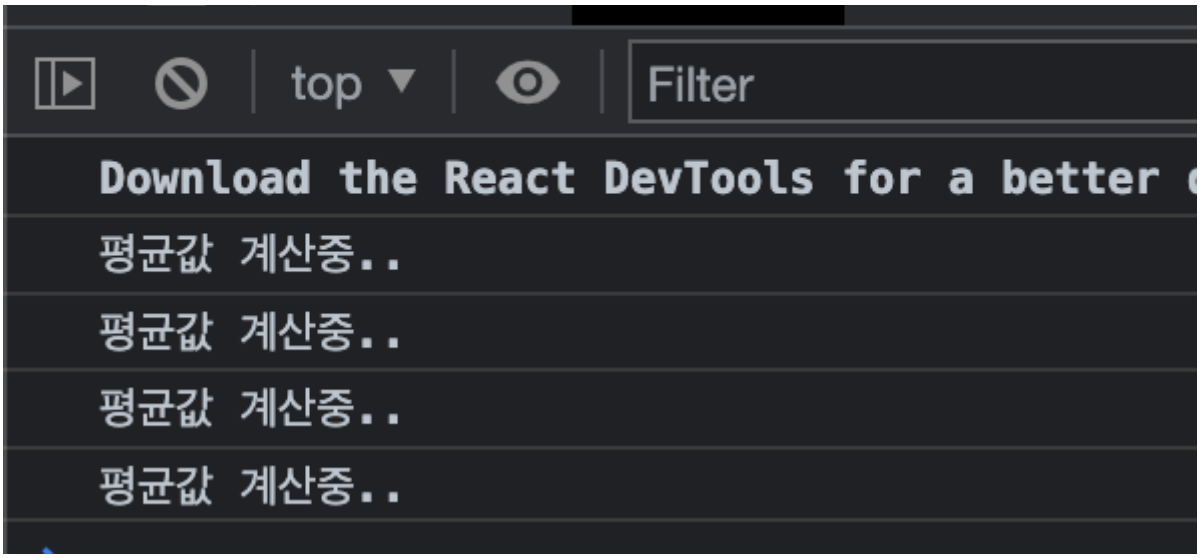
8.6 useRef

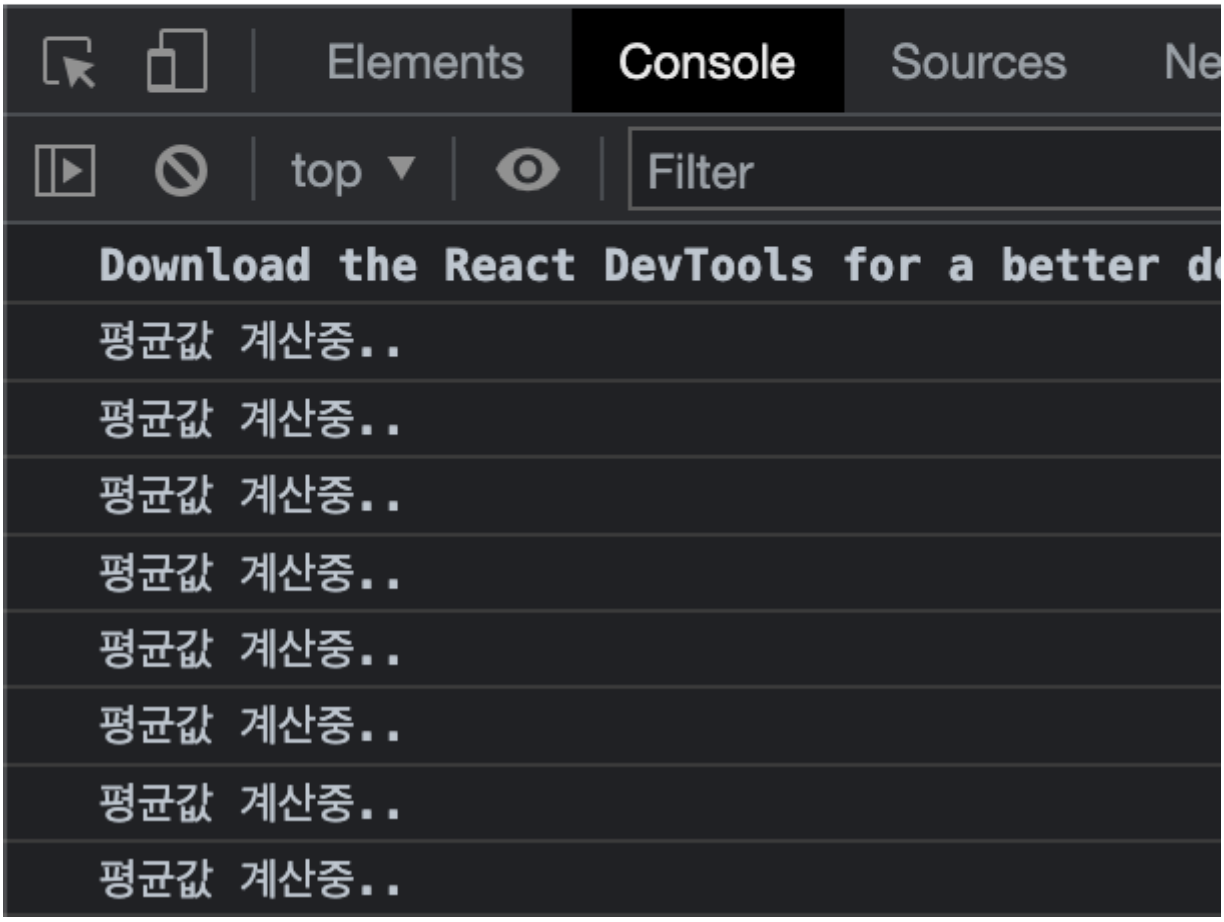
> src/Average.js

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

const getAverage = numbers => {
  console.log('평균값 계산중..');
  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 inputEl = useRef(null);

  const onChange = useCallback(e => {
    setNumber(e.target.value);
  }, []); // 컴포넌트가 처음 렌더링 될 때만 함수 생성
  const onInsert = useCallback(() => {
    const nextList = list.concat(parseInt(number));
    setList(nextList);
    setNumber('');
    inputEl.current.focus();
  }, [number, list]); // number 혹은 list 가 바뀌었을 때만 함수 생성

  const avg = useMemo(() => getAverage(list), [list]);

  return (
    <div>
      <input value={number} onChange={onChange} ref={inputEl} />
      <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;
```


localhost:3000
등록
평균값: 0
Elements
Console
Sources
Network
Performan
top
Filter
Download the React DevTools for a better development experi
평균값 계산중..
평균값 계산중..
localhost:3000
등록
1
평균값: 1
Elements
Console
Sources


top
Filter
Download the React DevTools for a better
평균값 계산중..
평균값 계산중..
평균값 계산중..
평균값 계산중..



8.7 customizing

여러컴포넌트에서 비슷한 기능을 공유할 경우 이를 여러분만의 Hook으로 작성 로직을 재사용할수 있다

> src/useInputs.js

```
import { useReducer } from 'react';

function reducer(state, action) {
```

```
  return {
    ...state,
    [action.name]: action.value
  };
}

export default function useInputs(initialForm) {
  const [state, dispatch] = useReducer(reducer, initialForm);
  const onChange = e => {
    dispatch(e.target);
  };
  return [state, onChange];
}
```

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

const Info = () => {
  const [state, onChange] = useInputs({
    name: '',
    nickname: ''
  });
  const { name, nickname } = state;

  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;
```

8.8 another

8.9 summary