8. Hooks

8.1 useState

- https://velog.io/@jjunyjjuny/React-useState는-어떻게-동작할까
- useState는 가장 기본적인 Hook, 함수 컴퍼넌트에서도 가변적인 상태를 가질 수 있게 한다.

8.1.1 가장 기본적인 Hook

Counter.js

> - 선언방법 : import { useState } from "react";
> - const [상태값, 상태변경함수] = useState(초기값);

```
import { useState } from "react";

const React0801Counter = () => {

  // value : 상태값, setValue : 상태변경함수 useState(0) 초기값
  const [value, setValue] = useState(0);

  return (
    <div>
      <p>현재 카운터의 값은 <b>{value}입니다.</b></p>
      <button onClick={() => setValue(value+1)}>+1</button>
      <button onClick={() => setValue(value+1)}>-1</button>
    </div>
  )
}

export default React0801Counter;
```

App.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

8.1.2 useState 여러번 사용하기

React0802Info.js

```
import { useState } from "react";

const React0802Info = () => {

  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 type="text" value={name} onChange={onChangeName} />
        <input type="text" value={nickname} onChange={onChangeNickName} />
      </div>
      <div>
        <b>name : </b> {name}
      </div>
      <div>
        <b>nikname : </b> {nickname}
      </div>
    </div>
  );
}


export default React0802Info;
```

App.js

```
import React, { Component } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';

class App extends Component {

  render() {
    return (
      <div>
          <React0801Counter />
          <hr/>
          <React0802Info />
      </div>
    );
  }
}
export default App;
```

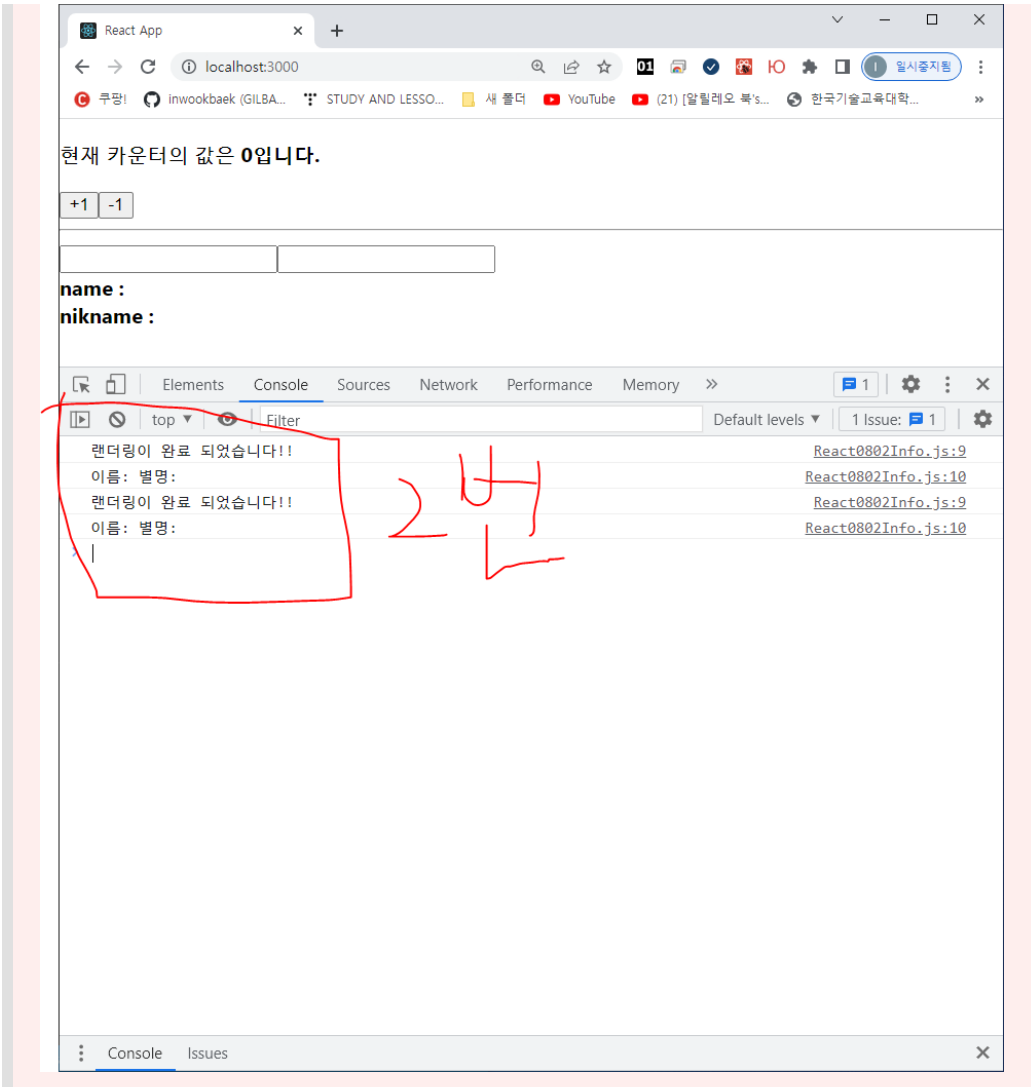
8.2 useEffect

- 참고 : https://velog.io/@babypig/React-useEffect

- userEffect는 리액트 컴퍼넌트가 랜더링 될 때 마다 특정작업을 수행하도록 한다.
- 클래스 컴퍼넌트의 componentDidMount와 componenDidUpdate를 합친 형태 이다.
- 컴퍼넌트가 처음 나타났을 때 `두 번 출력`이 된다. 이는 `index.js에서 <React.StrictMode>가 적용된 개발환경`에서만 발행
- 이는 useEffect에 문제가 있는 여부를 감지하기 이해 두 번실행된다. 향후 버전에서 컴퍼넌트가 리랜더링되되 상태유지 기능도입예정

React0802Info.js 수정

```
import { useState } from "react";

const React0802Info = () => {

  const [name, setName] = useState('')
  const [nickname, setNickName] = useState('');

  //=======================================================
  useEffect(() => {
    console.log('랜더링이 완료 되었습니다!!');
    console.log(`이름:${name} 별명: ${nickname}`);
  });
  //=======================================================

  const onChangeName = e => {
    setName(e.target.value);
  }
  const onChangeNickName = e => {
    setNickName(e.target.value);
  }

  return (
    <div>
      <div>
        <input type="text" value={name} onChange={onChangeName} />
        <input type="text" value={nickname} onChange={onChangeNickName} />
      </div>
      <div>
        <b>name : </b> {name}
      </div>
      <div>
        <b>nikname : </b> {nickname}
      </div>
    </div>
  );
}


export default React0802Info;
```



8.2.1 마운트될 때만 실행하기

- useEffect에서 설정한 함수를 컴퍼넌트가 화면에 맨 처음 랜더링될 때만 실행하고 업데이트될 때는 실해되지 않도록 방지하는 방법
- 이는 useEffect함수의 두 번째 파라미터에 빈 배열을 넣어 주면 된다.

```
useEffect(() => {
    console.log('랜더링이 완료 되었습니다!!');
    console.log(`이름:${name} 별명: ${nickname}`);
  }, []);
```

8.2.2 특정값이 업데이트될 때만 실행하기

- 클래스 컴퍼넌트형일 때 특정값이 변경될 때만 호출할 경우
- 아래와 같이 작성한다. 이 코드는 props안에 value값이 변경될 때만 특정 작업을 수행한다.

```
componentDidUpdate(prevProps, preState) {
    if(prevProps.value !== this.props.value) {
        doSomething.....
    }
}
```

- 상기 코드를 userEffect에서 작업하려면 `빈 베열에 변경되는 값을 넣어 주면 된다.`
- 배열 안에는 useState에서 관리하고 있는 상태를 넣어줘도 되고 props로 전달 받은 값을 넣어 줘도 된다.

```
useEffect(() => {
    console.log('랜더링이 완료 되었습니다!!');
    console.log(`이름:${name} 별명: ${nickname}`);
  }, [name]);
```

8.2.3 정리하기

- useEffect는 기본적으로 랜더링되고 난 직후마다 실행되며, 두 번째 파라미터 배열에 어떤 값을 넣는지에 따라 실행조건이 달라진다.
- 컴퍼넌트가 언마운트되기 직전이나 업데이트되기 직전에 어떠한 작업을 수행하고 싶다면 `useEffect에 cleanup함수를 반환` 해 주어야 한다.

React0803Info.js 작성

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

const React0803Info = () => {

  const [name, setName] = useState('')
  const [nickname, setNickName] = useState('');

  useEffect(() => {
    console.log('랜더링이 완료 되었습니다!!');
    console.log(`이름:${name} 별명: ${nickname}`);
    return () => {
      console.log('CleanUp!!');
      console.log(`이름:${name} 별명: ${nickname}`);
    };
  }, [name]);


  const onChangeName = e => {
    setName(e.target.value);
  }
  const onChangeNickName = e => {
    setNickName(e.target.value);
  }

  return (
    <div>
      <div>
        <input type="text" value={name} onChange={onChangeName} />
        <input type="text" value={nickname} onChange={onChangeNickName} />
      </div>
      <div>
        <b>name : </b> {name}
      </div>
      <div>
        <b>nikname : </b> {nickname}
      </div>
    </div>
```

```
  );
}


export default React0803Info;
```

App.js

```
import { useState } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';
import React0803Info from './mysrc/React0803Info';

const App = () => {

  const [visible, setVisible] = useState(false);
  return (
    <div>
        <React0801Counter />
        <hr/>
        <React0802Info />
        <hr/>
        <button onClick={() => {
          setVisible(!visible)
        }}>
          {visible ? '숨기기' : '보이기'}
        </button>
        { visible && <React0803Info /> }
    </div>
  );
}
export default App;
```

- 언마운팅될 때만 cleanup함수를 호출할 경우에는 useEffect에 빈배을을 넣으면 된다.

8.3 useReducer

- 참고 : https://velog.io/@babypig/useReducer
- `useReducer는 useState보다 더 다양한 컴퍼넌트 상황에 따라 다양한 상태를 다른 값으로 업데이트할 때 사용` 한다.
- 리듀서는 **현재 상태 그리고 업데이트를 위해 필요한 정보를 담은 액션(Action)값을 전달 받아 새로운 상태를 반환하는 함수**
- 리듀서 함수에서 `새로운 상태를 만들 때는 반드시 불변성` 을 지켜 주어야 한다.

```
function reducer(state, action) {
    return {  ...  };  // 불변성을 지키면서 업데이트한 새로운 상태를 반환
}

// 액션값의 형태
{
    type: 'INCREMENT',
    다른 값이 필요할 경우 추가
}
```

- 리덕스에서 사용하는 액션 객체에는 반드시 어떤 액션인지 알려 주는 type필드가 존재 해야 한다.
- 하지만, useReducer에서는 반드시 type을 지니고 있을 팔요가 없다. 심지어 객체가 아니라 문자열도 가능하다.

8.3.1 카운터 구현하기

- useReducer(리듀서함수, 초기값)을 설정 한다.
- 이 Hooks를 사용하면 state값과 dispatch함수를 받아온다 .
- state는 현재상태, dispatch는 액션을 발생시키는 함수 ,
- dispatch(action)과 같은 형태로 함수안에 파라미터를 넣어 주면 리듀서하수가 호출된다.
- useReducer의 가장 큰 장점은 컴퍼넌트 업데이트 로직을 외부로 뺴낼 수 있다는 점 이다.

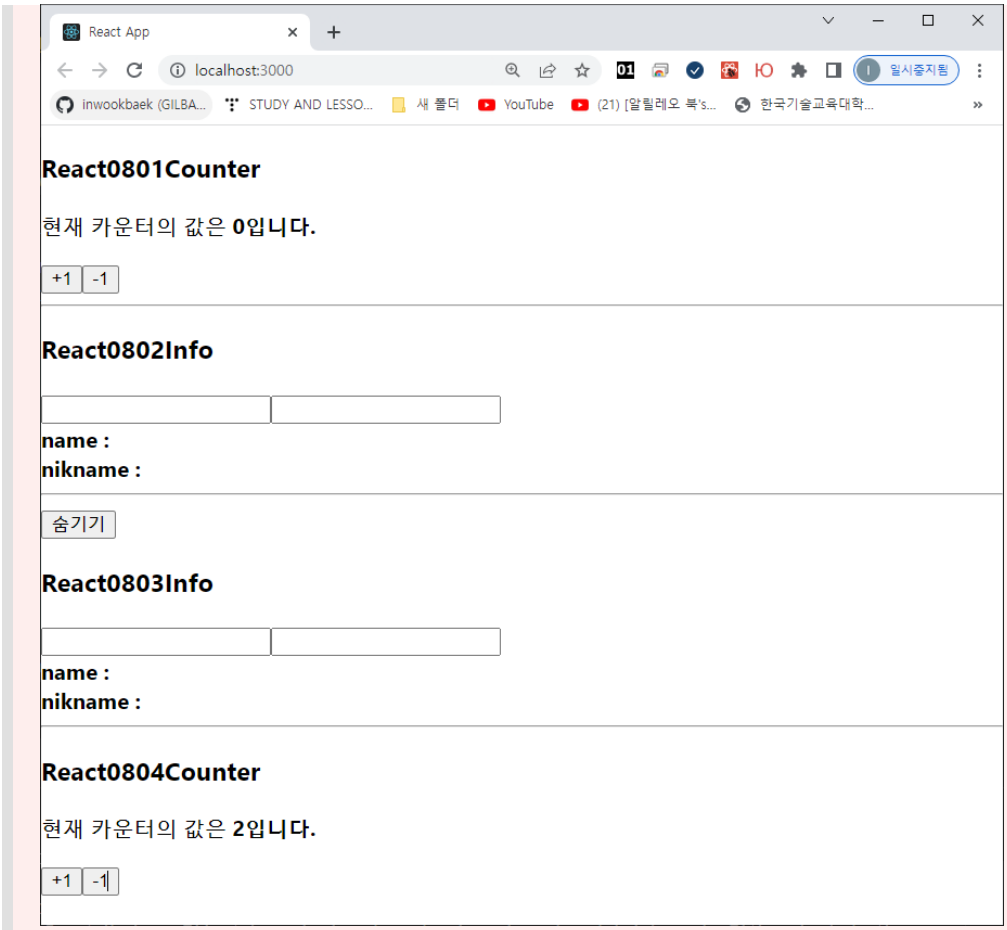
React0804Counter.js

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

function reducer(state, action) {
  switch(action.type) {
    case 'INCREMENT': return { value: state.value + 1 }
    case 'DECREMENT': return { value: state.value - 1 }
    default: return state; // 아무것도 해당되지 않을 경우 기전값
  }
}

const React0804Counter = () => {

  const [state, dispatch] = useReducer(reducer, {value: 0});

  return (
    <div>
      <p>현재 카운터의 값은 <b>{state.value}입니다.</b></p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>+1</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>-1</button>
    </div>
  )
}

export default React0804Counter;
```

App.js

```
import { useState } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';
import React0803Info from './mysrc/React0803Info';
import React0804Counter from './mysrc/React0804Counter';

const App = () => {

  const [visible, setVisible] = useState(false);
  return (
    <div>
        <React0801Counter />
```

```
        <hr/>
        <React0802Info />
        <hr/>
        <button onClick={() => {
          setVisible(!visible)
        }}>
          {visible ? '숨기기' : '보이기'}
        </button>
        { visible && <React0803Info /> }
        <hr/>
        <React0804Counter />
      </div>
    );
}
export default App;
```



8.3.2 input state관리하기

- useReducer를 사용하면 기존에 클래스형 컴퍼넌트에서 input태그에 값을 할당하고
- e.target.name을 참조하여 setState방식과 유사처리 가능하다
- useReducer에서의 `액션은 그 어떤 값도 사용이 가능` 하다.
- 이벤트 객체가 지고 있는 `e.target값 자체를 액션값으로 사용`
- 이 방법으로 input을 관리하면 input의 갯수와는 상관없이 코드를 간단하게 관리할 수 있다.

React0805Info.js

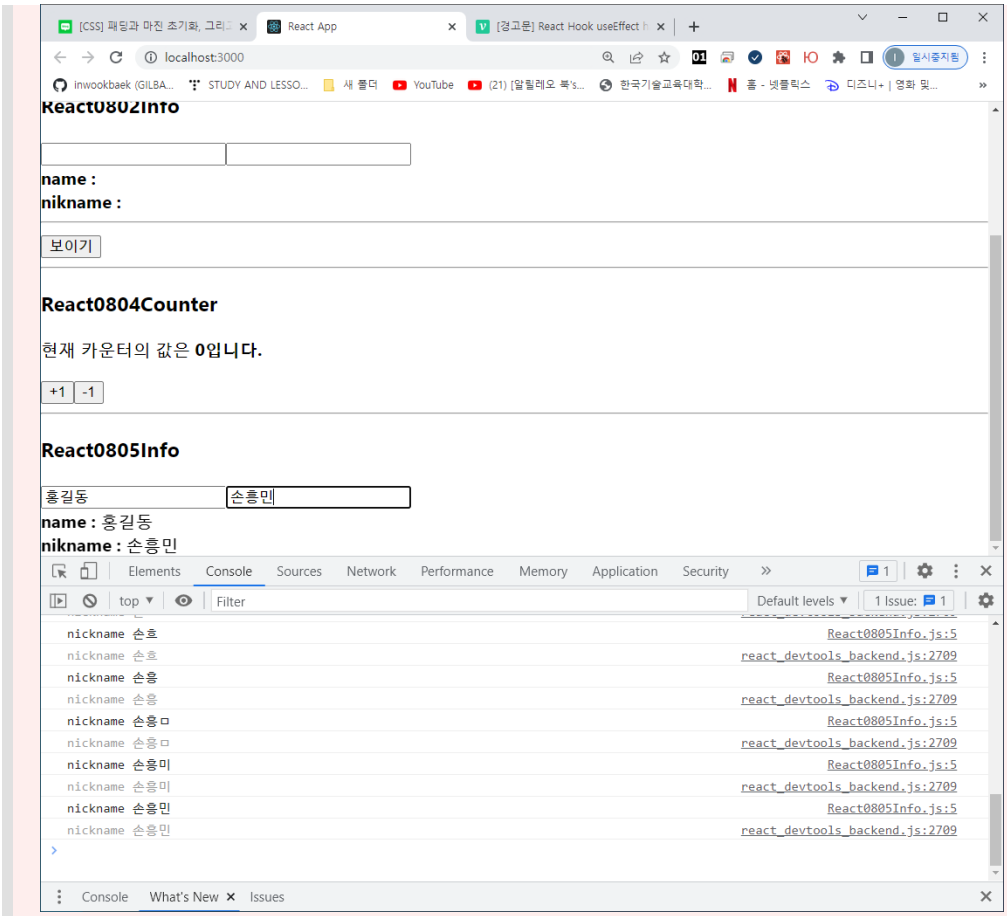
```
import { useReducer } from "react";

function reducer(state, action) {
```

```
  console.log(action.name, action.value);

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

const React0805Info = () => {

  const [state, dispatch] = useReducer(reducer, {
    name: '',
    nickname: ''
  });

  const {name, nickname} = state;

  const onChange = e => {
    dispatch(e.target);
  }

  return (
    <div>
      <h3>React0805Info</h3>
      <div>
        <input type="text" name="name" value={name} onChange={onChange} />
        <input type="text" name="nickname" value={nickname} onChange=
{onChange} />
      </div>
      <div>
        <b>name : </b> {name}
      </div>
      <div>
        <b>nikname : </b> {nickname}
      </div>
    </div>
  );
}


export default React0805Info;
```

App.js

```
import { useState } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';
import React0803Info from './mysrc/React0803Info';
import React0804Counter from './mysrc/React0804Counter';

const App = () => {

  const [visible, setVisible] = useState(false);
  return (
    <div>
        <React0801Counter />
        <hr/>
```

```
            <React0802Info />
            <hr/>
            <button onClick={() => {
              setVisible(!visible)
            }}>
              {visible ? '숨기기' : '보이기'}
            </button>
            { visible && <React0803Info /> }
            <hr/>
            <React0804Counter />
            <hr/>
            <React0805Info />
        </div>
    );
}
export default App;
```



8.4 useMemo

- 참고 : https://velog.io/@jinyoung985/React-useMemo란
- useMemo를 사용하면 함수 컴퍼넌트에서 발생하는 연산을 최작화할 수 있다.

React0806Average.js

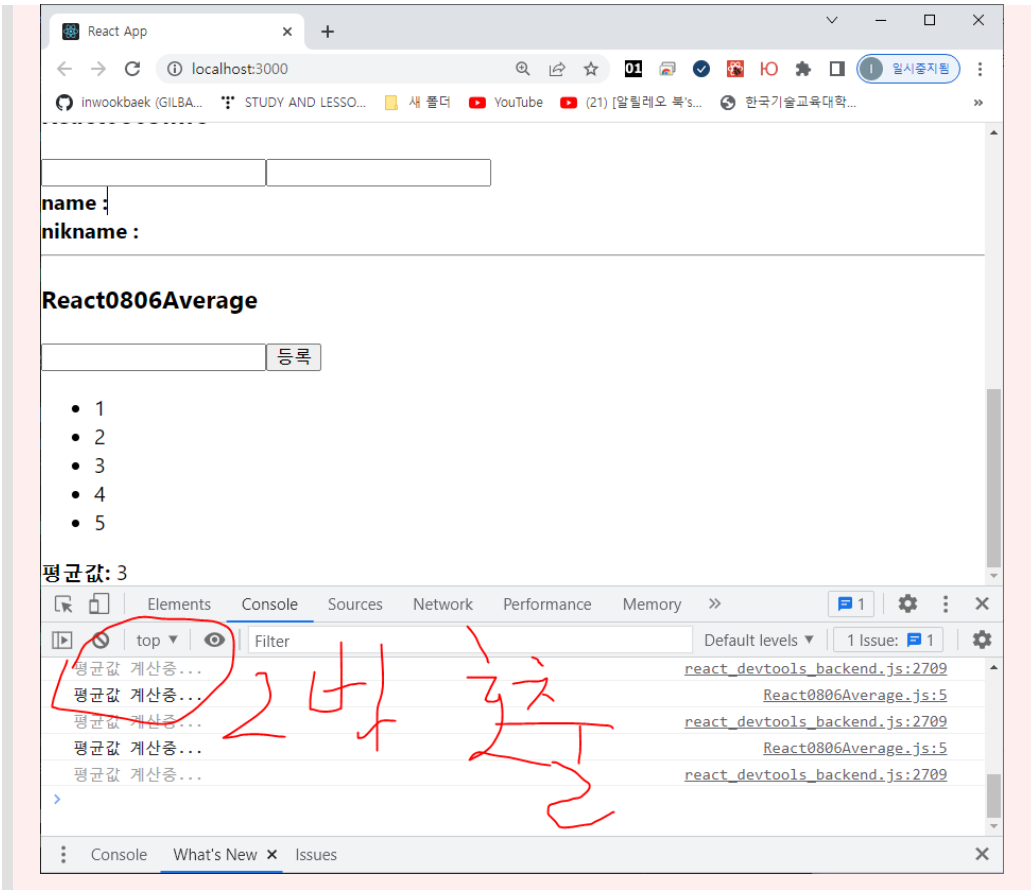
```
import { useState } from 'react';

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 React0806Average = () => {

  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('');
  }
  return (
    <div>
      <h3>React0806Average</h3>
      <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> {getAverage(list)}
      </div>
    </div>
  );
};

export default React0806Average;
```

App.js

```
import { useState } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';
import React0803Info from './mysrc/React0803Info';
import React0804Counter from './mysrc/React0804Counter';
import React0805Info from './mysrc/React0805Info';
import React0806Average from './mysrc/React0806Average';

const App = () => {

  const [visible, setVisible] = useState(false);
  return (
    <div>
        <React0801Counter />
        <hr/>
        <React0802Info />
        <hr/>
        <button onClick={() => {
          setVisible(!visible)
        }}>
```

```
            {visible ? '숨기기' : '보이기'}
          </button>
          { visible && <React0803Info /> }
          <hr/>
          <React0804Counter />
          <hr/>
          <React0805Info />
          <hr/>
          <React0806Average />
        </div>
    );
}
export default App;
```



- 상기코드의 문제는 숫자를 등록할 때뿐만 아니라 `input내용변경될 때마다 getAverage함수가 호출 된다.`
- input내용이 변경될 때는 랜더링 될 필요가 없기 때문에 `useMemo hook을 사용하면 이러한 작업을 최소화` 할 수 있다.
- 랜더링과정에서 특정값이 변경되었을 때만 연산을 실행하고 원하는 값이 변경되지 않았다면 연산했던 결과를 다시 사용하는 방식

React0806Average.js

```
import { useState, useMemo } 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 React0806Average = () => {

  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]);
  //=============================================================

  return (
    <div>
      <h3>React0806Average</h3>
      <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 React0806Average;
```

- 상기 결과는 `list값이 변경될 때만 getAverage함수가 호출된다.`

8.5 useCallback

- 참고 : https://velog.io/@jinyoung985/React-useCallback이란
- useCallback은 useMemo와 상다히 비슷한 함수, 주로 랜더링 성능을 최적화 하는 상황에서 사용
- `이 hook을 사용하면 기존의 함수를 재사용`할 수 있다.
- useMemo에서 정의 했던 onChange, onInsert의 경우에는 컴퍼넌트가 `리랜더링될 때마다 새로 생성된 함수를 사용`
- 이럴 경우엔 랜더링이 자주 되거나 랜더링해야할 함수의 갯수가 많아 지면 이 부분을 최적화 해 주는 것이 좋다.

React0806Average.js

- useCallback(생성하고자하는 함수, 배열) 형태로 사용
- 배열에는 어떤 값이 변경되었을 때 함수를 새로 생성해야 하는지 명시해야 한다.
- onChange처럼 빈 배열을 정의할 경우 컴퍼넌트가 랜더링될 때 만들었던 함수를 재사용한다.
- onInsert의 배열안에 값을 정의할 경우 해당 값이 변경되거나 새로운 값이 추가될 때 새로 만들어진 함수를 사용

```
import { useState, useMemo, 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 React0806Average = () => {

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

  //=============================================================
  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('');
  }, [number, list]); // number 혹은 list 가 바뀌었을 때만 함수 생성
  //=============================================================

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

  return (
    <div>
      <h3>React0806Average</h3>
      <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>
  );
};
```

8.6 useRef

- 참고 : https://velog.io/@jinyoung985/React-useRef란
- useRef함수는 함수 컴퍼넌트에서 ref를 쉽게 사용할 수 있도록 해 준다.
- useRef는 `저장공간 또는 DOM요소에 접근하기 위해 사용되는 React Hook`이다. 여기서 Ref는 reference, 즉 참조를 뜻한다.
- `useRef로 관리하는 값은 값이 변해도 화면이 렌더링되지 않는다.`

React0806Average.js

```
import { useState, useMemo, useCallback,useRef } 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 React0806Average = () => {

  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>
      <h3>React0806Average</h3>
//=============================================================
      <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 React0806Average;
```

8.6.1 로컬변수 사용하기

- 컴퍼넌트의 `로컬변수를 사용할 때도 useRef를 사용` 할 수 있다. `로컬변수란 랜더링과 상관없이 변경할 수 있는 값을 의미` 한다.
- 클래스 컨퍼넌트에서 로컬변수를 사용할 때 다음과 같이 작성할 수 있다.
- 이렇게 할 경우 `ref안의 값이 변경되어도 컴퍼넌트가 랜더링되지 않는다.`

예시코드

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

export default class MyComponent extends Component {

  id = 1
  setId = n => { this.id = id }
  parentId = () =>  console.log(this.id)
  render() {
    return (
      <div>MyComponent</div>
    )
  }
}

export default MyComponent
```

```
import React from 'react'

const a = () => {

  const id = useRef(1);
  const setId = n => id.current = n;
  const parentId = () => console.log(id.current)
  return (
    <div>a</div>
  )
}

export default a
```

8.7 custom hook 만들기

- 기존의 React08xxInfo 컴퍼넌트에서 여러 개의 input을 관리하기 위한 useReducer로 작성했던 로직을 useInputs라는 Hook을 작성

React0806useInputs.js

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

function reducer(state, action) {
```

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

export default function React0807useInputs(initialForm) {

  const[state, dispatch] = useReducer(reducer, initialForm);

  const onChange = e => {
    dispatch(e.target);
  }

  return [state, onChange];
}
```

React0808Info.js

```
import React0807useInputs from './React0807useInputs';

const React0808Info = () => {

  const [state, onChange] = React0807useInputs({
    name: '',
    nickname: ''
  });
  const {name, nickname} = state;

  return (
    <div>
      <h3>React0805Info</h3>
      <div>
        <input type="text" name="name" value={name} onChange={onChange} />
        <input type="text" name="nickname" value={nickname} onChange=
{onChange} />
      </div>
      <div>
        <b>name : </b> {name}
      </div>
      <div>
        <b>nikname : </b> {nickname}
      </div>
    </div>
  );
}

export default React0808Info;
```

App.js

```
import { useState } from 'react';
import React0801Counter from './mysrc/React0801Counter';
import React0802Info from './mysrc/React0802Info';
import React0803Info from './mysrc/React0803Info';
import React0804Counter from './mysrc/React0804Counter';
import React0805Info from './mysrc/React0805Info';
import React0806Average from './mysrc/React0806Average';
import React0808Info from './mysrc/React0808Info';
```

```
const App = () => {

  const [visible, setVisible] = useState(false);
  return (
    <div>
        <React0801Counter />
        <hr/>
        <React0802Info />
        <hr/>
        <button onClick={() => {
          setVisible(!visible)
        }}>
          {visible ? '숨기기' : '보이기'}
        </button>
        { visible && <React0803Info /> }
        <hr/>
        <React0804Counter />
        <hr/>
        <React0805Info />
        <hr/>
        <React0806Average />
        <hr/>
        <React0808Info />
    </div>
  );
}
export default App;
```

- 다른 개발자가 만든 Hooks
  - https://nikgraf.github.io/react-hooks/
  - http://github.com/rehooks/awesome-react-hooks

8.8 정리

- 리액트에서 Hooks패턴을 이용하면 클래스컴퍼넌트를 작성하지 않고 대부분의 기능을 구현할 수 있다.
- 기존의 setState를 사용하는 방식도 잘못된 것은 아니나 useState, useReducer 통해 구현을 권장
- 새로 작성하는 컴퍼넌트의 경우 `함수 컴퍼넌트오 Hooks을 사용할 것을 권장`