

## 6. 컴포넌트 반복

---

작성일시: 2022년 5월 9일 오전 11:21 참고 도서: 리엑트를 다루는 기술

### 6.0 IterationSample 예제

IterationSample.js

```
const IterationSample = () =>{
  return(
    <ul>
      <li>눈사람</li>
      <li>얼음</li>
      <li>눈</li>
      <li>바람</li>
    </ul>
  );
};
export default IterationSample;
```

### 6.1 자바스크립트 배열의 map() 함수

#### 6.1.1 문법

```
arr.map(callback, [thisArg])
```

#### 6.1.2 예제

```
var number = [1,2,3,4,5];

var processed = number.map(function(num){
  return num*num;
});

console.log(processed);
```

```
> var number = [1,2,3,4,5];

var processed = number.map(function(num){
  return num*num;
});

console.log(processed);
```

```
▶ (5) [1, 4, 9, 16, 25]
```

```
< undefined
```

```
> |
```

```
const numbers = [1,2,3,4,5];
const result = numbers.map(num=> num*num);

console.log(result);
```

```
> const numbers = [1,2,3,4,5];
const result = numbers.map(num=> num*num);

console.log(result);
```

```
▶ (5) [1, 4, 9, 16, 25]
```

```
VM85:4
```

```
< undefined
```

```
> |
```

## 6.2 데이터 배열을 컴포넌트 배열로 배열로 변환하기

### 6.2.1 컴포넌트 수정하기

```
const IterationSample = () =>{
  const names = ['눈사람', '얼음', '눈', '바람'];
  const nameList = names.map(name=><li>{name}</li>);
  return(
    <ul>{nameList}</ul>
    /*배열 (JSON 코드로된) */
  );
};
export default IterationSample;
```

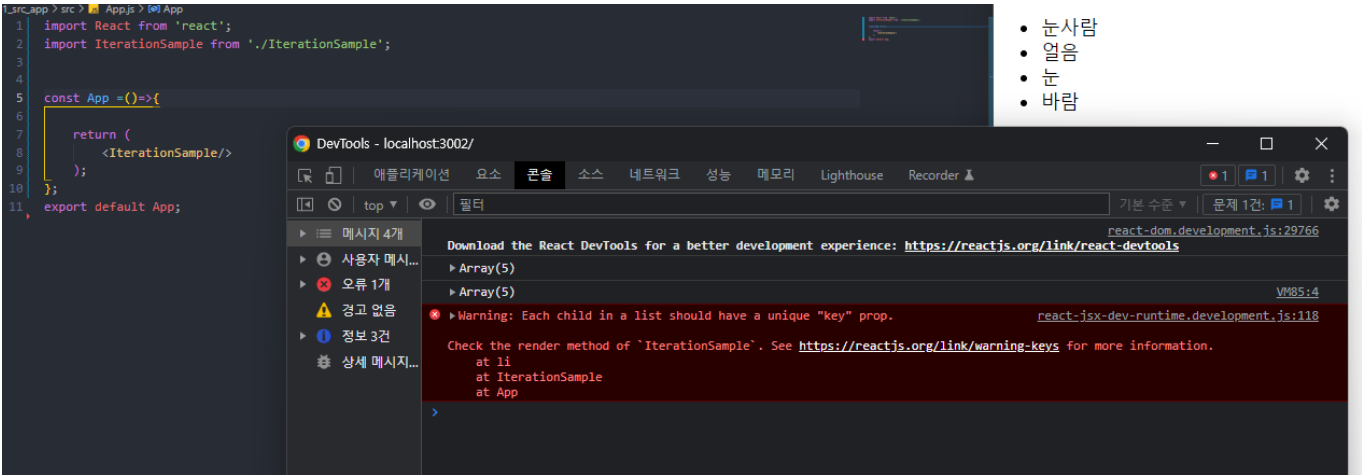
### 6.2.2 App 컴포넌트에서 예제 컴포넌트 렌더링

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

const App = ()=>{

  return (
```

```
        <IterationSample/>
      );
    };
    export default App;
```



## 6.3 key

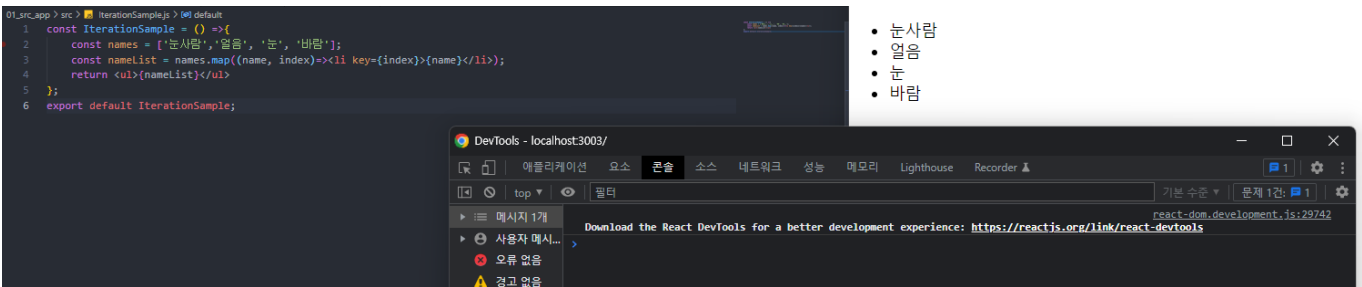
### key란?

컴포넌트 배열을 렌더링 했을 때 어떤 원소에 변동이 있었는지 알아내려고 사용한다.

key 값을 설정할 때는 map 함수의 인자로 전달되는 함수 내부에서 컴포넌트 props를 설정하듯이 설정하면 된다. key값은 언제나 유일해야한다. 따라서 데이터가 가진 고유값을 key값으로 설정 해야한다.

### 6.3.1 key 설정

```
const IterationSample = () => {
  const names = ['눈사람', '얼음', '눈', '바람'];
  const nameList = names.map((name, index) => <li key={index}>{name}</li>);
  return <ul>{nameList}</ul>
};
export default IterationSample;
```



## 6.4 응용

### 6.4.1 초기 상태 설정하기

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

const IterationSample = () => {
  const [names, setNames] = useState([
    {id: 1, text:'눈사람'},
    {id:2, text: '얼음'},
    {id: 3, text:'눈'},
    {id: 4, text:'바람'}
  ]);
  const [inputText, setInputText] = useState('');
  const [nextId, setNextId] = useState(5); //새로운 항목을 추가할 때 사용할 id

  const namesList = names.map(name => <li key={name.id}>{name.text}</li>);
  return <ul>{namesList}</ul>

}

export default IterationSample;
```



```
1 import React,{useState}from 'react'
2
3 const IterationSample = () => {
4   const [names, setNames] = useState([ 'setNames' is assigned a value but never used.
5     {id: 1, text:'눈사람'},
6     {id:2, text: '얼음'},
7     {id: 3, text:'눈'},
8     {id: 4, text:'바람'}
9   ]);
10  const [inputText, setInputText] = useState(''); 'inputText' is assigned a value but never used.
11  const [nextId, setNextId] = useState(5); //새로운 항목을 추가할 때 사용할 id 'nextId' is assigned a value but
12
13  const namesList = names.map(name => <li key={name.id}>{name.text}</li>);
14
15  return (
16    <ul>{namesList}</ul>
17  )
18 }
19
20 export default IterationSample;
```

- 눈사람
- 얼음
- 눈
- 바람

## 6.4.2 데이터 추가 기능 구현하기

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

const IterationSample = () => {
  const [names, setNames] = useState([
    {id: 1, text:'눈사람'},
    {id:2, text: '얼음'},
    {id: 3, text:'눈'},
    {id: 4, text:'바람'}
  ]);
  const [inputText, setInputText] = useState('');
  const [nextId, setNextId] = useState(5); //새로운 항목을 추가할 때 사용할 id

  const onChange = e => setInputText(e.target.value);
  //setInputText값을 inputText로 전달
```

```

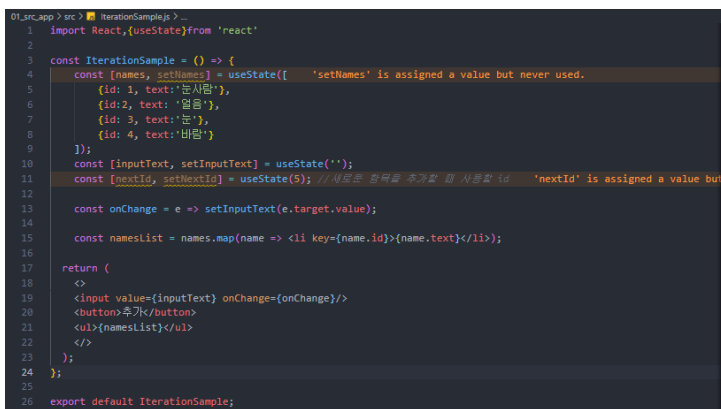
    const namesList = names.map(name => <li key={name.id}>{name.text}</li>);

    return (
      <>
        <input value={inputText} onChange={onChange}/>

        {/*onChange 값을 setInputText로 전달*/}
        <button>추가</button>
        <ul>{namesList}</ul>
      </>
    );
  };

  export default IterationSample;

```



```

01_src_app > src > IterationSample.js > ...
1  import React,{useState}from 'react'
2
3  const IterationSample = () => {
4    const [names, setNames] = useState([
5      {id: 1, text: '눈사람'},
6      {id: 2, text: '얼음'},
7      {id: 3, text: '눈'},
8      {id: 4, text: '바람'}
9    ]);
10   const [inputText, setInputText] = useState('');
11   const [nextId, setNextId] = useState(5); // 새로운 항목을 추가할 때 사용할 id
12
13   const onChange = e => setInputText(e.target.value);
14
15   const namesList = names.map(name => <li key={name.id}>{name.text}</li>);
16
17   return (
18     <>
19       <input value={inputText} onChange={onChange}/>
20       <button>추가</button>
21       <ul>{namesList}</ul>
22     </>
23   );
24 };
25
26 export default IterationSample;

```

- 눈사람
- 얼음
- 눈
- 바람

추가

## concat을 이용한 배열 만들기

```

import React,{useState}from 'react'

const IterationSample = () => {
  const [names, setNames] = useState([
    {id: 1, text: '눈사람'},
    {id: 2, text: '얼음'},
    {id: 3, text: '눈'},
    {id: 4, text: '바람'}
  ]);
  const [inputText, setInputText] = useState('');
  const [nextId, setNextId] = useState(5); // 새로운 항목을 추가할 때 사용할 id

  const onChange = e => setInputText(e.target.value);
  const onClick = () => {
    const nextNames = names.concat({ // concat은 원본+추가된 복사값을 만든다.
      id: nextId, //nextId값을 id로 설정
      text: inputText
    });
    setNextId(nextId + 1); //nextId 값에 1을 더해준다.
  };

```

```

    setNames(nextNames); //names값을 업데이트 한다.
    setInputText(''); //inputText를 비운다.
  };

  const namesList = names.map(name => <li key={name.id}>{name.text}</li>);

  return (
    <>
      <input value={inputText} onChange={onChange}/>
      <button onClick={onClick}>추가</button>
      <ul>{namesList}</ul>
    </>
  );
};

export default IterationSample;

```

```

01_src_app > src > IterationSample.js > IterationSample > namesList > names.map() callback
1  import React,{useState}from 'react'
2
3  const IterationSample = () => {
4    const [names, setNames] = useState([
5      {id: 1, text: '눈사람'},
6      {id: 2, text: '얼음'},
7      {id: 3, text: '눈'},
8      {id: 4, text: '바람'}
9    ]);
10   const [inputText, setInputText] = useState('');
11   const [nextId, setNextId] = useState(5); // 새로운 항목을 추가할 때 사용할 id
12
13   const onChange = e => setInputText(e.target.value);
14   const onClick = () => {
15     const nextNames = names.concat({ // concat은 원본+추가된 복사값을 만든다.
16       id: nextId, //nextId값을 id로 설정
17       text: inputText
18     });
19     setNextId(nextId +1); //nextId 값에 1을 더해준다. "next": Unknown word.
20     setNames(nextNames); //names값을 업데이트 한다.
21     setInputText(''); //inputText를 비운다.
22   };
23
24   const namesList = names.map(name => <li key={name.id}>{name.text}</li>);
25
26   return (
27     <>
28       <input value={inputText} onChange={onChange}/>
29       <button onClick={onClick}>추가</button>
30       <ul>{namesList}</ul>
31     </>
32   );
33 };
34
35 export default IterationSample;

```

추가

- 눈사람
- 얼음
- 눈
- 바람
- 눈사람
- 크리스마스
- 밤
- 낮

### 6.4.3 데이터 제거 기능 구현하기

```

import React,{useState}from 'react'

const IterationSample = () => {
  const [names, setNames] = useState([
    {id: 1, text: '눈사람'},
    {id: 2, text: '얼음'},
    {id: 3, text: '눈'},
    {id: 4, text: '바람'}
  ]);
  const [inputText, setInputText] = useState('');
  const [nextId, setNextId] = useState(5); //새로운 항목을 추가할 때 사용할 id

```

```

const onChange = e => setInputText(e.target.value);
const onClick = () => {
  const nextNames = names.concat({ // concat은 원본+추가된 복사값을 만든다.
    id: nextId, //nextId값을 id로 설정
    text: inputText
  });
  setNextId(nextId + 1); //nextId 값에 1을 더해준다.
  setNames(nextNames); //names값을 업데이트 한다.
  setInputText(''); //inputText를 비운다.
};

const onRemove = id =>{
  const nextNames = names.filter(name => name.id !== id);
  setNames(nextNames);
};

const namesList = names.map(name => (
  <li key={name.id} onClick={() => onRemove(name.id)}>
    {name.text}
  </li>
));

return (
  <>
    <input value={inputText} onChange={onChange}/>
    <button onClick={onClick}>추가</button>
    <ul>{namesList}</ul>
  </>
);
};

export default IterationSample;

```

```

src_app > src > IterationSample.js > IterationSample
1  import React,{useState}from 'react'
2
3  const IterationSample = () => {
4    const [names, setNames] = useState([
5      {id: 1, text: '눈사람'},
6      {id: 2, text: '얼음'},
7      {id: 3, text: '눈'},
8      {id: 4, text: '바람'}
9    ]);
10   const [inputText, setInputText] = useState('');
11   const [nextId, setNextId] = useState(5); // 새로운 항목을 추가할 때 사용할 id
12
13   const onChange = e => setInputText(e.target.value);
14   const onClick = () => {
15     const nextNames = names.concat({ // concat은 원본+추가된 복사본을 만든다.
16       id: nextId, //nextId값을 id로 설정
17       text: inputText
18     });
19     setNextId(nextId +1); //nextId 값에 1을 더해준다. "next": Unknown word.
20     setNames(nextNames); //names 값을 업데이트 한다.
21     setInputText(''); //inputText를 비운다.
22   };
23
24   const onRemove = id =>{
25     const nextNames = names.filter(name => name.id !== id);
26     setNames(nextNames);
27   };
28
29   const namesList = names.map(name => (
30     <li key={name.id} onDoubleClick={() => onRemove(name.id)}>
31       {name.text}
32     </li>
33   ));
34
35   return (
36     <>
37       <input value={inputText} onChange={onChange}/>
38       <button onClick={onClick}>추가</button>
39       <ul>{namesList}</ul>
40     </>
41   );
42 };
43
44 export default IterationSample;

```

추가

- 눈사람

## 6.5 정리

컴포넌트 배열을 렌더링 할 때는 key 값 설정에 항상 주의해야한다. key 값은 언제나 유일해야한다.

상태 안에서 배열을 변형할 때는 배열에 직접 접근하여 수정하는 것이 아니라 concat,filter등의 배열 내장 함수를 사용하여 새로운 배열을 만든 후 이를 새로운 상태로 설정해줘야한다.

배열, JSON ⇒ 참조복사를 탈피하기 위한 것은 비구조 문법이다. let k = [...] let x = k 이렇게 하면 안됨.