React component key에 대하여 .

회사는 바삐 돌아가고, 공부 할 내용은 많아, 기획 되어있던 시리즈가 진도를 못나가고 있다. 그래서 많은 이들이 실수 할 만한 내용에 대해서 작성해보려 한다. 이 내용은 타입스크립트로 작성되어 있는 함수형 컴포넌트를 사용하지만, REACT의 특징을 어느정도 익힌 사람이라면 이해 할 수 있도록 작성되어 있다.

## Map 을 통한 컴포넌트로의 매핑

Map 메서드는 배열을 재구성하는 용도로 사용되는 배열 메서드이다. 배열 타입이라면 map 메서드를 가지고 있다.

혹시라도 map 메서드를 사용하지 않은 사람을 위해 간단한 예를 들어보자 .

**Common.js에서의 사용**

const numbers = [1,2,3,4,5];

const addedTwoInNumbers = numbers.map((number)=>{

return number + 2;

})

console.log(addedTwoInNumbers) // 3,4,5,6,7

const addedIndexInNumbers = numbers.map((number,index)=>{

return number + index;

})

console.log(addedIndexInNumbers) // 1, 3, 5, 7, 9

map 함수는 콜백 함수를 첫번째 인자로 전달 받고, 이 콜백의 첫번째 인자는 순회되는 배열의 버킷, 즉 값이다. 두번째 인자는 현재 순회되고 있는 버킷의 index이다. 이를 조금 더 쉽게 for문으로 작성하면 다음과 같다.

function \_\_map(callBack){

let \_\_returnArray = [];

for(let i = 0; i < this.length; i++){

\_\_returnArray[i] = callBack(this[i],i)

}

return \_\_returnArray;

}

Array.prototype.\_\_map = \_\_map;

**React에서 사용**

React에서 JSX문법을 사용할 때 배열 내부에 컴포넌트들이 존재한다면 이를 render의 return 내부에 전달하게 될 경우, 해당 배열에 있는 컴포넌트들이 JSX문법에 따라 차례대로 나열되게 된다.

import React, {useState} from 'react';

import logo from './logo.svg';

import Tab from './Todo';

import './App.css';

interface TodoInterface {

value : string,

done : boolean

}

const App = ()=>{

const [value, setValue] = useState("");

const [todos, setTodos] = useState<TodoInterface[]>([]);

const create : (todo : string) => void = (value : string)=>{

const todo : TodoInterface= {value : value, done : false}

setTodos([...todos,todo])

}

const todoDelete : (index : number) => void = (index : number) =>{

setTodos([

...todos.slice(0,index),

...todos.slice(index + 1, todos.length)

])

}

return (

<>

<div>

<h1>Todos</h1>

<input type="text" value={value} onChange={(e)=>setValue(e.target.value)}/><button onClick={()=>create(value)}>생성</button>

{todos.map((todo,index)=>{

return <Tab todo={todo.value} index={index}delete={()=>todoDelete(index)}/>

})}

</div>

</>

)

}

export default App;

위의 코드는 Todos 배열에 존재하는 Todo 타입의 객체를 컴포넌트로 변경 후 return 하는 코드이다. Todo 컴포넌트 또한 보자.

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

interface TodoTabPropsInterface {

todo : string,

delete : () => void,

index : number

}

const TodoTab = (props : TodoTabPropsInterface) =>{

const { todo,index } = props;

return (

<div>

{index}. : {todo}<button onClick={props.delete}>삭제</button>

</div>

)

}

export default TodoTab;

단순히 props로 value : string을 받고 이에 대하여, 화면에 Tab형태로 나타나는 코드이다. 여기 까지 똑같이 작성하였다면 크롬 검사창에는 다음과 같은 경고 메세지가 우리를 반겨준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

즉 배열을 이용한 리스트 형태는 key prop을 전달하라는 메세지 이다. 경고가 나왔으니 삭제나 삽입을 시도 해보자 .



삽입이나 삭제도 별 문제가 없다. 그럼 여기서 Tab의 텍스트 영역을 클릭하게 되면 line-throgh가 되도록 Tab에 State를 넣어보자.

const TodoTab = (props : TodoTabPropsInterface) =>{

const [done, setdone] = useState(false)

const { todo,index } = props;

return (

<div>

{index}. : <label style={done ? {"textDecoration" : "line-through"} : {}}

onClick={()=>setdone(!done)}>

{todo}

</label><button onClick={props.delete}>삭제</button>

</div>

)

}

자 실행하니 별 문제 없이 state에 따라 작동한다. 그러나 문제는 다음이다. 어느 Tab의 label을 클릭 후 done을 true로 만든 다음, 해당 Tab을 삭제해보자.



저주 받은 코드인가 ? 2번 Tab의 done을 true로 한 뒤 삭제하니 3번에 있었던 라벨이 2번으로 오며, state가 동일하게 적용 되었다. 혹시 모르니 key를 index로 만들어 보자 .

{todos.map((todo,index)=>{

return <Tab todo={todo.value} key={index} index={index}delete={()=>todoDelete(index)}/>

})}

이번에는 경고 메세지가 아닌 다른 다른 메세지가 콘솔창에 출력된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Index를 넣으니 index를 사용하지 말라고 한다. 그렇다면 Tab-index 형태를 사용하여, key를 넣어보자.

{todos.map((todo,index)=>{

return <Tab todo={todo.value} key={`Tab-${index}`} index={index}delete={()=>todoDelete(index)}/>

})}

동일한 경고 메세지는 크롬 콘솔에 출력된다. 그렇다면 작동 하기는 할까 ?



동일한 결과를 보여준다. 그렇다면 이번에는 useEffect를 사용해서 삭제되는 컴포넌트의 index를 확인해보자 .

const TodoTab = (props : TodoTabPropsInterface) =>{

const [done, setdone] = useState(false)

useEffect(()=>{

console.log("component did mount")

},[])

useEffect(()=>{

return () =>{

console.log(`todo tab - ${props.index}component will unmount`);

}

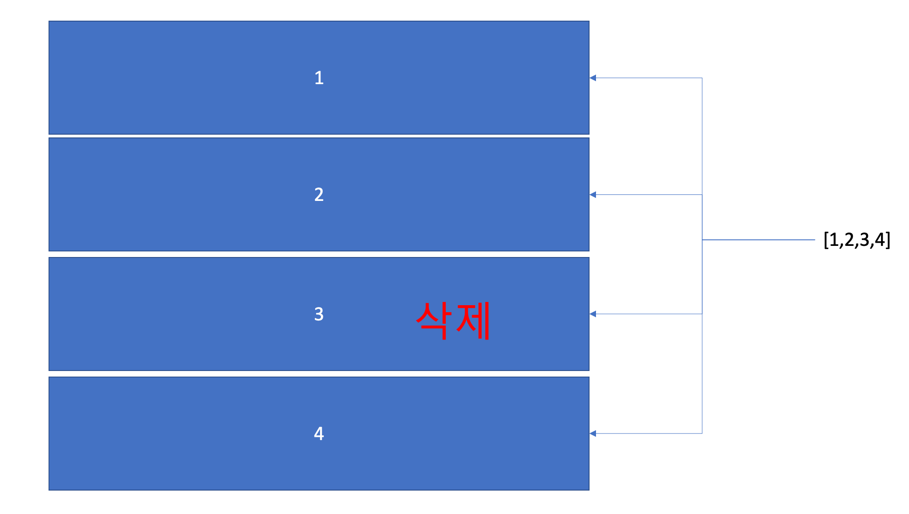
},[])

return {…}

텍스트이(가) 표시된 사진

자동 생성된 설명

3번을 삭제하게 되면 분명 todo tab-3 component will unmount 메세지가 출력되어야 한다. 그러나 결과는 5 번이 언마운트 되었다고 출력된다. 이유는 다음과 같다.



**배열로 구현된 list는 key라는 props로 state를 유지하고 구현한다.** 만약 key로 아무것도 전달하지 않게 되면 key는 자동으로 indexing되어 버킷의 인덱스 번호와 일치하게 된다.



이때 3번의 Tab을 삭제하게 된다면 자동적으로 다시 배열은 [1,2,4]로 변경되며, 각각의 탭은 key를 indexing하게되어 1,2,3의 key를 가지게 될 것이다. 그렇기 때문에 3번을 삭제한다하여도, 3번의 key는 유지되고, state가 유지되기 때문에 value가 4가 되어도 여전히 3번의 state를 유지하게 되는 것이다.

## 그렇다면 방법은 ?

당연하게도 유효하며 리스트 중 유일무이한 값을 key로 전달하면 된다. 이를 위하여, 위의 예제에서는 입력 시 millisecond까지 가지고 있는 timestamp를 전달하도록 하자 .

먼저 인터페이스에 timeStamp를 추가한다.

interface TodoInterface {

value : string,

done : boolean,

timeStamp : string

}

Todo객체를 만들 때 date객체에서 getTime()메서드를 이용하여, 꽤나 유효한 key를 만들도록 하자.

const todo : TodoInterface= {value : value, done : false, timeStamp : String(new Date().getTime())}

이제 Todo객체를 생성 후 콘솔창을 보니 더 이상 경고메세지가 나타나지 않는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 특정 Todo를 클릭 후에 그 Todo를 삭제해보도록 하자 .

테이블이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

동일한 Todo를 삭제 했음에도 삭제했던 컴포넌트의 state가 더 이상 유지되지 않고 , 정확한 component가 삭제되었음을 알 수 있다.

**참고하면 좋은 자료들**

[**https://robinpokorny.medium.com/index-as-a-key-is-an-anti-pattern-e0349aece318**](https://robinpokorny.medium.com/index-as-a-key-is-an-anti-pattern-e0349aece318)

[**https://ko.reactjs.org/docs/lists-and-keys.html**](https://ko.reactjs.org/docs/lists-and-keys.html)