React.memo와 렌더링 최적화

**학습목표**

React.memo를 사용하여 컴포넌트 렌더링 최적화 하기

**목차**

[1. 렌더링 최적화 2](#_Toc149138502)

[1) 개요 2](#_Toc149138503)

[2) Counter 예제 구현 4](#_Toc149138504)

[3) 실행 6](#_Toc149138505)

[2. useState 훅 예제 렌더링 최적화 7](#_Toc149138506)

[1) 개요 7](#_Toc149138507)

[2) javascript의 closure 7](#_Toc149138508)

[3) todo 앱 수정 8](#_Toc149138509)

[4) 실행 12](#_Toc149138510)

[5) 렌더링 최적화가 안되는 이유 13](#_Toc149138511)

[6) 결론 14](#_Toc149138512)

[3. useReducer 훅과 렌더링 최적화 15](#_Toc149138513)

[1) 개요 15](#_Toc149138514)

[2) 수정 15](#_Toc149138515)

[3) 실행 18](#_Toc149138516)

[4) 결론 18](#_Toc149138517)

[4. useReducer 코드 개선 19](#_Toc149138518)

[1) 개요 19](#_Toc149138519)

[2) 수정 21](#_Toc149138520)

[5. useCallback 훅 23](#_Toc149138521)

[1) 개요 23](#_Toc149138522)

[2) useCallback 훅 25](#_Toc149138523)

[6. useState + useCallback 렌더링 최적화 26](#_Toc149138524)

[7. 예제 30](#_Toc149138525)

[1) 학생 목록 - useReducer 30](#_Toc149138526)

[2) 제품 목록 - useState 34](#_Toc149138527)

# 렌더링 최적화

## 개요

리액트 컴포넌트가 렌더링 하는 내용은 다음 두 데이터에 의해 결정된다.

a) 컴포넌트가 생성한 리액트 상태 (useState)

b) 부모 컴포넌트로부터 전달 받은 props

따라서 위 두 값 중 하나가 변경되면,

컴포넌트는 다시 렌더링 되어야 한다.

부모 컴포넌트가 렌더링 될 때, 모든 자식 컴포넌트들도 렌더링 된다.

상태, props 둘 다 변경되지 않은 컴포넌트들은 다시 렌더링 되지 않게 만들기 -> 렌더링 최적화

### a) CounterA.tsx

|  |
| --- |
| import React, { useState } from 'react'  function CounterA() {    const [value, setValue] = useState<number>(0);    return (      <div className="box">        <h1>CounterA</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>      </div>    )  }  export default CounterA; |

버튼이 클릭되고 value 상태 값이 증가하면면 setValue(value + 1),

이 컴포넌트는 다시 렌더링 되어야 한다.

컴포넌트가 다시 렌더링 되어야 { value } 부분에 새 상태 값을 출력될 수 있다.

### b) TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type \* as types from './types'  type Props = {    todo: types.Todo  }  function TodoItem({ todo }: Props) {    return (      <tr>        <td>{ todo.id }</td>        <td>{ todo.title }</td>      </tr>    )  }  export default TodoItem; |

부모 컴포넌트로부터 전달된 todo props 값이 달라졌다면,

TodoItem 컴포넌트는 다시 렌더링 되어야 한다.

### 기본 렌더링 정책

부모 컴포넌트가 다시 렌더링 될 때, 자식 컴포넌트들도 전부 다시 렌더링 된다.

이것이 기본 렌더링 정책이다.

그래서, 새 Todo 항목이 추가되어, TodoRoot 컴포넌트의 todoList 상태가 변경되면,

TodoList 아래 모든 컴포넌트들이 다시 렌더링 되어야 한다.

그런데 새 Todo 항목만 추가 되었을뿐이고,

todoList에 이미 들어있던 Todo 항목의 값이 변경되지 않았는데,

변경되지도 않은 Todo 항목들도 다시 렌더링 되어야하지?

이것은 낭비 아닌가?

### 렌더링 최적화

부모 컴포넌트가 다시 렌더링 될 때, 자식 컴포넌트들도 전부 다시 렌더링 되는 것이 기본 정책이지만,

a) 그 컴포넌트가 관리하는 리액트 상태

b) 부모 컴포넌트로부터 전달 받은 props

위 두 값이 변경되지 않은 자식 컴포넌튼 다시 렌더링 될 필요가 없다.

위 두 값이 변경되었는지 확인해서,

불필요한 렌더링은 막아주는 리액트의 기능이 React.memo 이다.

### 렌더링 최적화가 중요한 이유

훨씬 쉽고 구현도 간편한 Vue 보다 React가 대세인 이유 중 하나는

React에서만 가능한 렌더링 최적화 때문이다.

렌더링 최적화를 하지 않을 거라면,

어렵고 불편한 React를 사용할 이유가 없다.

많은 컴포넌트들로 구성된 복잡한 스마트 폰 앱에서도 (예: facebook 앱)

매끄러운 사용자 경험을 제공하려면

렌더링 최적화가 필수이다.

## Counter 예제 구현

### src/CounterA.tsx

|  |
| --- |
| import React, { useState } from 'react'  function CounterA() {    console.log("CounterA 호출됨");    const [value, setValue] = useState<number>(0);    return (      <div className="box">        <h1>CounterA</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>      </div>    )  }  export default React.memo(CounterA); |

### src/CounterB.tsx 생성

|  |
| --- |
| import React, { useState } from 'react'  function CounterB() {    console.log("CounterB 호출됨");    const [value, setValue] = useState<number>(0);    return (      <div className="box">        <h1>CounterB</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>      </div>    )  }  export default React.memo(CounterB); |

### src/GoupComponent.tsx 생성

|  |
| --- |
| import React, { useState } from 'react'  import CounterA from './CounterA'  import CounterB from './CounterB'  function GroupComponent() {    console.log('GroupComponent 호출됨');    const [value, setValue] = useState<number>(0);    return (      <div className='box'>        <h1>GroupComponent</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>        <CounterA />        <CounterB />      </div>    )  }  export default React.memo(GroupComponent); |

### src/App.tsx 수정

|  |
| --- |
| import React, { useState } from 'react'  import './App.css'  import GroupComponent from './GroupComponent';  function App() {    console.log('App 호출됨');    const [value, setValue] = useState<number>(0);    return (      <div className="box">        <h1>App</h1>        <button onClick={() => setValue(value + 1)}>          { value }        </button>        <GroupComponent />      </div>    );  }  export default React.memo(App); |

### src/App.css 수정

|  |
| --- |
| h1 { font-size: 11pt; margin: 0; }  div.box { display: inline-block; margin: 10px; padding: 10px;    border: 1px solid gray; }  button { display: block; padding: 0.2em 2em; margin: 0.5em; } |

## 실행

F12 개발자의 콘솔 탭에서 컴포넌트들의 렌더링을 확인해 보자.

상태가 변경된 컴포넌트만 렌더링된다는 것을 확인할 수 있다.

부모 컴포넌트가 렌더링 될 때,

상태가 변경되지도 않은 자식 컴포넌트들은 렌더링 되지 않는다.

# useState 훅 예제 렌더링 최적화

## 개요

useState 훅으로 구현한 todo 앱 예제에 렌더링 최적화를 구현해 보자.

todo 항목 중 한 개의 체크 박스를 클릭했을 때,

상태가 변경된 그 TodoItem 컴포넌트만 렌더링되고,

나머지 TodoItem 컴포넌트는 렌더링 될 필요 없다.

그렇게 되도록 최적화 해보자.

## javascript의 closure

### closure1.js

|  |
| --- |
| function outterFunc(a) {      let innerFunc = (i) => i \* a;      return innerFunc;  }  const f1 = outterFunc(2);  const f2 = outterFunc(4);  console.log(f1(3));  console.log(f2(3)); |

출력

|  |
| --- |
| 6  12 |

outterFunc 함수는 innerFunc 함수를 만들어 리턴한다.

let innerFunc = (i) => i \* a;

이 코드가 실행될 때 마다, 새 함수가 생성되어 innerFunc 지역 변수에 대입된다.

이렇게 만들어지는 함수의 본문에서 a 지역 변수를 사용하고 있다.

a 지역 변수는 outterFunnc 함수의 지역 변수이다.

inner 함수에서 outter 함수의 지역 변수의 값을 읽을 수는 있고, 수정할 수는 없다.

inner 함수가 만들어지는 그 시점의 a 지역 변수 값이 inner 함수 내부에 캡처 된다.

inner 함수 본문에서 outter 함수의 지역 변수 값을 읽는다면,

inner 함수가 만들어지는 시점의 outter 함수 지역 변수의 값이 inner 함수 내부에 캡처 된다.

이렇게 outter 함수 지역 변수의 값을 캡처한 inner 함수를 closure라고도 부른다.

const f1 = outterFunc(2);

f1 변수에 대입되는 함수는 (closure)는 (i) => i \* 2 이다.

const f2 = outterFunc(4);

f2 변수에 대입되는 함수는 (closure)는 (i) => i \* 4 이다.

## todo 앱 수정

### types.ts

|  |
| --- |
| export type Todo = {    id: number,    title: string,    done: boolean  };  export type AddTodoFunc = (text: string) => void;  export type ToogleTodoFunc = (id: number) => void;  export type DeleteTodoFunc = (id: number) => void; |

전과 동일함

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type \* as types from './types'  type Props = {    todo: types.Todo,    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoItem({ todo, toggleTodo, deleteTodo }: Props) {    console.log("TodoItem", todo.id)    const onChange = () => toggleTodo(todo.id)    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id)    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(TodoItem); |

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import type \* as types from './types'  import TodoItem from './TodoItem'  type Props = {    todoList: types.Todo[],    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoList({ todoList, toggleTodo, deleteTodo }: Props) {    console.log("TodoList")    const trlist = todoList.map(todo =>      <TodoItem key={todo.id} todo={todo} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />);    return (      <table>        <thead>          <tr><td>id</td><td>할일</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default React.memo(TodoList) |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type \* as types from './types'  type Props = {    addTodo : types.AddTodoFunc  }  function TodoInput({ addTodo }: Props) {    console.log('TodoInput');    const [title, setTitle] = useState<string>('');    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => { addTodo(title); setTitle(''); }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default React.memo(TodoInput) |

### TodoRoot.tsx

|  |
| --- |
| import React from 'react'  import './TodoRoot.css'  import type \* as types from './types'  import TodoInput from './TodoInput';  import TodoList from './TodoList';  type Props = {    todoList: types.Todo[],    addTodo: types.AddTodoFunc,    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoRoot({ todoList, addTodo, toggleTodo, deleteTodo }: Props) {    console.log("TodoRoot")    return (      <div className="todoRoot">        <h1>할일</h1>        <TodoInput addTodo={addTodo} />        <TodoList todoList={todoList} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />      </div>    )  }  export default React.memo(TodoRoot); |

### TodoRoot.css

|  |
| --- |
| .todoRoot { border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px; }  .todoRoot h1 { margin: 0  0 20px 0; }  .todoRoot div > input { padding: 6px; width: 280px; }  .todoRoot div button { margin-left: 5px; padding: 3px 10px; }  .todoRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }  .todoRoot td { border: 1px solid gray; padding: 6px; }  .todoRoot thead { background-color: #eee; text-align: center; }  .todoRoot td:nth-child(1) { text-align: center; width: 30px; }  .todoRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }  .todoRoot span { float: right; font-weight: bold; cursor: pointer; }  .todoRoot tr.done { color: #bbb; text-decoration: line-through; } |

전과 동일

### TodoContainer.tsx

|  |
| --- |
| import React, { useState } from 'react'  import type \* as types from './types'  import TodoRoot from './TodoRoot';  const data : types.Todo[] = [    {id: 1, title: '프론트엔드 과제', done: false},    {id: 2, title: '백엔드 과제', done: true},    {id: 3, title: '웹프 시험공부', done: false}  ];  function TodoContainer() {    console.log("TodoContainer")    const [todoList, setTodoList] = useState<types.Todo[]>(data);    const addTodo : types.AddTodoFunc = (title: string) => {      const lastIndex = todoList.length - 1;      const lastId = todoList[lastIndex].id;      const todo = {id: lastId + 1, title: title, done: false};      setTodoList([...todoList, todo]);    }    const toggleTodo : types.ToogleTodoFunc = (id: number) => {      let newTodoList = todoList.map(        todo => todo.id !== id ? todo : {...todo, done: !todo.done }      );      setTodoList(newTodoList);    }    const deleteTodo : types.DeleteTodoFunc = (id: number) => {      setTodoList( todoList.filter(todo => todo.id !== id) );    }      return <TodoRoot todoList={todoList}                     addTodo={addTodo}                     toggleTodo={toggleTodo}                     deleteTodo={deleteTodo} />;  }  export default React.memo(TodoContainer) |

### App.tsx

|  |
| --- |
| import React from 'react';  import TodoContainer from './TodoContainer';  function App() {    console.log("App")    return (      <TodoContainer />    );  }  export default React.memo(App); |

## 실행

todo 앱을 실행하고 웹브라우저 개발자창의 console 탭 출력을 확인하자.

Graphical user interface, application

Description automatically generated

처음 렌더링될 때, 모든 컴포넌트가 다 렌더링 된다.

체크박스를 한 개 체크해보자.

A picture containing table

Description automatically generated

상태가 변경되지 않은 TodoItem 들도 전부 다시 렌더링된다.

렌더링 최적화가 안되고 있음을 알 수 있다.

## 렌더링 최적화가 안되는 이유

|  |
| --- |
| function TodoContainer() {    console.log("TodoContainer")    const [todoList, setTodoList] = useState<types.Todo[]>(data);    const addTodo : types.AddTodoFunc = (title: string) => {      const lastIndex = todoList.length - 1;      const lastId = todoList[lastIndex].id;      const todo = {id: lastId + 1, title: title, done: false};      setTodoList([...todoList, todo]);    }    const toggleTodo : types.ToogleTodoFunc = (id: number) => {      let newTodoList = todoList.map(        todo => todo.id !== id ? todo : {...todo, done: !todo.done }      );      setTodoList(newTodoList);    }    const deleteTodo : types.DeleteTodoFunc = (id: number) => {     setTodoList( todoList.filter(todo => todo.id !== id) );    }      return <TodoRoot todoList={todoList}                     addTodo={addTodo}                     toggleTodo={toggleTodo}                     deleteTodo={deleteTodo} />;  } |

(1) todo 항목의 체크 박스를 클릭하면, toggleTodo 함수가 호출된다.

(2) toggleTodo 함수는 새 배열(newTodoList)을 만들어 setTodoList 함수를 호출한다.

새 배열의 내용은 다음과 같다.

체크된 항목에 해당하는 todo 객체만 새 객체로 교체되었고, 다른 항목들의 todo 객체들은 그대로이다.

(3) setTodoList 함수가 호출되어, TodoContainer 컴포넌트의 상태가 변경되었으니,

TodoContainer 컴포넌트가 다시 렌더링 된다.

(4) TodoContainer 함수 내부의 다음 코드를 보자.

|  |
| --- |
| const addTodo : types.AddTodoFunc = (title: string) => {      const lastIndex = todoList.length - 1;      const lastId = todoList[lastIndex].id;      const todo = {id: lastId + 1, title: title, done: false};      setTodoList([...todoList, todo]);    } |

위 코드는 함수를 만들어서, addTodo 변수에 대입한다.

addTodo 함수는 TodoContainer 함수의 내부에서 만들어지므로, TodoContainer의 inner 함수이다.

위 코드를 실행할 때 마다 함수가 다시 새로 만들어진다.

여기서 todoList, setTodoList는 outter 함수인 TodoContainer의 지역 변수이다.

위 코드를 실행할 때 마다, todoList, setTodoList 지역 변수의 값이 캡처되어, 새 inner 함수가 생성된다.

<TodoRoot todoList={todoList}

          addTodo={addTodo}

           toggleTodo={toggleTodo}

           deleteTodo={deleteTodo} />

TodoContainer 컴포넌트가 다시 렌더링 될 때 마다,

addTodo, toggleTodo, deleteTodo 함수가 새로 생성되어, TodoRoot 컴포넌트의 props로 전달된다.

export default React.memo(TodoRoot);

이렇게 TodoRoot 컴포넌트에 React.memo를 적용해 보았자 아무 의미가 없다.

React.memo는 state 값이나 props 값이 전과 동일한 컴포넌트가 다시 렌더링 되지 않도록 해준다.

그런데 TodoContainer 컴포넌트가 다시 렌더링 될 때 마다,

TodoRoot의 props인 addTodo, toggleTodo, deleteTodo 값이 바뀐다.

따라서 TodoRoot도 다시 렌더링 되어야 한다.

<TodoList todoList={todoList} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />

TodoContainer 컴포넌트가 다시 렌더링 될 때 마다,

TodoList의 props인 toggleTodo, deleteTodo 값이 바뀌기 때문에,

TodoList 컴포넌트도 다시 렌더링 되어야 한다.

<TodoItem todo={todo} key={todo.id} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />

TodoContainer 컴포넌트가 다시 렌더링 될 때 마다, 모든 TodoItem 들도 다시 렌더링 되어야 한다.

todo props 값이 바뀌지 않고 그대로인 TodoItem들도 많지만,

toggleTodo, deleteTodo 값은 매번 바뀌기 때문에, 모든 TodoItem들도 렌더링 되어야 한다.

## 결론

여러 컴포넌트들로 나누어 구현되는 기능을, useState 훅으로 구현하는 것은 렌더링 최적화가 어렵기 때문에 바람직하지 않다.

useState 훅은 input, select 태그에 입력되는 값의 상태 관리 정도에만 적합하다.

한 두 컴포넌트 안에서만 사용되는 상태 관리 정도에만 useState 훅을 사용하자.

# useReducer 훅과 렌더링 최적화

## 개요

useReducer 훅으로 구현한 todo 앱 예제에 렌더링 최적화를 구현해 보자.

## 수정

todoReducer.ts

|  |
| --- |
| export type Todo = {      id: number,      title: string,      done: boolean  }  export type State = {      todoList: Todo[]  }  export const initialState = { todoList: [      {id: 1, title: '프론트엔드 과제', done: false},      {id: 2, title: '백엔드 과제', done: true},      {id: 3, title: '웹프 시험공부', done: false}    ] }    export type Action =    { type: "addTodo", payload: { title: string } } |    { type: "toggleTodo", payload: { id: number } } |    { type: "deleteTodo", payload: { id: number } }  export type DispatchFunc = (action: Action) => void;  export type ReducerFunc = (state: State, action: Action) => State;    export const reducer : ReducerFunc = (state: State, action: Action) => {      const todoList = state.todoList;      switch (action.type) {          case "addTodo":              return { todoList: [...todoList,                 { id: todoList[todoList.length-1].id + 1,                   title: action.payload.title, done: false } ] };          case "deleteTodo":              return { todoList: todoList.filter(                  todo => todo.id !== action.payload.id ) };          case "toggleTodo":              return { todoList: todoList.map(                  todo => todo.id !== action.payload.id ?                          todo :                          {...todo, done: !todo.done}) };          default:              throw new Error("unknown action type");        }  } |

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type { Todo, DispatchFunc } from './todoReducer'  type Props = {    todo: Todo,    dispatch: DispatchFunc  }  function TodoItem({ todo, dispatch }: Props) {    console.log('TodoItem', todo.id);    const onChange = () => dispatch({ type: "toggleTodo", payload: { id: todo.id } });    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        dispatch({ type: "deleteTodo", payload: { id: todo.id } });    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(TodoItem); |

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import TodoItem from './TodoItem'  import type { Todo, DispatchFunc } from './todoReducer'  type Props = {    todoList: Todo[],    dispatch: DispatchFunc  }  function TodoList({ todoList, dispatch }: Props) {    console.log('TodoList');    const trlist = todoList.map(todo =>      <TodoItem key={todo.id} todo={todo} dispatch={dispatch} />);    return (      <table>        <thead>          <tr><td>id</td><td>할일</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default React.memo(TodoList) |

### TodoRoot.tsx

|  |
| --- |
| import React, { useReducer } from 'react'  import './TodoRoot.css'  import TodoInput from './TodoInput';  import TodoList from './TodoList';  import { reducer, initialState } from './todoReducer'  function TodoRoot() {    console.log('TodoRoot');    const [state, dispatch] = useReducer(reducer, initialState)    return (      <div className="todoRoot">        <h1>할일</h1>        <TodoInput dispatch={dispatch} />        <TodoList todoList={state.todoList} dispatch={dispatch} />      </div>    )  }  export default React.memo(TodoRoot); |

### TodoRoot.css

|  |
| --- |
| .todoRoot { border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px; }  .todoRoot h1 { margin: 0  0 20px 0; }  .todoRoot div > input { padding: 6px; width: 280px; }  .todoRoot div button { margin-left: 5px; padding: 3px 10px; }  .todoRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }  .todoRoot td { border: 1px solid gray; padding: 6px; }  .todoRoot thead { background-color: #eee; text-align: center; }  .todoRoot td:nth-child(1) { text-align: center; width: 30px; }  .todoRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }  .todoRoot span { float: right; font-weight: bold; cursor: pointer; }  .todoRoot tr.done { color: #bbb; text-decoration: line-through; } |

### App.tsx

|  |
| --- |
| import React from 'react';  import TodoRoot from './TodoRoot';  function App() {    console.log('App');    return (      <TodoRoot />    );  }  export default React.memo(App); |

## 실행

todo 항목 한 개의 체크 박스를 계속 클릭해 보자.

처음 클릭했을 때는, 모든 컴포넌트들이 렌더링 되지만,

두 번째 부터는 상태가 변한 TodoItem 만 렌더링 되고, 상태가 변하지 않은 다른 TodoItem 들은 렌더링 되지 않는 것을 확인할 수 있다.

Graphical user interface

Description automatically generated with low confidence

TodoItem 3 만 렌더링 되고, TodoItem 1, TodoItem 2는 렌더링 되고 있지 않다.

### 랜더링 최적화가 잘 되는 이유

<TodoItem key={todo.id} todo={todo} dispatch={dispatch} />

TodoItem 컴포넌트에 props로 전달되는 dispatch 함수는 언제나 같은 함수이다.

따라서 todo props 값이 전과 동일한 TodoItem 컴포넌트는 다시 렌더링 되지 않는다.

## 결론

여러 컴포넌트들로 나누어 구현되는 상태 관리는 useReducer 훅으로 구현하는 것이 바람직하다.

# useReducer 코드 개선

## 개요

### 불편한 점

useReducer 액션 객체를 생성하여 dispatch 하는 코드 구현이 조금 번거롭다.

|  |
| --- |
| const onChange = () => dispatch({ type: "toggleTodo", payload: { id: todo.id } });    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        dispatch({ type: "deleteTodo", payload: { id: todo.id } });    } |

|  |
| --- |
| const onClick = () => {      dispatch({ type: "addTodo", payload: { title } });      setTitle('');    } |

### 개선

|  |
| --- |
| export const addTodo = (title: string): Action => ({ type: "addTodo", payload: { title }})  export const toggleTodo = (id: number): Action => ({ type: "toggleTodo", payload: { id }})  export const deleteTodo = (id: number): Action => ({ type: "deleteTodo", payload: { id }}) |

위와 같은 함수를 todoReducer.ts에 미리 구현하면, 컴포넌트들에서 액션 객체를 생성할 때 편리하다.

위 함수들은 화살표 함수 문법으로 구현되었다.

화살표 함수에서 객체를 생성해서 리턴할 때는, 그 객체를 ( ) 괄호로 묶어 주어야 한다.

그렇지 않으면, 함수 본문을 묶는 { } 괄호와 객체를 생성하는 { } 괄호를 구분할 수 없기 때문이다.

addTodo 함수는 { type: "addTodo", payload: { title }} 객체를 생성해서 리턴한다.

toggleTodo 함수는 { type: "toggleTodo", payload: { id }} 객체를 생성해서 리턴한다.

deleteTodo 함수는 { type: "deleteTodo", payload: { id }} 객체를 생성해서 리턴한다.

컴포넌트에서 액션 객체를 생성하여 dispatch 하는 코드를 다음과 같이 구현할 수 있다.

|  |
| --- |
| const onChange = () => dispatch(toggleTodo(todo.id));    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        dispatch(deleteTodo(todo.id));    } |

|  |
| --- |
| const onClick = () => {      dispatch(addTodo(title));      setTitle('');    } |

### 구조 분해 할당 문법을 이용한 파라미터 변수 선언

|  |
| --- |
| export function reducer(state: State, action: Action) {      switch (action.type) {          case "addTodo":              return { todoList: [...state.todoList,                 { id: ++lastId, title: action.payload.title, done: false } ] };          case "deleteTodo":              return { todoList: state.todoList.filter(                  todo => todo.id !== action.payload.id ) };          case "toggleTodo":              return { todoList: state.todoList.map(                  todo => todo.id !== action.payload.id ?                          todo :                          {...todo, done: !todo.done}) };          default:              throw new Error("unknown action type");      }  } |

구조 분해 할당 문법을 이용하면 아래와 같이 간결하게 구현할 수 있다.

++lastId 부분은 바람직하지 않다. 순수 함수는 어떤 변수의 값도 변경하면 안된다.  
reducer 함수는 순수 함수 형태로 구현해야 한다.

|  |
| --- |
| export function reducer({ todoList }: State, { type, payload }: Action) {      switch (type) {          case "addTodo":              return { todoList: [...todoList,                 { id: todoList[todoList.length-1].id + 1,  title: payload.title, done: false } ] };          case "deleteTodo":              return { todoList: todoList.filter(                  todo => todo.id !== payload.id ) };          case "toggleTodo":              return { todoList: todoList.map(                  todo => todo.id !== payload.id ?                          todo :                          {...todo, done: !todo.done}) };          default:              throw new Error("unknown action type");      }  } |

## 수정

todoReducer.ts

|  |
| --- |
| export type Todo = {      id: number,      title: string,      done: boolean  }  export type State = {      todoList: Todo[]  }  export const initialState = { todoList: [      {id: 1, title: '프론트엔드 과제', done: false},      {id: 2, title: '백엔드 과제', done: true},      {id: 3, title: '웹프 시험공부', done: false}    ] }  type Action =    { type: "addTodo", payload: { title: string } } |    { type: "toggleTodo", payload: { id: number } } |    { type: "deleteTodo", payload: { id: number } }  export const addTodo = (title: string): Action => ({ type: "addTodo", payload: { title }})  export const toggleTodo = (id: number): Action => ({ type: "toggleTodo", payload: { id }})  export const deleteTodo = (id: number): Action => ({ type: "deleteTodo", payload: { id }})  export type DispatchFunc = (action: Action) => void;  export function reducer({ todoList }: State, { type, payload }: Action) {      switch (type) {          case "addTodo":              return { todoList: [...todoList,                 { id: todoList[todoList.length-1].id + 1,  title: payload.title, done: false } ] };          case "deleteTodo":              return { todoList: todoList.filter(                  todo => todo.id !== payload.id ) };          case "toggleTodo":              return { todoList: todoList.map(                  todo => todo.id !== payload.id ?                          todo :                          {...todo, done: !todo.done}) };          default:              throw new Error("unknown action type");      }  } |

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type { Todo, DispatchFunc } from './todoReducer'  import { toggleTodo, deleteTodo } from './todoReducer'  type Props = {    todo: Todo,    dispatch: DispatchFunc  }  function TodoItem({ todo, dispatch }: Props) {    console.log('TodoItem', todo.id);    const onChange = () => dispatch(toggleTodo(todo.id));    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        dispatch(deleteTodo(todo.id));    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(TodoItem); |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type { DispatchFunc } from './todoReducer'  import { addTodo } from './todoReducer'  type Props = {    dispatch : DispatchFunc  }  function TodoInput({ dispatch }: Props) {    console.log('TodoInput');    const [title, setTitle] = useState<string>('');    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      dispatch(addTodo(title));      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default React.memo(TodoInput) |

# useCallback 훅

## 개요

### App.tsx

|  |
| --- |
| import React, { useState } from 'react';  function App() {    const [count, setCount] = useState<number>(0);    const increase = () => setCount(count + 1);    return (      <div>        <h1>{ count }</h1>        <button onClick={increase}>increase</button>      </div>    );  }  export default App; |

App 컴포넌트가 랜더링 될 때 마다, App 함수가 호출된다.

App 함수가 호출될 때 마다 연두색으로 칠한 inner 함수가 다시 생성된다.

연두색 inner 함수 본문의 count는 outer 함수에 선언된 지역 변수이다.

이 count 값이 inner 함수 본문에 캡쳐되어 포함된다.

즉 연두색 inner 함수가 생성될 때 마다, inner 함수 본문에, 현재 count 값이 캡쳐되어 포함된다.

버튼이 클릭될 때 마다 count 값이 변하기 때문에,

그 새 count 값을 캡쳐하여 연두색 inner 함수가 새로 만들어질 수 밖에 없다.

() => setCount(count + 1)

외부의 count 값을 캡쳐하는 연두색 inner 함수는 순수한 함수가 아니다.

이 count 값이 변경될 때 마다 연두색 inner 함수가 새로 만들어져야 한다.

다음과 같이 구현할 수도 있다.

|  |
| --- |
| import React, { useState } from 'react';  function App() {    const [count, setCount] = useState<number>(0);    const increase = () => setCount(count => count + 1);    return (      <div>        <h1>{ count }</h1>        <button onClick={increase}>increase</button>      </div>    );  }  export default App; |

() => setCount(count => count + 1)

이 inner 함수는 outter 함수의 count 값을 캡쳐하지 않는다.

이 inner 함수 본문에서 setCount 함수의 파라미터값은 count => count + 1 함수이다.

이 함수는 setCount 함수 내부에서 호출된다.

이 함수가 setCount 함수 내부에서 호출될 때, 현재 count 값이 count 파라미터 변수에 전달된다.

count은 App 함수의 지역 변수이다.

count는 count => count + 1 함수의 파라미터 변수이다.

() => setCount(count => count + 1)

이 연두색 inner 함수는 count 값을 캡쳐하지 않는다.

따라서 이 연두색 inner 함수는 count 값이 변경될 때 마다 다시 만들 필요 없다.

그렇지만 위와 같이 평범한 inner 함수로 구현하면

App 함수가 호출될 때 마다 연두색 inner 함수가 매번 새로 만들어진다.

이렇게 inner 함수가 불필요하게 새로 만들어지는 것을 막으려면 useCallback 훅을 사용한다.

## useCallback 훅

|  |
| --- |
| import React, { useState, useCallback } from 'react';  function App() {    const [count, setCount] = useState<number>(0);    const increase = useCallback(() => setCount(count => count + 1), []);    return (      <div>        <h1>{ count }</h1>        <button onClick={increase}>increase</button>      </div>    );  }  export default App; |

매번 다시 만들 필요 없는 inner 함수를 다음과 같이 생성하자.

useCallback(inner 함수, []);

위와 같이 생성하면 inner 함수는 매번 새로 만들어지지 않고,

위 코드가 처음 호출될 때 inner 함수를 만들고, 그 inner 함수를 계속 사용하게 된다.

|  |
| --- |
| 참고: 지금까지 설명이 잘 이해되지 않는다면, 아래 내용을 읽으면 더 헷갈릴 테니 그냥 무시하라.  사실 useCallback 훅을 사용해도 inner 함수는 언제나 새로 만들어진다.  useCallback 훅은 함수이고, 이 함수의 첫째 파라미터가 inner 함수이다.  useCallback 함수가 호출될 때 마다 첫째 파라미터 inner 함수가 생성되어서 파라미터로 전달된다.  그런데 위 코드에서 useCallback 훅은  처음 호출될 때 생성되어 파라미터로 전달된 inner 함수만 보관하고,  그 다음 호출될 때 생성되어 파라미터로 전달된 inner 함수들은 무시한다.  그리고 언제나 처음 전달된 inner 함수만 리턴한다.  과정이야 어쨌든 위 코드의 useCallback 훅이 처음 호출될 때 생성된 연두색 inner 함수만  계속 사용하게 된다. |

# useState + useCallback 렌더링 최적화

### types.ts

|  |
| --- |
| export type Todo = {    id: number,    title: string,    done: boolean  };  export type AddTodoFunc = (text: string) => void;  export type ToogleTodoFunc = (id: number) => void;  export type DeleteTodoFunc = (id: number) => void; |

전과 동일함

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type \* as types from './types'  type Props = {    todo: types.Todo,    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoItem({ todo, toggleTodo, deleteTodo }: Props) {    console.log("TodoItem", todo.id)    const onChange = () => toggleTodo(todo.id)    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id)    }    return (      <tr className={ todo.done ? "done" : "" }>        <td>{ todo.id }</td>        <td>          <input type='checkbox' checked={ todo.done } onChange={onChange} />          { todo.title }          <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(TodoItem); |

전과 동일함

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import type \* as types from './types'  import TodoItem from './TodoItem'  type Props = {    todoList: types.Todo[],    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoList({ todoList, toggleTodo, deleteTodo }: Props) {    console.log("TodoList")    const trlist = todoList.map(todo =>      <TodoItem key={todo.id} todo={todo} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />);    return (      <table>        <thead>          <tr><td>id</td><td>할일</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default React.memo(TodoList) |

전과 동일함

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type \* as types from './types'  type Props = {    addTodo : types.AddTodoFunc  }  function TodoInput({ addTodo }: Props) {    console.log('TodoInput');    const [title, setTitle] = useState<string>('');    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => { addTodo(title); setTitle(''); }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default React.memo(TodoInput) |

전과 동일함

### TodoRoot.tsx

|  |
| --- |
| import React from 'react'  import './TodoRoot.css'  import type \* as types from './types'  import TodoInput from './TodoInput';  import TodoList from './TodoList';  type Props = {    todoList: types.Todo[],    addTodo: types.AddTodoFunc,    toggleTodo: types.ToogleTodoFunc,    deleteTodo: types.DeleteTodoFunc  }  function TodoRoot({ todoList, addTodo, toggleTodo, deleteTodo }: Props) {    console.log("TodoRoot")    return (      <div className="todoRoot">        <h1>할일</h1>        <TodoInput addTodo={addTodo} />        <TodoList todoList={todoList} toggleTodo={toggleTodo} deleteTodo={deleteTodo} />      </div>    )  }  export default React.memo(TodoRoot); |

전과 동일함

### TodoRoot.css

|  |
| --- |
| .todoRoot { border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px; }  .todoRoot h1 { margin: 0  0 20px 0; }  .todoRoot div > input { padding: 6px; width: 280px; }  .todoRoot div button { margin-left: 5px; padding: 3px 10px; }  .todoRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }  .todoRoot td { border: 1px solid gray; padding: 6px; }  .todoRoot thead { background-color: #eee; text-align: center; }  .todoRoot td:nth-child(1) { text-align: center; width: 30px; }  .todoRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }  .todoRoot span { float: right; font-weight: bold; cursor: pointer; }  .todoRoot tr.done { color: #bbb; text-decoration: line-through; } |

전과 동일

### TodoContainer.tsx

|  |
| --- |
| import React, { useState, useCallback } from 'react'  import type \* as types from './types'  import TodoRoot from './TodoRoot';  const data : types.Todo[] = [    {id: 1, title: '프론트엔드 과제', done: false},    {id: 2, title: '백엔드 과제', done: true},    {id: 3, title: '웹프 시험공부', done: false}  ];  function TodoContainer() {    console.log("TodoContainer")    const [todoList, setTodoList] = useState<types.Todo[]>(data);    const addTodo : types.AddTodoFunc = useCallback((title: string) => {      setTodoList(todoList => {        const lastIndex = todoList.length - 1;        const lastId = todoList[lastIndex].id;        const todo = {id: lastId + 1, title: title, done: false};        return [...todoList, todo]      });    }, []);    const toggleTodo : types.ToogleTodoFunc = useCallback((id: number) => {      setTodoList(todoLsit =>        todoList.map(          todo => todo.id !== id ? todo : {...todo, done: !todo.done }        )      )    }, []);    const deleteTodo : types.DeleteTodoFunc = useCallback((id: number) => {      setTodoList(todoList => todoList.filter(todo => todo.id !== id))    }, []);    return <TodoRoot todoList={todoList}                     addTodo={addTodo}                     toggleTodo={toggleTodo}                     deleteTodo={deleteTodo} />;  }  export default React.memo(TodoContainer) |

### App.tsx

|  |
| --- |
| import React from 'react';  import TodoContainer from './TodoContainer';  function App() {    console.log("App")    return (      <TodoContainer />    );  }  export default React.memo(App); |

# 예제

## 학생 목록 - useReducer

### studentReducer.ts

|  |
| --- |
| export type Student = {      id: number,      studentNo: string,      name: string  }  export type State = {      students: Student[]  }  export const initialState = { students: [      {id: 1, studentNo: '201514191', name: '홍길동'},      {id: 2, studentNo: '201514192', name: '임꺽정'},      {id: 3, studentNo: '201514193', name: '전우치'},  ] }  export type Action =    { type: "addStudent", payload: { student: Student } } |    { type: "deleteStudent", payload: { id: number } }  export const addStudent = (student: Student): Action => ({ type: "addStudent", payload: { student } })  export const deleteStudent = (id: number): Action => ({ type: "deleteStudent", payload: { id } })  export type DispatchFunc = (action: Action) => void;  export type ReducerFunc = (state: State, action: Action) => State;  export const reducer: ReducerFunc = (state: State, action: Action) => {      const students = state.students;      switch (action.type) {          case "addStudent":              return { students: [...students,                 { ...action.payload.student, id: students[students.length-1].id + 1 } ] };          case "deleteStudent":              return { students: students.filter(                  student => student.id !== action.payload.id ) };          default:              throw new Error("unknown action type");      }  } |

### StudentItem.tsx

|  |
| --- |
| import React from 'react'  import type { Student, DispatchFunc } from './studentReducer'  import { deleteStudent } from './studentReducer'  type Props = {    student: Student,    dispatch: DispatchFunc  }  function StudentItem({ student, dispatch }: Props) {    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        dispatch(deleteStudent(student.id));    }    return (      <tr>        <td>{ student.id }</td>        <td>{ student.studentNo }</td>        <td>{ student.name }            <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(StudentItem) |

### StudentList.tsx

|  |
| --- |
| import React from 'react'  import StudentItem from './StudentItem'  import type { Student, DispatchFunc } from './studentReducer'  type Props = {    students: Student[],    dispatch: DispatchFunc  }  function StudentList({ students, dispatch }: Props) {    const trlist = students.map(student =>      <StudentItem key={student.id} student={student} dispatch={dispatch} />);    return (      <table>        <thead>          <tr><td>id</td><td>학번</td><td>이름</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default React.memo(StudentList) |

### StudentInput.tsx

|  |
| --- |
| import React, { useState } from 'react'  import type { DispatchFunc, Student } from './studentReducer'  type Props = {    dispatch : DispatchFunc  }  function StudentInput({ dispatch }: Props) {    const [studentNo, setStudentNo] = useState<string>('');    const [name, setName] = useState<string>('');    const onClick = () => {      const student: Student = {id: 0, studentNo, name};      dispatch({ type: "addStudent", payload: { student } });      setStudentNo('');      setName('');    }    return (      <div className="input">        <div>학번: <input type='text' value={studentNo}  onChange={e => setStudentNo(e.target.value)} /></div>        <div>이름: <input type='text' value={name}  onChange={e => setName(e.target.value)} /></div>        <div><button onClick={onClick}>추가</button></div>      </div>    )  }  export default React.memo(StudentInput) |

### StudentRoot.tsx

|  |
| --- |
| import React, { useReducer } from 'react'  import './StudentRoot.css'  import StudentInput from './StudentInput';  import StudentList from './StudentList';  import { reducer, initialState } from './studentReducer'  function StudentRoot() {    const [state, dispatch] = useReducer(reducer, initialState)    return (      <div className="studentRoot">        <h1>학생 목록</h1>        <StudentInput dispatch={dispatch} />        <StudentList students={state.students} dispatch={dispatch} />      </div>    )  }  export default React.memo(StudentRoot) |

### StudentRoot.css

|  |
| --- |
| .studentRoot { border: 1px solid gray; width: 400px; margin: 10px auto;      box-shadow: 5px 5px 5px #DDD; padding: 30px; }    .studentRoot h1 { margin: 0  0 20px 0; }    .studentRoot div > input { padding: 6px; width: 280px; }    .studentRoot div button { margin-left: 5px; padding: 3px 10px; }    .studentRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }    .studentRoot td { border: 1px solid gray; padding: 6px; }    .studentRoot thead { background-color: #eee; text-align: center; }    .studentRoot td:nth-child(1) { text-align: center; width: 30px; }    .studentRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }    .studentRoot span { float: right; font-weight: bold; cursor: pointer; }    .studentRoot tr.done { color: #bbb; text-decoration: line-through; }    .studentRoot div.input div { margin: 5px; }    .studentRoot div.input button { margin-left: 40px; padding: 0.3em 1.5em; } |

## 제품 목록 - useState

### types.ts

|  |
| --- |
| export type Product = {      id: number,      title: string,      price: number    };      export type AddProductFunc = (title: string, price: number) => void;    export type DeleteProductFunc = (id: number) => void; |

### ProductItem.tsx

|  |
| --- |
| import React from 'react'  import type { Product, DeleteProductFunc } from './types'  type Props = {    product: Product,    deleteProduct: DeleteProductFunc  }  function ProductItem({ product, deleteProduct }: Props) {    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))      deleteProduct(product.id);    }    return (      <tr>        <td>{ product.id }</td>        <td>{ product.title }</td>        <td>{ product.price }            <span onClick={onClick}>x</span>        </td>      </tr>    )  }  export default React.memo(ProductItem) |

### ProductList.tsx

|  |
| --- |
| import React from 'react'  import ProductItem from './ProductItem'  import type { Product, DeleteProductFunc } from './types'  type Props = {    products: Product[],    deleteProduct: DeleteProductFunc  }  function ProductList({ products, deleteProduct }: Props) {    const trlist = products.map(product =>      <ProductItem key={product.id} product={product} deleteProduct={deleteProduct} />);    return (      <table>        <thead>          <tr><td>id</td><td>제픔</td><td>가격</td></tr>        </thead>        <tbody>          {trlist}        </tbody>      </table>    )  }  export default React.memo(ProductList) |

### ProductInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type { AddProductFunc } from './types'  type Props = {    addProduct : AddProductFunc  }  function ProductInput({ addProduct }: Props) {    const [title, setTitle] = useState<string>('');    const [price, setPrice] = useState<number>(0);    const onChange1 = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onChange2 = (e: ChangeEvent<HTMLInputElement>) => setPrice(Number(e.target.value));    const onClick = () => {      addProduct(title, price);      setTitle('');      setPrice(0);    }    return (      <div className="input">        <div>제품: <input type='text' value={title} onChange={onChange1} /></div>        <div>가격: <input type='text' value={price} onChange={onChange2} /></div>        <div><button onClick={onClick}>추가</button></div>      </div>    )  }  export default React.memo(ProductInput) |

### ProductRoot.tsx

|  |
| --- |
| import React from 'react'  import './ProductRoot.css'  import type \* as types from './types'  import ProductInput from './ProductInput';  import ProductList from './ProductList';  type Props = {    products: types.Product[],    addProduct: types.AddProductFunc,    deleteProduct: types.DeleteProductFunc  }  function ProductRoot({ products, addProduct, deleteProduct }: Props) {    return (      <div className="productRoot">        <h1>제품 목록</h1>        <ProductInput addProduct={addProduct} />        <ProductList products={products} deleteProduct={deleteProduct} />      </div>    )  }  export default React.memo(ProductRoot) |

### ProductRoot.css

|  |
| --- |
| .productRoot { border: 1px solid gray; width: 400px; margin: 10px auto;      box-shadow: 5px 5px 5px #DDD; padding: 30px; }    .productRoot h1 { margin: 0  0 20px 0; }    .productRoot div > input { padding: 6px; width: 280px; }    .productRoot div button { margin-left: 5px; padding: 3px 10px; }    .productRoot table { border-collapse: collapse; margin-top: 20px; width: 100%; }    .productRoot td { border: 1px solid gray; padding: 6px; }    .productRoot thead { background-color: #eee; text-align: center; }    .productRoot td:nth-child(1) { text-align: center; width: 30px; }    .productRoot input[type=checkbox] { margin-right: 10px; accent-color: gray; }    .productRoot span { float: right; font-weight: bold; cursor: pointer; }    .productRoot tr.done { color: #bbb; text-decoration: line-through; }    .productRoot div.input div { margin: 5px; }    .productRoot div.input button { margin-left: 40px; padding: 0.3em 1.5em; } |

### ProductContainer.tsx

|  |
| --- |
| import React, { useState } from 'react'  import type \* as types from './types'  import ProductRoot from './ProductRoot';  const data : types.Product[] = [    {id: 1, title: '우유', price: 2000},    {id: 2, title: '콜라', price: 1800},    {id: 3, title: '맥주', price: 4000},  ];  function ProductContainer() {    console.log("ProductContainer")    const [products, setProducts] = useState<types.Product[]>(data);    const addProduct : types.AddProductFunc = (title: string, price: number) => {      const product = {id: products[products.length-1].id + 1, title, price};      setProducts([...products, product]);    }    const deleteProduct : types.DeleteProductFunc = (id: number) => {      setProducts( products.filter(product => product.id !== id) );    }      return <ProductRoot products={products}                     addProduct={addProduct}                     deleteProduct={deleteProduct} />;  }  export default React.memo(ProductContainer) |

### App.tsx

|  |
| --- |
| import React from 'react';  import StudentRoot from './StudentRoot';  import ProductContainer from './ProductContainer';  function App() {    return (      <div>        <StudentRoot />        <br />        <ProductContainer />      </div>    );  }  export default React.memo(App) |