axios로 todo앱 구현

**학습목표**

json-server + axios + useReducer 조합으로 todo앱 구현하기

**목차**

[1. todo5 2](#_Toc150942943)

[1) 개요 2](#_Toc150942944)

[2) 프로젝트 생성 2](#_Toc150942945)

[3) 구현 3](#_Toc150942946)

[2. 랜더링 최적화 9](#_Toc150942947)

[1) 개요 9](#_Toc150942948)

[2) 구현 9](#_Toc150942949)

[3. 항상 reload 하기 13](#_Toc150942950)

[1) 개요 13](#_Toc150942951)

[2) 구현 13](#_Toc150942952)

[4. 항상 reload & 렌더링 최적화 16](#_Toc150942953)

[1) 개요 16](#_Toc150942954)

[2) 구현 17](#_Toc150942955)

# todo5

## 개요

useReducer + axios 조합으로 todo 앱을 구현하자.

## 프로젝트 생성

|  |
| --- |
| npx create-react-app todo5 --template typescript |

### axios 설치

|  |
| --- |
| npm install axios |

## 구현

### todoReducer.ts

|  |
| --- |
| export type Todo = {    id: number,    title: string,    done: boolean  }  export type State = {    todos: Todo[]  }  export const initialState: State = { todos: [] }  export type Action =  { type: "setTodos", payload: { todos: Todo[] }} |  { type: "addTodo", payload: { todo: Todo } } |  { type: "updateTodo", payload: { todo: Todo } } |  { 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) => {  let newTodos: Todo[];  switch (action.type) {    case "setTodos":      newTodos = action.payload.todos;      break;    case "addTodo":      newTodos = [ ...state.todos, action.payload.todo ];      break;    case "deleteTodo":      newTodos = state.todos.filter(todo => todo.id !== action.payload.id );      break;    case "updateTodo":      newTodos = state.todos.map(        todo => todo.id !== action.payload.todo.id ? todo : action.payload.todo);      break;    default:      throw new Error("unknown action type");  }  return { todos: newTodos };  } |

앱의 상태는 todos 목록이다.

앱의 메모리에는 todos 목록을 변경하는 기능을 reducer로 구현하였다.

### todoService.ts

|  |
| --- |
| import type { Todo, DispatchFunc } from './todoReducer';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export async function reloadTodos(dispatch: DispatchFunc) {    try {      const response = await axios.get("/todos");      const todos: Todo[] = response.data;      dispatch({ type: 'setTodos', payload: { todos } });    } catch (error) {      if (error instanceof Error) alert('조회 에러: ' + error.message);      else alert('조회 에러: ' + error);    }  }  export async function updateTodo(todo: Todo, dispatch: DispatchFunc) {    try {      await axios.put("/todos/" + todo.id, todo);      dispatch({ type: 'updateTodo', payload: { todo: todo } });    } catch (error) {      if (error instanceof Error) alert('저장 에러: ' + error.message);      else alert('저장 에러: ' + error);    }  }  export async function deleteTodo(id: number, dispatch: DispatchFunc) {    try {      await axios.delete("/todos/" + id);      dispatch({ type: 'deleteTodo', payload: { id } });    } catch (error) {      if (error instanceof Error) alert('삭제 에러: ' + error.message);      else alert('삭제 에러: ' + error);    }  }  export async function addTodo(todo: Todo, dispatch: DispatchFunc) {    try {      const response = await axios.post("/todos/", todo);      dispatch({ type: 'addTodo', payload: { todo: response.data } });    } catch (error) {      if (error instanceof Error) alert('저장 에러: ' + error.message);      else alert('저장 에러: ' + error);    }  } |

reloadTodos 함수

백엔드 API를 호출하여 todo 객체 목록을 조회한다.

dispatch 함수를 호출하여, 조회된 todo 객체 목록을 앱의 상태에 저장한다.

updateTodo

백엔드 API를 호출하여 todo 객체를 저장(update) 한다.

dispatch 함수를 호출하여, 앱의 상태 todo 목록에서도 그 todo 객체를 update 한다.

deleteTodo

백엔드 API를 호출하여 todo 객체를 삭제 한다.

dispatch 함수를 호출하여, 앱의 상태 todo 목록에서도 그 todo 객체를 삭제 한다.

addTodo

백엔드 API를 호출하여 todo 객체를 저장(insert) 한다.

dispatch 함수를 호출하여, 앱의 상태 todo 목록에서도 그 todo 객체를 추가 한다.

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type { Todo, DispatchFunc } from './todoReducer'  import { updateTodo, deleteTodo } from './todoService'  type Props = {    todo: Todo,    dispatch: DispatchFunc  }  function TodoItem({ todo, dispatch }: Props) {    const onChange = () => updateTodo({...todo, done: !todo.done}, dispatch);    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id, dispatch);    }    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 TodoItem; |

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import TodoItem from './TodoItem'  import type { Todo, DispatchFunc } from './todoReducer'  type Props = {    todos: Todo[],    dispatch: DispatchFunc  }  function TodoList({ todos, dispatch }: Props) {    const trlist = todos.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 TodoList |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type { DispatchFunc } from './todoReducer'  import { addTodo } from './todoService'  type Props = {    dispatch : DispatchFunc  }  function TodoInput({ dispatch }: Props) {    const [title, setTitle] = useState<string>("");    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      addTodo({id: 0, title: title, done: false}, dispatch);      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  }  export default TodoInput |

### TodoRoot.tsx

|  |
| --- |
| import React, { useReducer, useEffect } from 'react'  import styled from 'styled-components';  import TodoInput from './TodoInput';  import TodoList from './TodoList';  import { reducer, initialState } from './todoReducer';  import { reloadTodos } from './todoService';  const Div = styled.div`    border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px;    & h1 { margin: 0  0 20px 0; }    & div > input { padding: 6px; width: 280px; }    & button { padding: 3px 10px; }    & table { border-collapse: collapse; margin: 20px 0; width: 100%; }    & td { border: 1px solid gray; padding: 6px; }    & thead { background-color: #eee; text-align: center; }    & td:nth-child(1) { text-align: center; width: 30px; }    & input { margin-right: 5px; }    & input[type=checkbox] { margin-right: 10px; accent-color: gray; }    & span { float: right; font-weight: bold; cursor: pointer; }    & tr.done { color: #bbb; text-decoration: line-through; }  `;  function TodoRoot() {    const [state, dispatch] = useReducer(reducer, initialState)    useEffect(() => { reloadTodos(dispatch); }, []);    return (      <Div>        <h1>할일</h1>        <TodoInput dispatch={dispatch} />        <TodoList todos={state.todos} dispatch={dispatch} />        <button onClick={() => reloadTodos(dispatch)}>새로고침</button>      </Div>    )  }  export default TodoRoot; |

useEffect(() => { reloadTodos(dispatch); }, []);

TodoRoot 컴포넌트가 mount될 때, todo 목록을 reload 한다.

### App.tsx

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

### todo-db.json

|  |
| --- |
| {    "todos": [      {        "id": 1,        "title": "프론트엔드 과제",        "done": false      },      {        "id": 2,        "title": "백엔드 과제",        "done": true      },      {        "id": 3,        "title": "웹프 시험공부",        "done": false      }    ]  } |

### 실행

터미널 창 #1 - 백엔드 실행

|  |
| --- |
| json-server --watch todo-db.json |

위 명령은 todo-db.json 파일이 있는 src 디렉토리에서 실행해야 한다.

터미널 창 #2 - 프론트엔드 플젝 실행

|  |
| --- |
| npm run start |

# 랜더링 최적화

여기서부터 2023년 2학기 학습목표에서 제외

## 개요

React.memo 고차 함수를 사용하여 렌더링 최적화를 구현하자.

const 새컴포넌트함수 = React.memo(컴포넌트함수)

export default 새컴포넌트함수

## 구현

### TodoItem.tsx

|  |
| --- |
| import React from 'react'  import type { Todo, DispatchFunc } from './todoReducer'  import { updateTodo, deleteTodo } from './todoService'  type Props = {    todo: Todo,    dispatch: DispatchFunc  }  const TodoItem = React.memo(({ todo, dispatch }: Props) => {    console.log('TodoItem', todo.id);    const onChange = () => updateTodo({...todo, done: !todo.done}, dispatch);    const onClick = () => {      if (window.confirm('삭제하시겠습니까?'))        deleteTodo(todo.id, dispatch);    }    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 TodoItem; |

### TodoList.tsx

|  |
| --- |
| import React from 'react'  import TodoItem from './TodoItem'  import type { Todo, DispatchFunc } from './todoReducer'  type Props = {    todos: Todo[],    dispatch: DispatchFunc  }  const TodoList = React.memo(({ todos, dispatch }: Props) => {    console.log('TodoList');    const trlist = todos.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 TodoList |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type { DispatchFunc } from './todoReducer'  import { addTodo } from './todoService'  type Props = {    dispatch : DispatchFunc  }  const TodoInput = React.memo(({ dispatch }: Props) => {    console.log('TodoInput');    const [title, setTitle] = useState<string>("");    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      addTodo({id: 0, title: title, done: false}, dispatch);      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  })  export default TodoInput |

### TodoRoot.tsx

|  |
| --- |
| import React, { useReducer, useEffect } from 'react'  import styled from 'styled-components';  import TodoInput from './TodoInput';  import TodoList from './TodoList';  import { reducer, initialState } from './todoReducer';  import { reloadTodos } from './todoService';  const Div = styled.div`    border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px;    & h1 { margin: 0  0 20px 0; }    & div > input { padding: 6px; width: 280px; }    & button { padding: 3px 10px; }    & table { border-collapse: collapse; margin: 20px 0; width: 100%; }    & td { border: 1px solid gray; padding: 6px; }    & thead { background-color: #eee; text-align: center; }    & td:nth-child(1) { text-align: center; width: 30px; }    & input { margin-right: 5px; }    & input[type=checkbox] { margin-right: 10px; accent-color: gray; }    & span { float: right; font-weight: bold; cursor: pointer; }    & tr.done { color: #bbb; text-decoration: line-through; }  `;  const TodoRoot = React.memo(() => {    console.log('TodoRoot');    const [state, dispatch] = useReducer(reducer, initialState)    useEffect(() => { reloadTodos(dispatch); }, []);    return (      <Div>        <h1>할일</h1>        <TodoInput dispatch={dispatch} />        <TodoList todos={state.todos} dispatch={dispatch} />        <button onClick={() => reloadTodos(dispatch)}>새로고침</button>      </Div>    )  })  export default TodoRoot; |

### App.tsx

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

App 컴포넌트는 React.memo 함수로 감쌀 필요가 없다.

불필요하게 다시 렌더링 되는 경우가 없기 때문이다.

불필요한 랜더링

부모 컴포넌트가 랜더링 될 때, 자식 컴포넌트들도 전부 다시 랜더링된다.

상태나 props 값이 변경되지 않은 자식 컴포넌트들까지 다시 랜더링 되는 것은 불필요한 랜더링이다.

App 컴포넌트는 부모 컴포넌트가 없기 때문에, 불필요한 랜더링이 발생하지 않는다.

# 항상 reload 하기

## 개요

todo 앱 DB를 여러 사람이 공유한다면, 다른 사람이 데이터를 수정했을 수도 있으니,

백엔드 API를 호출하여 데이터를 수정한 후 reload 해야 한다.

## 구현

### todoReducer.ts

|  |
| --- |
| export type Todo = {    id: number,    title: string,    done: boolean  }  export type State = {    todos: Todo[],    errorMsg: string,  }  export const initialState: State = {    todos: [],    errorMsg: ''  }  export type Action =    { type: "setTodos", payload: { todos: Todo[] }} |    { type: "setErrorMsg", payload: { errorMsg: string } }  export type DispatchFunc = (action: Action) => void;  export type ReducerFunc = (state: State, action: Action) => State;  export const reducer : ReducerFunc = ({todos, errorMsg}: State, action: Action) => {    switch (action.type) {      case "setTodos":        todos = action.payload.todos;        break;      case "setErrorMsg":        errorMsg = action.payload.errorMsg;        break;      default:        throw new Error("unknown action type");    }    return { todos, errorMsg };  } |

백엔드 API를 호출하여 데이터를 수정한 후 reload한 todo 목록을

reducer 상태에 set 하기 위한 "setTodos" 액션만 필요하다.

에러 메시지를 표시하기 위한, errorMsg 상태 변수 추가.

### todoService.ts

|  |
| --- |
| import type { Todo, DispatchFunc } from './todoReducer';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export async function reloadTodos(dispatch: DispatchFunc) {    try {      const response = await axios.get("/todos");      const todos: Todo[] = response.data;      dispatch({ type: 'setTodos', payload: { todos } });    } catch (error) {      const s = '조회 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function updateTodo(todo: Todo, dispatch: DispatchFunc) {    try {      await axios.put("/todos/" + todo.id, todo);      reloadTodos(dispatch);    } catch (error) {      const s = '저장 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function deleteTodo(id: number, dispatch: DispatchFunc) {    try {      await axios.delete("/todos/" + id);      reloadTodos(dispatch);    } catch (error) {      const s = '삭제 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function addTodo(todo: Todo, dispatch: DispatchFunc) {    try {      await axios.post("/todos/", todo);      reloadTodos(dispatch);    } catch (error) {      const s = '저장 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  } |

### TodoRoot.tsx

|  |
| --- |
| import React, { useReducer, useEffect } from 'react'  import styled from 'styled-components';  import TodoInput from './TodoInput';  import TodoList from './TodoList';  import { reducer, initialState } from './todoReducer';  import { reloadTodos } from './todoService';  const Div = styled.div`    border: 1px solid gray; width: 400px; margin: 10px auto;    box-shadow: 5px 5px 5px #DDD; padding: 30px;    & h1 { margin: 0  0 20px 0; }    & div > input { padding: 6px; width: 280px; }    & button { padding: 3px 10px; }    & table { border-collapse: collapse; margin: 20px 0; width: 100%; }    & td { border: 1px solid gray; padding: 6px; }    & thead { background-color: #eee; text-align: center; }    & td:nth-child(1) { text-align: center; width: 30px; }    & input { margin-right: 5px; }    & input[type=checkbox] { margin-right: 10px; accent-color: gray; }    & span { float: right; font-weight: bold; cursor: pointer; }    & tr.done { color: #bbb; text-decoration: line-through; }    & #error { margin-top: 20px; color: red; }  `;  const TodoRoot = React.memo(() => {    console.log('TodoRoot');    const [state, dispatch] = useReducer(reducer, initialState)    useEffect(() => { reloadTodos(dispatch); }, []);    const clearErr = () => dispatch({type: 'setErrorMsg', payload: { errorMsg: "" }});    return (      <Div>        <h1>할일</h1>        <TodoInput dispatch={dispatch} />        <TodoList todos={state.todos} dispatch={dispatch} />        <button onClick={() => reloadTodos(dispatch)}>새로고침</button>        { state.errorMsg && <div id="error">{state.errorMsg}                             <span onClick={clearErr}>x</span></div> }      </Div>    )  })  export default TodoRoot; |

#### (A && B) 표현식의 값

A가 false에 해당하는 값이면, 위 표현식의 값은 A 이다.

A가 true에 해당하는 값이면, 위 표현식의 값은 B 이다.

{ state.errorMsg && <div id="error">{state.errorMsg}

                     <span onClick={clearErr}>x</span></div> }

state.errorMsg의 값이 '' 이면, 이 표현식의 값은 '' 이다.

state.errorMsg의 값이 공백이 아닌 문자열이면, 이 표현식의 값은 <div id="error">...</div> 이다.

### 실행

todoService.ts 파일의 addTodo 함수에서 백엔드 API URL을 틀리게 수정하고 할일을 입력해 보자.

URL이 틀렸으니 버튼을 클릭하면 404 Not Found 에러가 발생할 것이다.

# 항상 reload & 렌더링 최적화

## 개요

백엔드에서 reload할 때마다 todo 목록의 todo 객체들은 모두 새로 생성된다.

렌더링이 필요한지 아닌지 판단할 때, todo 객체의 내용이 달라졌는지 바교하는 것이 아니고,

직전에 렌더링한 todo 객체와 새로 렌더링할 todo 객체의 참조를 비교한다.

따라서 직전에 reload한 todo 목록의 todo 객체들과 내용이 같더라도,

모두 새로 만들어진 새 todo 객체들이기 때문에, 모두 다시 렌더링 된다.

불필요한 렌더링을 막으려면, 내용이 달라지지 않은 todo 객체들을, 이전 todo 객체로 교체해 주면 된다.

내용이 달라졌는지 아닌지 판단할 때, 객체의 내용 전체를 비교하는 것은 비효율적이기 때문에,

마지막으로 수정된 시각을 정수로 표현한 timestamp 속성을 추가하자.

### 실행

먼저 앞의 구현은 렌더링 최적화가 안되고 있음을 확인하자.

## 구현

### todoReducer.ts

|  |
| --- |
| export type Todo = {    id: number,    title: string,    done: boolean,    timestamp: number  }  export type State = {    todos: Todo[],    errorMsg: string,  }  export const initialState: State = {    todos: [],    errorMsg: ''  }  export type Action =    { type: "setTodos", payload: { todos: Todo[] }} |    { type: "setErrorMsg", payload: { errorMsg: string } }  export type DispatchFunc = (action: Action) => void;  export type ReducerFunc = (state: State, action: Action) => State;  function mergeTodos(oldTodos: Todo[], newTodos: Todo[]): Todo[] {    const map = new Map<number, Todo>();    for (const todo of oldTodos)      map.set(todo.id, todo);    return newTodos.map(todo => {        const oldTodo = map.get(todo.id);        return (oldTodo && oldTodo.timestamp == todo.timestamp) ? oldTodo : todo;    });  }  export const reducer : ReducerFunc = ({todos, errorMsg}: State, action: Action) => {    switch (action.type) {      case "setTodos":        todos = mergeTodos(todos, action.payload.todos);        break;      case "setErrorMsg":        errorMsg = action.payload.errorMsg;        break;      default:        throw new Error("unknown action type");    }    return { todos, errorMsg };  } |

### todoService.ts

|  |
| --- |
| import type { Todo, DispatchFunc } from './todoReducer';  import axios from 'axios';  axios.defaults.baseURL = "http://localhost:3000";  export async function reloadTodos(dispatch: DispatchFunc) {    try {      const response = await axios.get("/todos");      const todos: Todo[] = response.data;      dispatch({ type: 'setTodos', payload: { todos } });    } catch (error) {      const s = '조회 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function updateTodo(todo: Todo, dispatch: DispatchFunc) {    try {      todo.timestamp = Date.now();      await axios.put("/todos/" + todo.id, todo);      reloadTodos(dispatch);    } catch (error) {      const s = '저장 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function deleteTodo(id: number, dispatch: DispatchFunc) {    try {      await axios.delete("/todos/" + id);      reloadTodos(dispatch);    } catch (error) {      const s = '삭제 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  }  export async function addTodo(todo: Todo, dispatch: DispatchFunc) {    try {      await axios.post("/todos/", todo);      reloadTodos(dispatch);    } catch (error) {      const s = '저장 에러: ' + (error instanceof Error ? error.message : error);      dispatch({type: 'setErrorMsg', payload: { errorMsg: s } });    }  } |

### TodoInput.tsx

|  |
| --- |
| import React, { ChangeEvent, useState } from 'react'  import type { DispatchFunc } from './todoReducer'  import { addTodo } from './todoService'  type Props = {    dispatch : DispatchFunc  }  const TodoInput = React.memo(({ dispatch }: Props) => {    console.log('TodoInput');    const [title, setTitle] = useState<string>("");    const onChange = (e: ChangeEvent<HTMLInputElement>) => setTitle(e.target.value);    const onClick = () => {      addTodo({id: 0, title: title, done: false, timestamp: 0}, dispatch);      setTitle('');    }    return (      <div>        <input type='text' value={title} onChange={onChange} />        <button onClick={onClick}>추가</button>      </div>    )  })  export default TodoInput |

### 실행

렌더링 최적화 구현을 확인해보자.