12강

React의 상태관리 Redux 사용법)

&

리덕스 툴킷(Redux Tooliket)

➢ Redux가 필요한 이유

- 문제상황: Props Drilling
  - 깊은 컴포넌트 트리에서 데이터 전달의 어려움
  - 중간 컴포넌트들의 불필요한 props 전달

- 해결책: Redux
  - 중앙 집중식 상태 관리
  - 어느 컴포넌트에서든 직접 접근 가능

➢ Redux란 ?

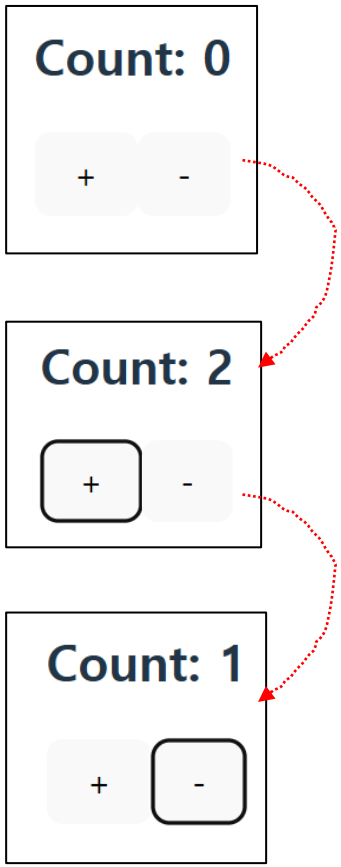
- JavaScript 애플리케이션의 전역 상태 관리 라이브러리 이다.
- "상태(State)를 한곳(Store)에 모아 관리한다."
- Context API의 확장판 이다.

➢ Context API와 Redux 차이점

- Context API는 간단하지만 대규모 상태 관리에는 한계가 있다.
- Redux는 예측 가능한 상태 관리 구조를 제공한다.

항목	Context API	Redux
관리 단위	소규모 상태	대규모 상태
업데이트 구조	단순	엄격 (Reducer 필수)
유지보수성	낮음	높음
사용 예	로그인, 테마	사용자, 장바구니, 주문, UI 상태 등

➢ Redux 데이터 흐름도



사용자 클릭

↓

<Button onClick>
(dispatch)

↓

Action 객체 생성

{ type: 'INCREMENT' }
또는
{ type: 'DECREMENT' }

↓

Dispatch가 Store로 전달

↓

Store
(createStore로 생성)

↓

Reducer(counterReducer)
- 현재 state와 action을 받음
- 새로운 state를 계산하여 반환

↓

새로운 상태(state) 저장
store 내부에서 count 값 업데이트

→

React 컴포넌트(Counter)
- useSelector로 store의 count 구독
- count가 바뀌면 자동 리렌더링

↑

화면(UI)의
"Count: 1"
또는
"Count: 2" 로
업데이트

React Component (Counter.jsx)
↓ dispatch({ type: 'INCREMENT' })
Redux Store (store.js)
↓ counterReducer(state, action)
Reducer (counterReducer.js)
↓ return { count: state.count + 1 }
Store 상태 변경
↓ useSelector → 새로운 state 반영
UI 업데이트 (Count: 1)

➢ Redux의 3가지 핵심 요소

1. Store: 상태 저장소
   - 애플리케이션의 전체 상태를 보관
   - 단 하나만 존재

2. Action: 행동
   - 상태 변경을 요청하는 객체
   - type과 payload 포함

3. Reducer: 상태 변경 함수
   - 현재 상태 + Action -> 새로운 상태
   - 순수 함수로 작성

4. Dispatch: Action을 보내는 함수

➢ Redux 설치

- npm install redux
- npm install react-redux

redux: 상태 관리 핵심

react-redux: React와 연결

➢ 액션(Action)

- 애플리케이션에서 일어나는 사건을 설명하는 객체
- Type 속성을 필수로 가지며, 상태 변경을 트리거(어떤 사건을 촉발시키는)하는 역할을 한다.
- 예) { type: 'INCREMENT' }, { type: 'ADD_TODO', text: 'Learn Redux' }

➢ 리듀서(Reducer)

- 액션을 처리하여 새로운 상태를 변환하는 함수
- 이전 상태와 액션 객체를 인자로 받아 새로운 상태 객체를 반환한다.
- 순수 함수여야 하며, 입력이 같으면 출력도 항상 같아야 하다.
  - 순수 함수(Pure Function)
    - 동일한 인자가 주어졌을 때 항상 동일한 결과를 반환
    - 함수의 실행이 외부 상태에 의존하지 않은
    - 외부 상태를 변경하지 않는 함수

➢ 스토어(Store)

- 애플리케이션의 상태를 담고 있는 객체
- createStore 함수를 사용하여 스토어를 생성한다.
- 스토어는 3가지 메소드를 가진다.
  - getState() : 현재 상태 반환함.
  - dispatch(action) : 상태를 변경하는 액션 보내기
  - Subscribe(listener) : 상태가 변경될 때마다 호출되는 리스너를 등록하기

➢ .js vs .jsx의 차이

확장자	의미	용도	예시
.js	순수 자바스크립트 파일	JSX 문법이 없는 로직, 설정 파일	store.js, reducer.js, utils.js
.jsx	JSX 문법 포함된 파일	React 컴포넌트 (HTML처럼 보이는 코드 있음)	App.jsx, Counter.jsx, Login.jsx

store.js 작성

```
// createStore 최소선은 앞으로는 사라질 예정이니
// 새 방식으로 바꿔 써라"는 경고 표시
// Redux 최신 버전(>= 4.2, 특히 Toolkit 통합 이후)에서는
// createStore()를 더 이상 공식적으로 권장하지 않는다.
import { createStore } from 'redux';
import counterReducer from './counterReducer';

export const store = createStore(counterReducer);
```

counterReducer.js 작성

```
// 상태(state)의 초기값 설정
// - Redux에서는 반드시 초기 상태를 정의해야 함
// - 앱이 시작될 때 state가 undefined인 경우, 이 값이 기본으로 사용됨
const initialState = { count: 0 };


// 리듀서 함수 (Reducer Function)
// - "상태(state)"와 "액션(action)"을 받아서 새로운 상태를 반환하는 순수 함수
// - 기존 state를 직접 변경하지 않고, 항상 새로운 객체를 반환해야 함
export default function counterReducer(state = initialState, action) {
  switch (action.type) {
    // 카운트 증가 액션
    // { type: 'INCREMENT' } 가 전달되면 count를 +1 시킴
    case 'INCREMENT':
      return { count: state.count + 1 };

    // 카운트 감소 액션
    // { type: 'DECREMENT' } 가 전달되면 count를 -1 시킴
    case 'DECREMENT':
      return { count: state.count - 1 };

    // 그 외의 액션 (매칭되지 않는 경우)
    // state를 그대로 반환 (즉, 아무 변화 없음)
    default:
      return state;
  }
}
```

➢ Provider 적용 (main.jsx)

```
import { Provider } from 'react-redux';
import { store } from './store';
import App from './App';

<Provider store={store}>
  <App />
</Provider>
```

➢ Counter.jsx

```
import { useSelector, useDispatch } from 'react-redux';

export default function Counter() {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();

  return (
    <>
      <h2>Count: {count}</h2>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>+</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>-</button>
    </>
  );
}
```

➢ Redux 단점

- 액션 타입, 액션 생성자, 리듀서 등 파일과 코드가 많음
- switch-case 문 사용 → 복잡한 문법
- 상태 업데이트 시 불변성을 수동으로 처리해야 함 → 실수 가능

➢ Redux Toolkit 사용하는 이유

- Redux의 복잡성을 해결하기 위한 공식 도구
- 간단한 설정: configureStore() 하나로 스토어 설정 가능
- 더 적은 코드로 상태 관리 가능
- 현대적인 Redux 사용 패턴
- createSlice()로 액션과 리듀서를 한 번에 생성
- Immer.js 내장 → 불변성 처리 자동

➢ Redux Toolkit 설치

- npm install @reduxjs/toolkit

➢ Redux Toolkit의 핵심 API

API	설명
configureStore()	스토어 생성 및 미들웨어 설정을 간단하게 처리
createSlice()	상태(state), 리듀서(reducer), 액션(action)을 한 번에 생성
createAsyncThunk()	비동기 로직 처리(예: API 호출)
createEntityAdapter()	리스트 상태를 효율적으로 관리

➢ Redux Toolkit 사용 방법

```
createSlice({
  name : 'state이름~',
  initialState : '값',
})
```

```
export default configureStore({
    reducer : {

        이 부분에 등록해야 사용 가능
    }
})
```

➢ Redux store에 state 보관하는 법

```
const user  = createSlice({
  name : 'user',
  initialState : 'kim',
})
```

```
export default configureStore({
    reducer : {

        user : user.reducer
    }
})
```

➢ Redux store에서 state 꺼내는 법

```
const a = useSelector((state)=>{return state })
console.log(a)
```

Redux Toolkit 폴더 구조

```
src/
├── store/
│   ├── store.js
│   └── counterSlice.js
├── components/
│   └── Counter.jsx    카운터 UI 컴포넌트
└── App.jsx            최상위 컴포넌트
```

store.js

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './counterSlice';


export const store = configureStore({
  reducer: {
    counter: counterReducer, // 슬라이스를 등록
  },        Redux store에 state 보관하는 방법
});
```

counterSlice.js

```
import { createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter', <= state 이름 (state 하나를 slice라고 부름)
  initialState: { value: 0 },
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
    reset: (state) => {
      state.value = 0;
    },
  },
});


export const { increment, decrement, reset } = counterSlice.actions;
export default counterSlice.reducer;
```

counter.jsx

```jsx
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement, reset } from '../store/counterSlice';

function Counter() {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>카운터: {count}</h1>
      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
      <button onClick={() => dispatch(reset())}>Reset</button>
    </div>
  );
}

export default Counter;
```

Redux store 가져와줌

Redux store의 state 꺼내는 방법

〈초기 화면〉

〈[+] 버튼 클릭〉

〈[-] 버튼 클릭〉

〈[Reset] 버튼 클릭〉

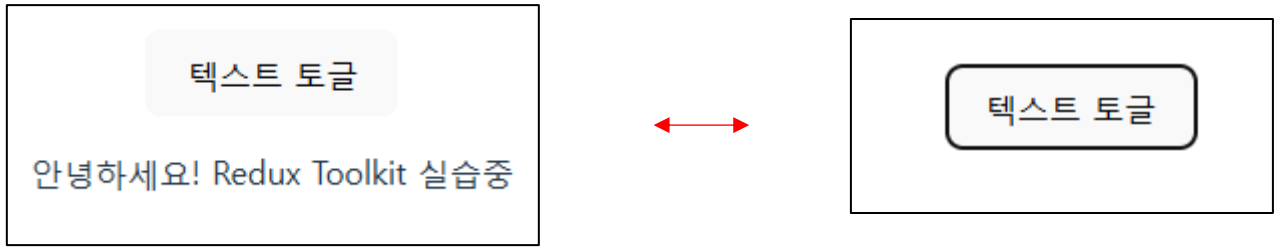
기초 연습 문제

문제1] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 버튼 클릭 시 텍스트를 보이거나 숨기는 기능이 구현되도록 작성하시오.
② toggleSlice 생성, 상태: { show: true }
③ toggleText 액션 구현 → 상태 show 반전
④ 버튼 클릭 시 텍스트 표시 여부 변경
⑤ src 폴더안에 -> storeEx 폴더 생성 -> toggleSlice.js 파일을 생성하여 작성하시오.
⑥ src 폴더안에 -> storeEx 폴더 생성 -> toggleApp.jsx 파일을 생성하여 작성하시오.
⑦ src 폴더안에 -> store 폴더안에 -> store.js에 toggle을 등록하여 작성하시오.
⑧ src 폴더안에 -> App.jsx에 작성한 toggleApp.jsx 를 import하여 실행 하시오.

<출력 결과>

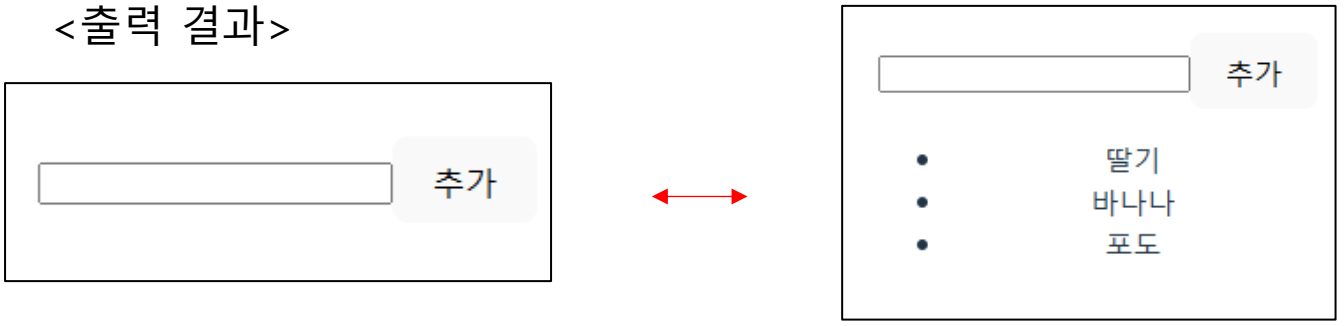


문제2] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 입력값을 리스트에 추가하는 기능이 구현되도록 작성하시오.
② listSlice 생성, 상태: items: []
③ addItem 액션 구현
④ 버튼 클릭 시 입력값을 배열에 추가하고 화면에 렌더링
⑤ src 폴더안에 -> storeEx 폴더 생성 -> listSlice.js 파일을 생성하여 작성하시오.
⑥ src 폴더안에 -> storeEx 폴더 생성 -> listSliceApp.jsx 파일을 생성하여 작성하시오.
⑦ src 폴더안에 -> store 폴더안에 -> store.js에 listSlice 을 등록하여 작성하시오.
⑧ src 폴더안에 -> App.jsx에 작성한 listSliceApp.jsx 를 import하여 실행 하시오.

<출력 결과>

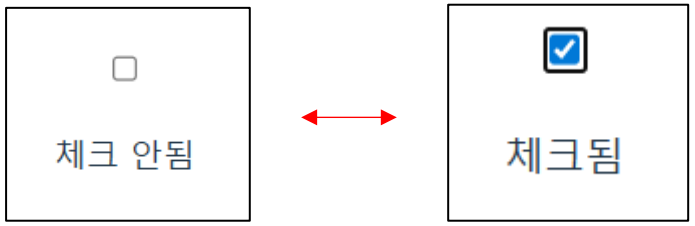


문제3] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 체크박스 선택 여부를 Redux 상태로 관리되도록 작성하시오.
② checkboxSlice 생성, 상태: { checked: false }
③ toggleCheck 액션 구현 → 상태 반전
④ 체크박스 클릭 시 상태 업데이트
⑤ src 폴더안에 -> storeEx 폴더 생성 -> checkboxSlice.js 파일을 생성하여 작성하시오.
⑥ src 폴더안에 -> storeEx 폴더 생성 -> checkboxApp.jsx 파일을 생성하여 작성하시오.
⑦ src 폴더안에 -> store 폴더안에 -> store.js에 checkboxSlice 을 등록하여 작성하시오.
⑧ src 폴더안에 -> App.jsx에 작성한 checkboxApp.jsx 를 import하여 실행 하시오.

<출력 결과>

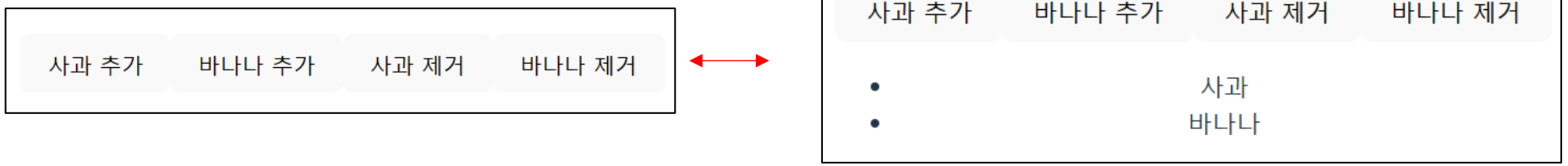


문제4] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 상품을 카트에 추가/제거 되도록 작성하시오.
② cartSlice 생성, 상태: items: []
③ addCartItem, removeCartItem 액션 구현
④ 화면에서 버튼 클릭 시 상태 변경 후 렌더링
⑤ src 폴더안에 -> storeEx 폴더 생성 -> cartSlice.js 파일을 생성하여 작성하시오.
⑥ src 폴더안에 -> storeEx 폴더 생성 -> cartSliceApp.jsx 파일을 생성하여 작성하시오.
⑦ src 폴더안에 -> store 폴더안에 -> store.js에 cartSlice 을 등록하여 작성하시오.
⑧ src 폴더안에 -> App.jsx에 작성한 cartSliceApp.jsx 를 import하여 실행 하시오.

<출력 결과>



문제5] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 리스트 중 선택한 날짜를 Redux 상태로 관리되도록 작성하시오.
② dateSlice 생성, 상태: { selected: '' }
③ selectDate 액션 구현 → 선택된 날짜 업데이트
④ 화면에서 버튼 클릭 시 상태 변경 후 표시
⑤ src 폴더안에 -> storeEx 폴더 생성 -> dateSlice.js 파일을 생성하여 작성하시오.
⑥ src 폴더안에 -> storeEx 폴더 생성 -> dateApp.jsx 파일을 생성하여 작성하시오.
⑦ src 폴더안에 -> store 폴더안에 -> store.js에 dateSlice 을 등록하여 작성하시오.
⑧ src 폴더안에 -> App.jsx에 작성한 dateApp.jsx 를 import하여 실행 하시오.

<출력 결과>