17. Redux Application

- 리덕스 애플리케이션에서 리덕스를 사용하면 상태 업데이트에 대한 로직을 모듈로 분리하여 관리할 수 있다.
- 또한, 여러 컴퍼넌트에서 동일한 상태를 공유해야 할 때 유용하다.
- 리액트에서는 리덕스를 사용할 때 store인스턴스를 직접사용하기 보다 `react-redux의 connect와 Provider를 사용`

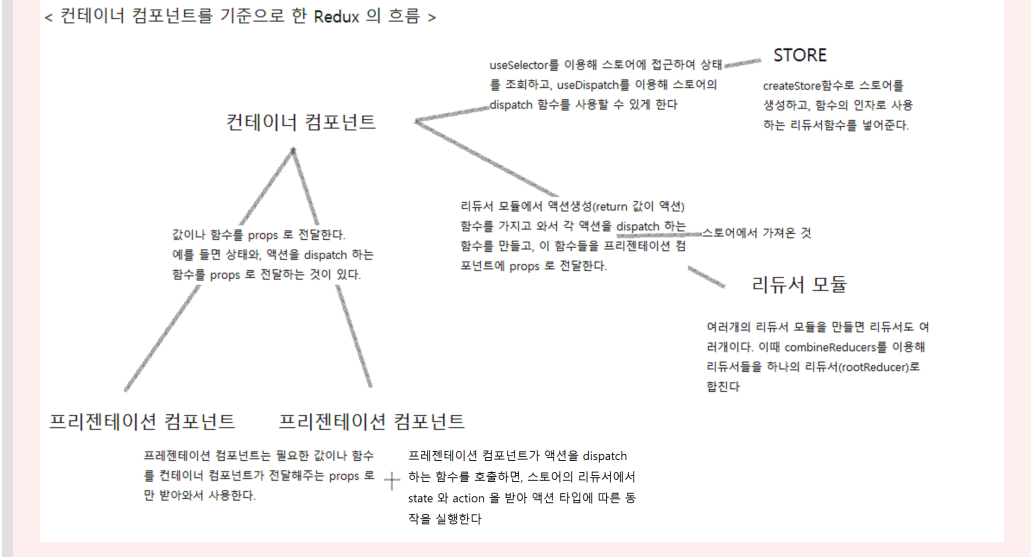
17.1 작업환경 설정

- 설치 : yarn add redux react-redux`
- .prettierrc

```
{
"singleQuote": true,
"semi": true,
"useTabs": false,
"tabWidth": 2,
"trailingComma": "all",
"printWidth": 80
}
```

17.2 UI 준비

- 리덕스를 사용할 때 가장 많이 사용하는 패턴은 `프레젠테이셔널 컴퍼넌트와 컨테이너컴퍼넌트를 분리` 하는 것이다.
  - 프레젠테이셔널 컴퍼넌트 : 상태관리가 이루어지지 않고 props를 받아 와서 화면에 UI를 보여주기만 하는 컴퍼넌트
  - 컨테이너 컴퍼넌트 : 리덕스와 연동되어 리덕스로부터 상태를 받기도 하고 리덕스 스토어에 액션을 디스패치하기도 한다.



17.2.1 예제 component 만들기

src/components/Counter.js

```
const Counter = ({ number, onIncrease, onDecrease }) => {
  return (
    <div>
      <h1>{number}</h1>
      <div>
        <button onClick={onIncrease}>+1</button>
        <button onClick={onDecrease}>-1</button>
      </div>
    </div>
  );
};

export default Counter;
```

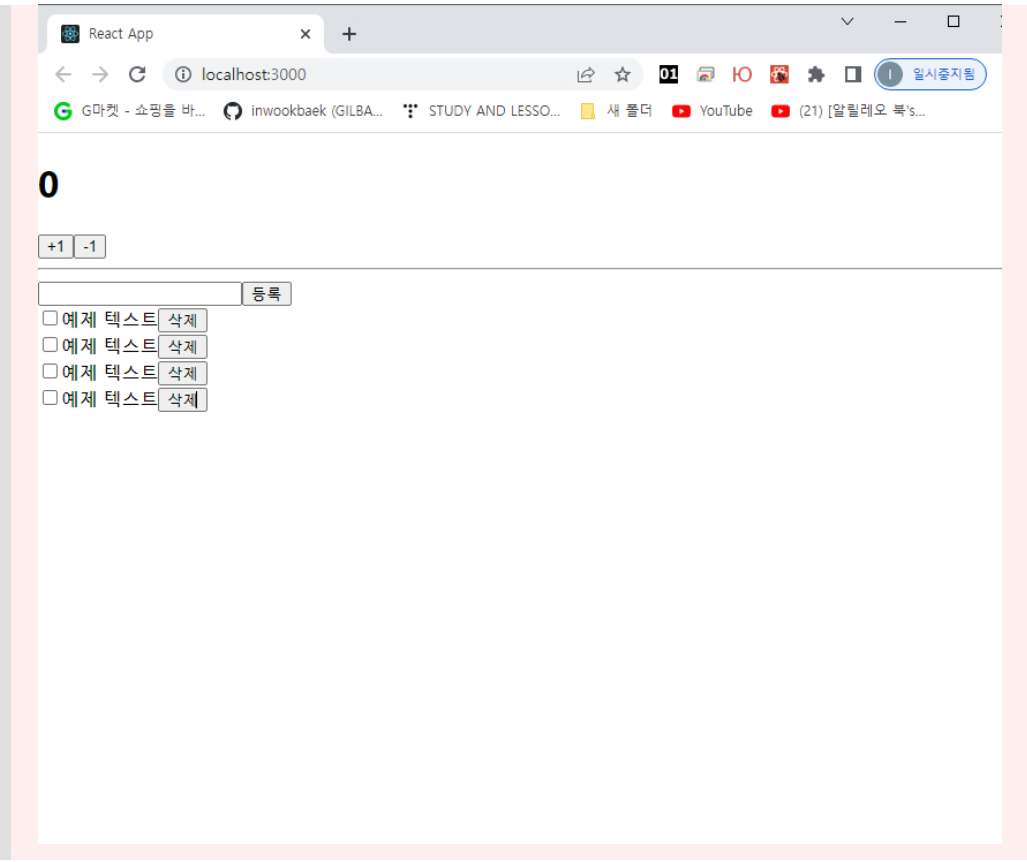
src/components/Todos.js

```
import React from 'react';

const TodoItem = ({ todo, onToggle, onRemove }) => {
  return (
    <div>
      <input type="checkbox"/>
      <span>예제 텍스트</span>
      <button>삭제</button>
    </div>
  );
};

const Todos = ({
  input, // 인풋에 입력되는 텍스트
  todos, // 할 일 목록이 들어있는 객체
  onChangeInput,
  onInsert,
  onToggle,
  onRemove,
}) => {
  const onSubmit = e => {
    e.preventDefault();
  };

  return (
    <div>
      <form onSubmit={onSubmit}>
        <input />
        <button type="submit">등록</button>
      </form>
      <div>
        <TodoItem />
        <TodoItem />
        <TodoItem />
        <TodoItem />
      </div>
    </div>
  );
};
```

```
export default Todos;
```

src/App.js

```
import Counter from './components/Counter';
import Todos from './components/Todos';

function App() {
  return (
    <div>
      <Counter number={0} />
      <hr />
      <Todos />
    </div>
  );
}

export default App;
```



17.3 Redux 코드 작성하기

- 액션타입, 액션생성함수, 리듀서코드를 작성
- 가장 일반적인 방법은 actions, constants, reducers 폴더를 별도 관리
- 세 가지를 파일하나에 작성하는 방식을 Ducks패턴이라고 한다.

17.3.1 모듈작성하기

- 액션타입은 대문자로 정의하고 문자열 내용은 '모듈이름/액션이름'과 같은 형태로 작성

src/modules/counter.js

```
// 17.3.1.1 액션타입 정의
const INCREASE = 'counter/INCREASE';
const DECREASE = 'counter/DECREASE';

// 17.3.1.2 액션생성함수 작성
export const increase = () => ({ type: INCREASE });
export const decrease = () => ({ type: DECREASE });

// 17.3.1.3 초기상태
const initialState = {
  number: 0,
};

// 리듀서함수
function counter(state = initialState, action) {
  switch (action.type) {
    case INCREASE:
      return {
        number: state.number + 1
      };
    case DECREASE:
      return {
        number: state.number - 1
      };
    default:
      return state;
  }
}

export default counter;
```

```
// 17.3.2.1 액션타입정의
const CHANGE_INPUT = 'todos/CHANGE_INPUT'; // 인풋 값을 변경함
const INSERT = 'todos/INSERT'; // 새로운 todo 를 등록함
const TOGGLE = 'todos/TOGGLE'; // todo 를 체크/체크해제 함
const REMOVE = 'todos/REMOVE'; // todo 를 제거함

// 17.3.2. 액션생성함수
export const changeInput = input => ({
  type: CHANGE_INPUT,
  input
})

let id = 3; // insert 가 호출 될 때마다 1씩 더해집니다.
export const insert = text => ({
  type: INSERT,
  todo: {
    id: id++,
    text,
    done: false,
  }
});

export const toggle = id => ({
  type: TOGGLE,
  id
```

```
});

export const remove = id => ({
  type: REMOVE,
  id
});

// 17.3.2.3 초기상태 및 리듀서함수 만들기
const initialState = {
  input: '',
  todos: [
    {
      id: 1,
      text: '리덕스 기초 배우기',
      done: true
    },
    {
      id: 2,
      text: '리액트와 리덕스 사용하기',
      done: false
    }
  ]
};

function todos(state = initialState, action) {
  switch(action.type) {
    case CHANGE_INPUT:
      return {
        ...state,
        input: action.input
      };
    case INSERT:
      return {
        ...state,
        todos: state.todos.concat(action.todo)
      };
    case TOGGLE:
      return {
        ...state,
        todos: state.todos.map(todo =>
          todo.id === action.id ? {...todo, done: !todo.done} : todo)
      };
    case REMOVE:
      return {
        ...state,
        todos: state.todos.filter(todo => todo.id !== action.id)
      };
    default:
      return state;
  }
};

export default todos;
```

modules/index.js - 루트리듀서 만들기

- 이번에는 리듀서를 여러개 만들었기 때문에 createStore함수를 사용하여 store를 만들 때는 리듀서를 하나만 사용해야 한다.
- 따라서, 리듀서를 하나로 합쳐야 하는데 `리덕스에서 제공하는 combineReducer 유틸함수를 사용`하면 쉽게 처리할 수 있다.

```
import { combineReducers } from 'redux';
import counter from './counter';
import todos from './todos';

const rootReducer = combineReducers({
  counter,
  todos,
});

export default rootReducer;
```

7.4 리덕스 적용하기

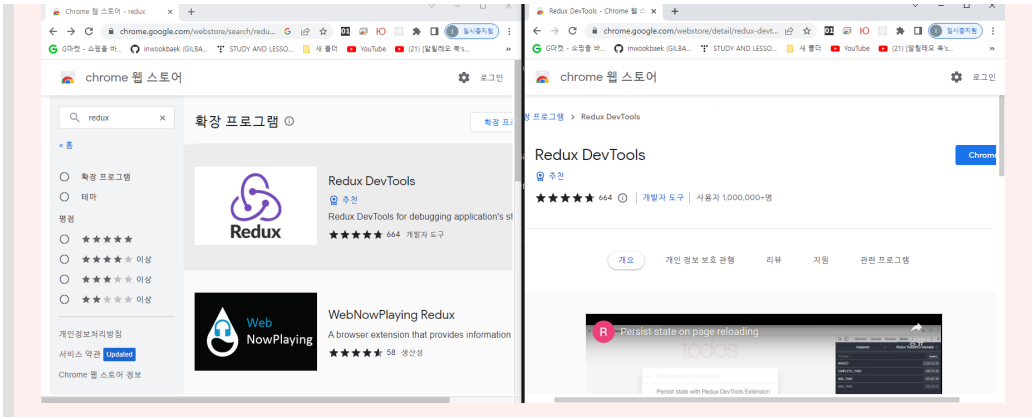
17.4.1 스토어만들기

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { createStore } from 'redux';
import rootReducer from './modules';
import { Provider } from 'react-redux';


// 17.4.1 스토어만들기
const store = createStore(rootReducer);

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  // 17.4.2 Provider컴퍼넌트사용 -> 리덕스적용하기
  <Provider store={store}>
    <App />
  </Provider>
);
```

17.4.3 Redux DevTools 설치

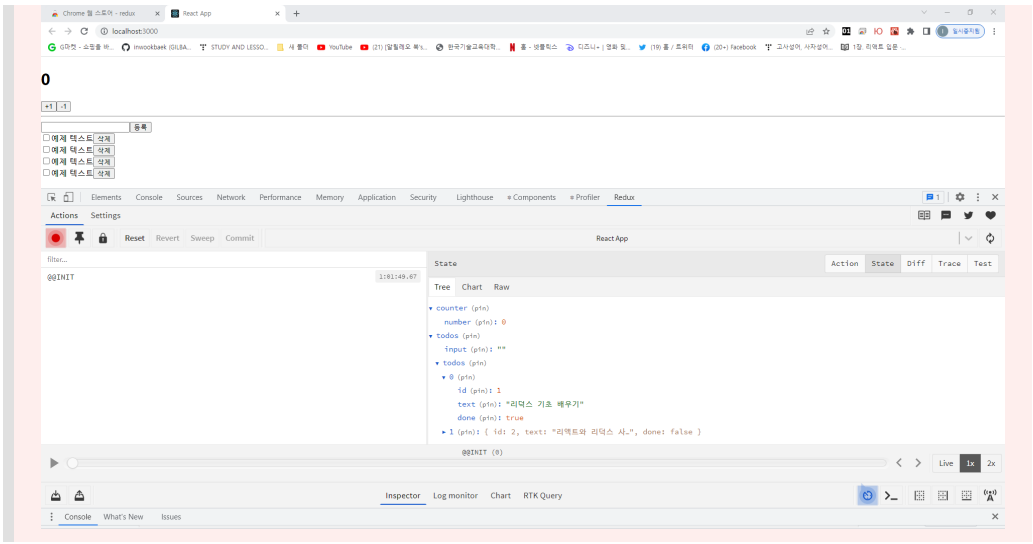
- 크롬 확장프로그램 설치 : chrome.google.com/webstore에서 Redux DevTools 검색, 설치



- 패키지설치 : `yarn add @redux-devtools/extension`

src/index.js - Redux DevTools 적용

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { createStore } from 'redux';
import rootReducer from './modules';
import { Provider } from 'react-redux';
import { devToolsEnhancer } from '@redux-devtools/extension';

// 17.4.1 스토어만들기
// const store = createStore(rootReducer);

// 17.4.3 Redux DevTools 적용
const store = createStore(rootReducer, devToolsEnhancer());

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  // 17.4.2 Provider컴퍼넌트사용 -> 리덕스적용하기
  <Provider store={store}>
    <App />
  </Provider>
);
```



17.5 컨테이너 컴퍼넌트 만들기

17.5.1 CounterContainer

- 리덕스와 연동하려면 react-redux에서 제공하는 `connect함수를 사용`해야 ㅎ나다.

> `connect(mapStateTopProps, mapDispathchToProps)(연동할 컴퍼넌트)`
>
> - mapStateTopProps : 리덕스 스터어 안의 샅태를 컴퍼넌트의 props를 전달하기 위해 설정
> - mapDispathchToProps : 액션생성함수를 컴퍼넌트의 props로 전달하기 위해 사용

- 이렇게 connect함수를 호출하고 나면 또 다른 함수를 반환한다.
- 반환된 함수에 컴퍼넌트를 파라미터로 전달하면 리덕스와 연동된 컴퍼넌트가 만들어 진다.

> - `const makeContainer = connect(mapStateTopProps, mapDispathchToProps)`
> - `makeContainer(타깃 컴퍼넌트)`

containers/CounterContainer.js - (1) console.log()

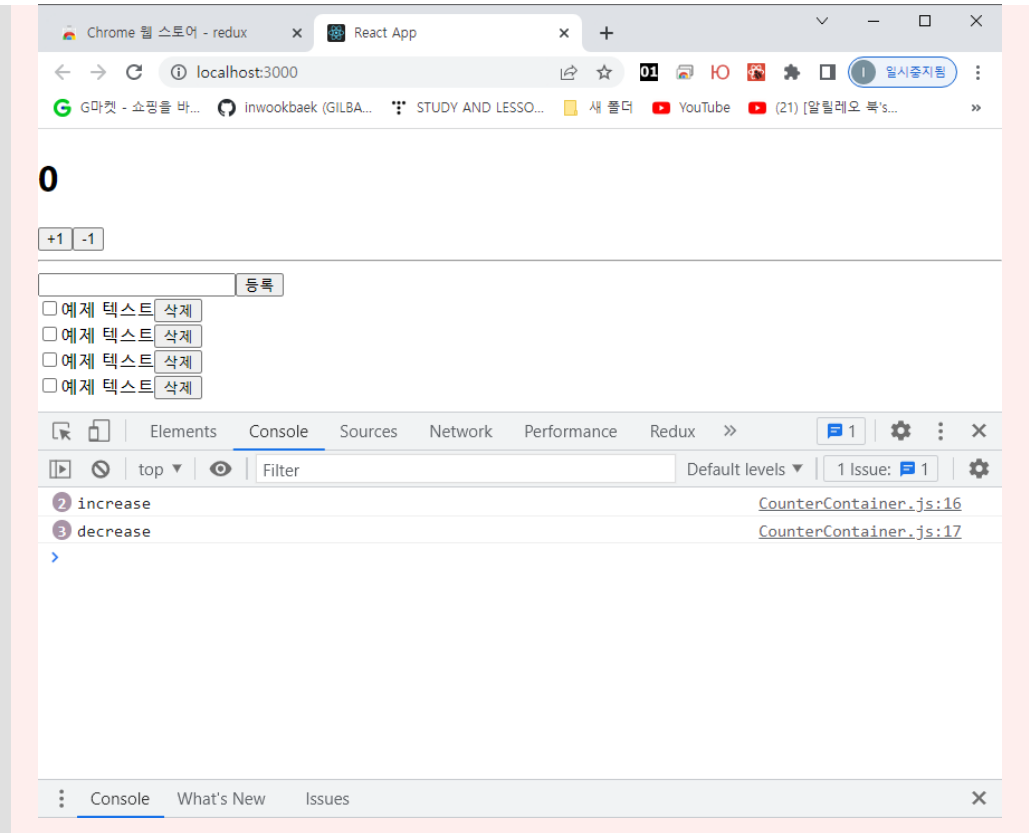
- mapStateToProps와 mapDispatchTOProps에서 반환하는 객체 내부의 값들은 컴퍼넌트의 props로 전달
- mapStateToProps는 state를 받아 오며 이 값은 현재 스토어가 지니고 있는 상태를 가리킨다.
- mapDispatchTOProps의 경우 store의 내장함수 dispatch를 받아 온다.

```
import { connect } from 'react-redux';
import Counter from '../components/Counter';

const CounterContainer = ({number, increase, decrease}) => {
  return (
    <Counter number={number}  onIncrease={increase} onDecrease={decrease}/>
  );
};

const mapStateToProps = state => ({
  number: state.counter.number
});

const mapDispatchTOProps = dispatch => ({
  // 임시함수
  increase: () => console.log('increase'),
  decrease: () => console.log('decrease'),
})

export default connect(mapStateToProps, mapDispatchTOProps)(CounterContainer);
```

src/App.js

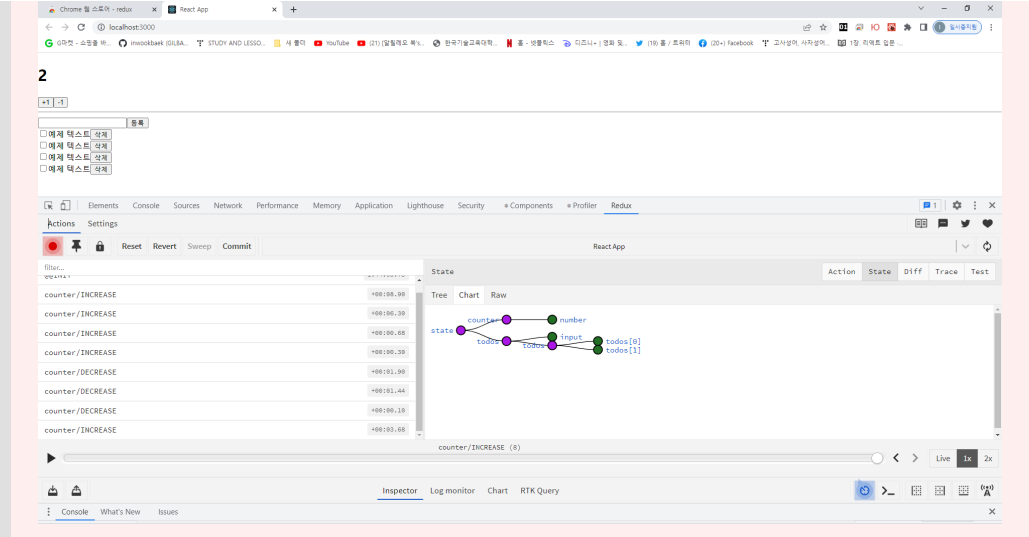
```
import Todos from './components/Todos';
import CounterContainer from './containers/CounterContainer';

function App() {
```

```
    return (
      <div>
        <CounterContainer />
        <hr />
        <Todos />
      </div>
    );
  }

  export default App;
```



containers/CounterContainer.js - (2) 액션생성함수

```
import { connect } from 'react-redux';
import Counter from '../components/Counter';
import { increase, decrease } from '../modules/counter';

const CounterContainer = ({number, increase, decrease}) => {
  return (
    <Counter number={number}  onIncrease={increase} onDecrease={decrease}/>
  );
};

const mapStateToProps = state => ({
  number: state.counter.number
});

const mapDispatchTOProps = dispatch => ({
  // 임시함수
  increase: () => dispatch(increase()),
  decrease: () => dispatch(decrease()),
})

export default connect(mapStateToProps, mapDispatchTOProps)(CounterContainer);
```



containers/CounterContainer.js - (3) connect함수 내부에 익명함수로 선언

```
import { connect } from 'react-redux';
import Counter from '../components/Counter';
import { increase, decrease } from '../modules/counter';

const CounterContainer = ({number, increase, decrease}) => {
  return (
    <Counter number={number}  onIncrease={increase} onDecrease={decrease}/>
  );
};

// const mapStateToProps = state => ({
//   number: state.counter.number
// });

// const mapDispatchTOProps = dispatch => ({
//   // 임시함수
//   increase: () => dispatch(increase()),
//   decrease: () => dispatch(decrease()),
// })

export default connect(
  state => ({ number: state.counter.number }),
  dispatch => ({
    increase: () => dispatch(increase()),
    decrease: () => dispatch(decrease()),
  })
)(CounterContainer);
```

containers/CounterContainer.js - (4) bindActionCreators함수

- 액션생성함수가 많아 질겨우 리덕스에서 제공하는 bindActionCreators함수를 사용하면 간결해 진다.

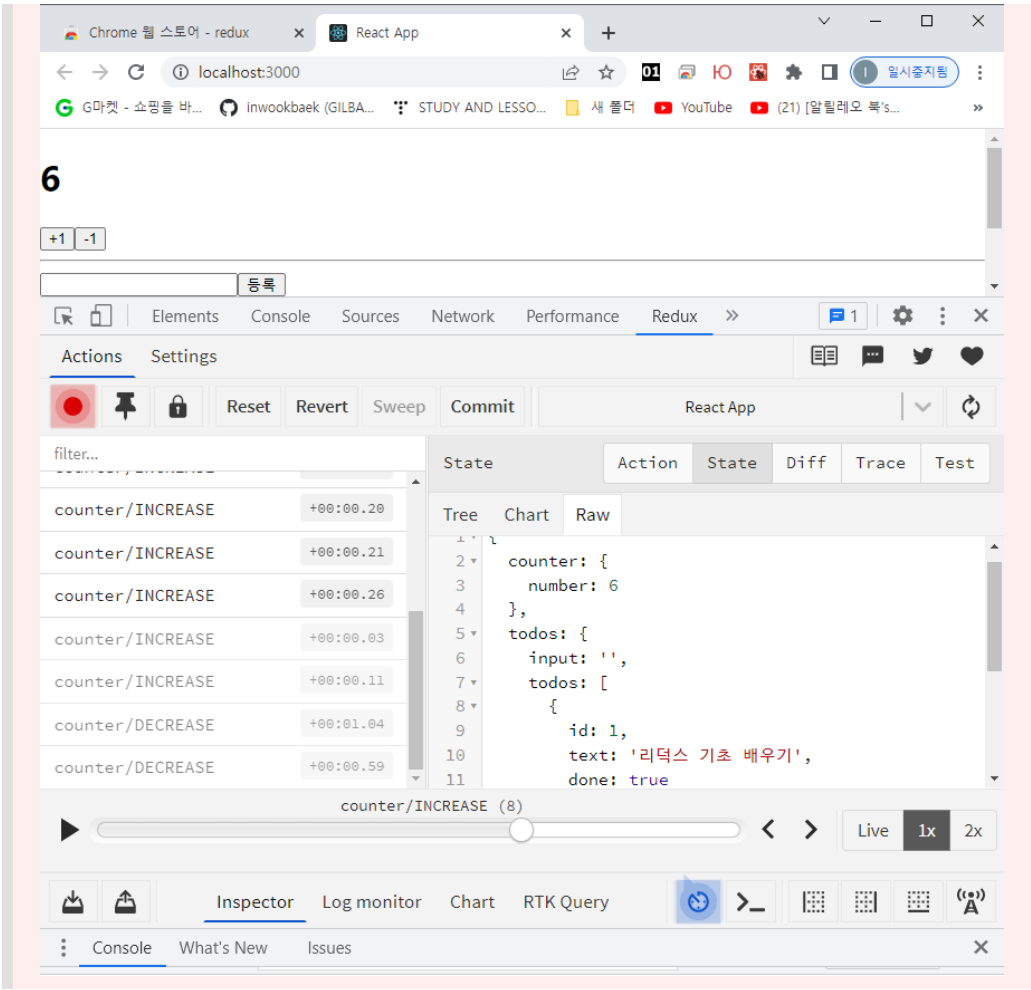
```
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import Counter from '../components/Counter';
import { increase, decrease } from '../modules/counter';

const CounterContainer = ({number, increase, decrease}) => {
  return (
```

```
    <Counter number={number}  onIncrease={increase} onDecrease={decrease}/>
  );
};

// const mapStateToProps = state => ({
//   number: state.counter.number
// });

// const mapDispatchTOProps = dispatch => ({
//   // 임시함수
//   increase: () => dispatch(increase()),
//   decrease: () => dispatch(decrease()),
// })

// 3) 익명함수
// export default connect(
//   state => ({ number: state.counter.number }),
//   dispatch => ({
//     increase: () => dispatch(increase()),
//     decrease: () => dispatch(decrease()),
//   })
// )(CounterContainer);

// 4) bindActionCreators()
export default connect(
  state => ({ number: state.counter.number }),
  dispatch =>
    bindActionCreators({
      increase,
      decrease
    },
    dispatch
    )
)(CounterContainer);
```

containers/CounterContainer.js - (5) 객체형태로 전달

```
import { connect } from 'react-redux';
import Counter from '../components/Counter';
import { increase, decrease } from '../modules/counter';

const CounterContainer = ({number, increase, decrease}) => {
  return (
    <Counter number={number}  onIncrease={increase} onDecrease={decrease}/>
  );
};

// const mapStateToProps = state => ({
//   number: state.counter.number
// });

// const mapDispatchTOProps = dispatch => ({
//   // 임시함수
//   increase: () => dispatch(increase()),
//   decrease: () => dispatch(decrease()),
// })
```

```
// 3) 익명함수
// export default connect(
//   state => ({ number: state.counter.number }),
//   dispatch => ({
//     increase: () => dispatch(increase()),
//     decrease: () => dispatch(decrease()),
//   })
// )(CounterContainer);

// 4) bindActionCreators()
// export default connect(
//   state => ({ number: state.counter.number }),
//   dispatch =>
//     bindActionCreators({
//       increase,
//       decrease
//     },
//     dispatch
//     )
// )(CounterContainer);

// 5) mapDispatchToProps대신에 액션생성함수로 이루어진 객체형태로 전달
export default connect(
  state => ({ number: state.counter.number }),
  {
    increase,
    decrease
  }
)(CounterContainer);
```



17.5.2 TodosContainer

containers/TodosContiner.js

```
import { connect } from 'react-redux';
import { changeInput, insert, toggle, remove } from '../modules/todos';
import Todos from '../components/Todos';

const TodosContainer = ({
    input,
    todos,
    changeInput,
    insert,
    toggle,
    remove
  }) => {
    return (
      <Todos
        input={input}
        todos={todos}
        onChangeInput= {changeInput}
        onInsert={insert}
        onToggle={toggle}
        onRemove={remove}
      />
    );
  };

export default connect(
  // 비구조화할당을 통해 todos를 분리하여
  // state.todos.input대신 todos.input을 사용
  ({ todos }) => ({
    input: todos.input,
    todos: todos.todos,
  }),
  {
    changeInput,
    insert,
    toggle,
    remove
  }
)(TodosContainer);
```

src/App.js

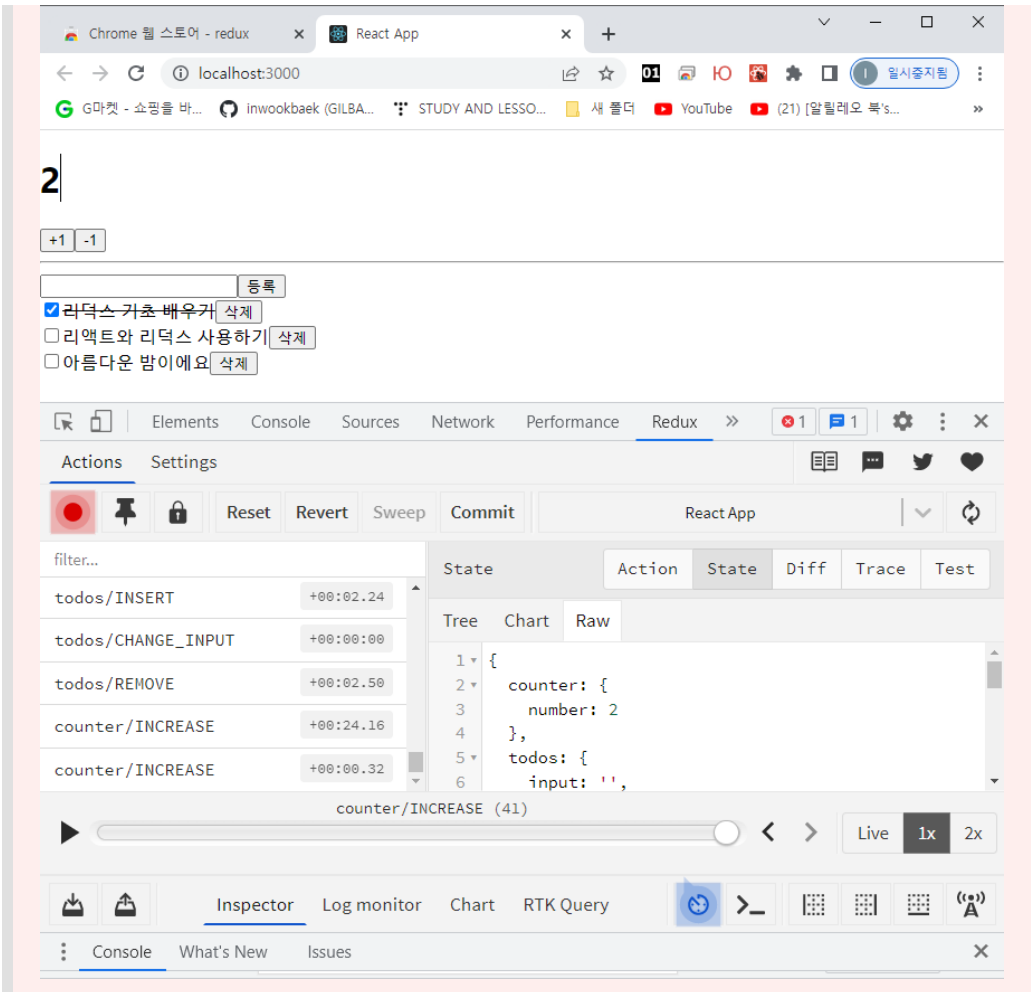
```
import CounterContainer from './containers/CounterContainer';
import TodosContainer from './containers/TodosContainer';

function App() {
  return (
    <div>
      <CounterContainer />
      <hr />
      {/* <Todos /> */}
      <TodosContainer />
    </div>
  );
}
```

```
export default App;
```

components/Totos.js

```
import React from 'react';

const TodoItem = ({ todo, onToggle, onRemove }) => {
  return (
    <div>
      <input type="checkbox"
        onClick={() => onToggle(todo.id)}
        checked={todo.done}
        readOnly={true}
      />
      <span style={{textDecoration: todo.done ? 'line-through' : 'none'}}>
{todo.text}</span>
      <button onClick={() => onRemove(todo.id)}>삭제</button>
    </div>
  );
};

const Todos = ({
  input, // 인풋에 입력되는 텍스트
  todos, // 할 일 목록이 들어있는 객체
  onChangeInput,
  onInsert,
  onToggle,
  onRemove,
}) => {
  const onSubmit = e => {
    e.preventDefault();
    onInsert(input);
    onChangeInput(''); // 등록후 초기화
  };

  const onChange = e => onChangeInput(e.target.value);


  return (
    <div>
      <form onSubmit={onSubmit}>
        <input value={input} onChange={onChange}/>
        <button type="submit">등록</button>
      </form>
      <div>
        {
          todos.map(todo => (
            <TodoItem
              todo={todo}
              key={todo.id}
              onToggle={onToggle}
              onRemove={onRemove}
            />
          ))
        }
      </div>
```

```
      </div>
    );
};

export default Todos;
```



17.6 리덕스 편하게 사용하기

- redux-actions와 immer라이브러를 사용하면 편하게 리덕스를 사용할 수 있다.

17.6.1 redux-actions

- 설치 : `yarn add redux-actions`
- redux-actions은 액션생성함수를 짧은 코드로 작성할 수 있다.
- switch문 대신에 `handleActions`라는 함수를 사용하여 각 액션마다 업데이트함수를 설정하는 형식으로 작성할 수 있다.

17.6.1.1 counter모듈에 적용하기

- createAction을 사용하면 매번 객체를 직접 만들어 줄 필요 없이 간단하게 액션생성함수를 선언할 수 있다.
- 리듀서 함수는 handleActions라는 함수를 사용, 첫 번째는 액션에 대한 업데이트함수 두 번째는 초기상태를 선언한다.

modules/counter.js

```
// 17.3.1.1 액션타입 정의
import { createAction, handleActions } from 'redux-actions';
const INCREASE = 'counter/INCREASE';
const DECREASE = 'counter/DECREASE';

// 17.3.1.2 액션생성함수 작성
// export const increase = () => ({ type: INCREASE });
// export const decrease = () => ({ type: DECREASE });

// 17.3.1.3 초기상태
const initialState = {
  number: 0,
};

// 리듀서함수
// function counter(state = initialState, action) {
//   switch (action.type) {
//     case INCREASE:
//       return {
//         number: state.number + 1
//       };
//     case DECREASE:
//       return {
//         number: state.number - 1
//       };
//     default:
//       return state;
//   }
// }

// 17.6.1.1 redux-actions
export const increase = createAction(INCREASE);
export const decrease = createAction(DECREASE);

const counter = handleActions({
  [INCREASE]: (state, action) => ({ number: state.number + 1 }),
  [DECREASE]: (state, action) => ({ number: state.number - 1 })
}, initialState);

export default counter;
```

17.6.1.2 todos모듈에 적용하기

- `carateAction`을 액션을 만들면 액션에 필요한 추가 데이터는 `payload`라는 이름을 사용한다. 예를 들면

```
const MY_ACTION = 'sample/MY_ACTION';
const myAction = createAction(MY_ACTION);
const action = myAction('hello world');
// 결과 : {type: MY_ACTION, payload: 'hello world' }
```

- 액션생성함수에서 받은 파라미터를 변형하고 실차다면 createAction함수의 두 번째 함수에 payload를 정의한 함수를 별도 선언

```
const MY_ACTION = 'sample/MY_ACTION';
const myAction = createAction(MY_ACTION, text => `${text}`);
const action = myAction('hello world');
// 결과 : {type: MY_ACTION, payload: 'hello world' }
```

modules/todos.js

```
import { createAction, handleActions } from "redux-actions";

// 17.3.2.1 액션타입정의
const CHANGE_INPUT = 'todos/CHANGE_INPUT'; // 인풋 값을 변경함
const INSERT = 'todos/INSERT'; // 새로운 todo 를 등록함
const TOGGLE = 'todos/TOGGLE'; // todo 를 체크/체크해제 함
const REMOVE = 'todos/REMOVE'; // todo 를 제거함

// 17.3.2. 액션생성함수
// export const changeInput = input => ({
//   type: CHANGE_INPUT,
//   input
// })

// let id = 3; // insert 가 호출 될 때마다 1씩 더해집니다.
// export const insert = text => ({
//   type: INSERT,
//   todo: {
//     id: id++,
//     text,
//     done: false,
//   }
// });

// export const toggle = id => ({
//   type: TOGGLE,
//   id
// });

// export const remove = id => ({
//   type: REMOVE,
//   id
// });

// 17.3.2.3 초기상태 및 리듀서함수 만들기
const initialState = {
  input: '',
  todos: [
    {
      id: 1,
      text: '리덕스 기초 배우기',
      done: true
    },
    {
      id: 2,
      text: '리액트와 리덕스 사용하기',
      done: false
    }
  ]
};

// function todos(state = initialState, action) {
//   switch(action.type) {
//     case CHANGE_INPUT:
//       return {
```

```
//         ...state,
//         input: action.input
//       };
//     case INSERT:
//       return {
//         ...state,
//         todos: state.todos.concat(action.todo)
//       };
//     case TOGGLE:
//       return {
//         ...state,
//         todos: state.todos.map(todo =>
//           todo.id === action.id ? {...todo, done: !todo.done} : todo)
//       };
//     case REMOVE:
//       return {
//         ...state,
//         todos: state.todos.filter(todo => todo.id !== action.id)
//       };
//     default:
//       return state;
//   }
// };

// 17.6.1.2 createAction 적용
export const changeInput = createAction(CHANGE_INPUT, input => input);

let id = 3;
export const insert = createAction(INSERT, text => ({
  id: id++,
  text,
  done: false,
}));

export const toggle = createAction(TOGGLE, id => id);
export const remove = createAction(REMOVE, id => id);

const todos = handleActions({
  [CHANGE_INPUT]: (state, action) => ({ ...state,
      input: action.payload
  }),
  [INSERT]: (state, action) => ({
    ...state,
    todos: state.todos.concat(action.payload)
  }),
  [TOGGLE]: (state, action) => ({
    ...state,
    todos: state.todos.map(todo =>
      todo.id === action.payload ? { ...todo, done: !todo.done} : todo
  )}),
  [REMOVE]: (state, action) => ({
    ...state,
    todos: state.todos.filter(todo => todo.id !== action.payload)
  }),
  }, initialState
);
```

```
export default todos;
```

- insert의 경우 todo객체를 액션객체안에 넣어야 하기 때문에 두 번째에 text를 전달하면 todo객체가 반환되는 함수를 전달
- 나머지 함수는 text => text, id => id형태로 전달된 파라미터 그대로 반환함수를 전달
- 이 작업은 필수는 아니나 코드를 보면 액션생성함수의 파라미터값이 필요한지 쉽게 파악 가능
- createAction으로 만든 액션생성함수는 파라미터로 받은 값을 객체 안에 전달할 때 원하는 이름으로 넣는 것이 아니라
- action.id, action.todo와 같이 action.payload라는 이름을 공통적으로 전달하게 된다.
- 그렇기 때문에 기존 업데이트 로직에서도 모두 action.payload값을 조회하여 업데이트하도록 해야 한다.
- `액션상성함수는 액션에 필요한 추가데이터를 모두 payload라는 이름으로 사용한다`
- 그렇기 때문에 `모두 공통적으로 action.payload값을 조회하도록 리듀서를 구현` 해야 한다.

modules/todos.js - 객체 비구조화할당 문법 적용

- 모든 추가 데이터 값을 action.pyaload로 사용하기 때문에 햇갈리 수가 있다.
- `객체 비구조화 할당 문법으로 action값의 payload이름을 새로 설정`해 주면 action.payload가 어떤 값인지 파악하기 쉬다.

```
import { createAction, handleActions } from "redux-actions";

// 17.3.2.1 액션타입정의
const CHANGE_INPUT = 'todos/CHANGE_INPUT'; // 인풋 값을 변경함
const INSERT = 'todos/INSERT'; // 새로운 todo 를 등록함
const TOGGLE = 'todos/TOGGLE'; // todo 를 체크/체크해제 함
const REMOVE = 'todos/REMOVE'; // todo 를 제거함

// 17.3.2. 액션생성함수
// export const changeInput = input => ({
//   type: CHANGE_INPUT,
//   input
// })

// let id = 3; // insert 가 호출 될 때마다 1씩 더해집니다.
// export const insert = text => ({
//   type: INSERT,
//   todo: {
//     id: id++,
//     text,
//     done: false,
//   }
// });

// export const toggle = id => ({
//   type: TOGGLE,
//   id
```

```
// });

// export const remove = id => ({
//   type: REMOVE,
//   id
// });

// 17.3.2.3 초기상태 및 리듀서함수 만들기
const initialState = {
  input: '',
  todos: [
    {
      id: 1,
      text: '리덕스 기초 배우기',
      done: true
    },
    {
      id: 2,
      text: '리액트와 리덕스 사용하기',
      done: false
    }
  ]
};

// function todos(state = initialState, action) {
//   switch(action.type) {
//     case CHANGE_INPUT:
//       return {
//         ...state,
//         input: action.input
//       };
//     case INSERT:
//       return {
//         ...state,
//         todos: state.todos.concat(action.todo)
//       };
//     case TOGGLE:
//       return {
//         ...state,
//         todos: state.todos.map(todo =>
//           todo.id === action.id ? {...todo, done: !todo.done} : todo)
//       };
//     case REMOVE:
//       return {
//         ...state,
//         todos: state.todos.filter(todo => todo.id !== action.id)
//       };
//     default:
//       return state;
//   }
// };

// 17.6.1.2 createAction 적용
export const changeInput = createAction(CHANGE_INPUT, input => input);

let id = 3;
export const insert = createAction(INSERT, text => ({
```

```
  id: id++,
  text,
  done: false,
}));

export const toggle = createAction(TOGGLE, id => id);
export const remove = createAction(REMOVE, id => id);

// hancleActions 리듀서함수
// const todos = handleActions({
//   [CHANGE_INPUT]: (state, action) => ({ ...state,
//       input: action.payload
//   }),
//   [INSERT]: (state, action) => ({
//     ...state,
//     todos: state.todos.concat(action.payload)
//   }),
//   [TOGGLE]: (state, action) => ({
//     ...state,
//     todos: state.todos.map(todo =>
//       todo.id === action.payload ? { ...todo, done: !todo.done} : todo
//   )}),
//   [REMOVE]: (state, action) => ({
//     ...state,
//     todos: state.todos.filter(todo => todo.id !== action.payload)
//   }),
//   }, initialState
// );

// hancleActions 리듀서함수 - action.payload의 이름을 새로 설정
const todos = handleActions({
  [CHANGE_INPUT]: (state, { payload: input }) => ({ ...state,
      input: input,
  }),
  [INSERT]: (state, { payload: todo }) => ({
    ...state,
    todos: state.todos.concat(todo)
  }),
  [TOGGLE]: (state, { payload: id }) => ({
    ...state,
    todos: state.todos.map(todo =>
      todo.id === id ? { ...todo, done: !todo.done } : todo
  )}),
  [REMOVE]: (state, { payload: id }) => ({
    ...state,
    todos: state.todos.filter(todo => todo.id !== id)
  }),
  }, initialState
);

export default todos;
```

17.6.2 immer

- 설치 : `yarn add immer`

- 객체의 구조가 복잡해 지거나 객체로 이루어진 배열을 다룰 경우 immer를 사용하면 편리하게 상태관리를 할 수 있다.

modules/todos.js - immer적용

```
import { createAction, handleActions } from "redux-actions";
import produce from 'immer';

// 17.3.2.1 액션타입정의
const CHANGE_INPUT = 'todos/CHANGE_INPUT'; // 인풋 값을 변경함
const INSERT = 'todos/INSERT'; // 새로운 todo 를 등록함
const TOGGLE = 'todos/TOGGLE'; // todo 를 체크/체크해제 함
const REMOVE = 'todos/REMOVE'; // todo 를 제거함

// 17.3.2. 액션생성함수
// export const changeInput = input => ({
//   type: CHANGE_INPUT,
//   input
// })

// let id = 3; // insert 가 호출 될 때마다 1씩 더해집니다.
// export const insert = text => ({
//   type: INSERT,
//   todo: {
//     id: id++,
//     text,
//     done: false,
//   }
// });

// export const toggle = id => ({
//   type: TOGGLE,
//   id
// });

// export const remove = id => ({
//   type: REMOVE,
//   id
// });

// 17.3.2.3 초기상태 및 리듀서함수 만들기
const initialState = {
  input: '',
  todos: [
    {
      id: 1,
      text: '리덕스 기초 배우기',
      done: true
    },
    {
      id: 2,
      text: '리액트와 리덕스 사용하기',
      done: false
    }
  ]
};
```

```
// function todos(state = initialState, action) {
//   switch(action.type) {
//     case CHANGE_INPUT:
//       return {
//         ...state,
//         input: action.input
//       };
//     case INSERT:
//       return {
//         ...state,
//         todos: state.todos.concat(action.todo)
//       };
//     case TOGGLE:
//       return {
//         ...state,
//         todos: state.todos.map(todo =>
//           todo.id === action.id ? {...todo, done: !todo.done} : todo)
//       };
//     case REMOVE:
//       return {
//         ...state,
//         todos: state.todos.filter(todo => todo.id !== action.id)
//       };
//     default:
//       return state;
//   }
// };

// 17.6.1.2 createAction 적용
export const changeInput = createAction(CHANGE_INPUT, input => input);

let id = 3;
export const insert = createAction(INSERT, text => ({
  id: id++,
  text,
  done: false,
}));

export const toggle = createAction(TOGGLE, id => id);
export const remove = createAction(REMOVE, id => id);

// hancleActions 리듀서함수
// const todos = handleActions({
//   [CHANGE_INPUT]: (state, action) => ({ ...state,
//       input: action.payload
//   }),
//   [INSERT]: (state, action) => ({
//     ...state,
//     todos: state.todos.concat(action.payload)
//   }),
//   [TOGGLE]: (state, action) => ({
//     ...state,
//     todos: state.todos.map(todo =>
//       todo.id === action.payload ? { ...todo, done: !todo.done} : todo
//   )}),
//   [REMOVE]: (state, action) => ({
//     ...state,
```

```
//     todos: state.todos.filter(todo => todo.id !== action.payload)
//   }),
//   }, initialState
// );

// hancleActions 리듀서함수 - action.payload의 이름을 새로 설정
// const todos = handleActions({
//   [CHANGE_INPUT]: (state, { payload: input }) => ({ ...state, input }),
//   [INSERT]: (state, { payload: todo }) => ({
//     ...state,
//     todos: state.todos.concat(todo)
//   }),
//   [TOGGLE]: (state, { payload: id }) => ({
//     ...state,
//     todos: state.todos.map(todo =>
//       todo.id === id ? { ...todo, done: !todo.done} : todo)
//   }),
//   [REMOVE]: (state, { payload: id }) => ({
//     ...state,
//     todos: state.todos.filter(todo => todo.id !== id)
//   }),
//   }, initialState
// );

// 17.6.2 immer
const todos = handleActions({
    [CHANGE_INPUT]: (state, { payload: input }) =>
      produce(state, draft => {
        draft.input = input;
      }),
    [INSERT]: (state, { payload: todo }) =>
      produce(state, draft => {
        draft.todos.push(todo)
    }),
    [TOGGLE]: (state, { payload: id }) =>
      produce(state, draft => {
        const todo = draft.todos.find(todo => todo.id === id);
        todo.done = !todo.done;
    }),
    [REMOVE]: (state, { payload: id }) =>
      produce(state, draft => {
        const index = draft.todos.findIndex(todo => todo.id === id);
        draft.todos.splice(index, 1);
    }),
  }, initialState
);
export default todos;
```

17.7 Hooks를 사용 컨테이너 컴퍼넌트 만들기

- 리덕스 스토어와 연동된 컨테이너 컴퍼넌트를 만들 때 connecct함수를 사용하는 대신 react-redux 제공 Hooks를 사용할 수 있다.

17.7.1 `useSelctor`로 상태조회

- `useSelector Hook`을 사용하면 `connect` 함수를 사용하지 않고도 리덕스상태를 조회 할 수 있다.
  - 사용법 : `const result = useSelector(상태선택함수)`
  - 상태선택함수는 mapStateProps와 형태가 똑같다. connect함수 대신 useSelector를 사용

containers/CounterContainerUseSelector.js

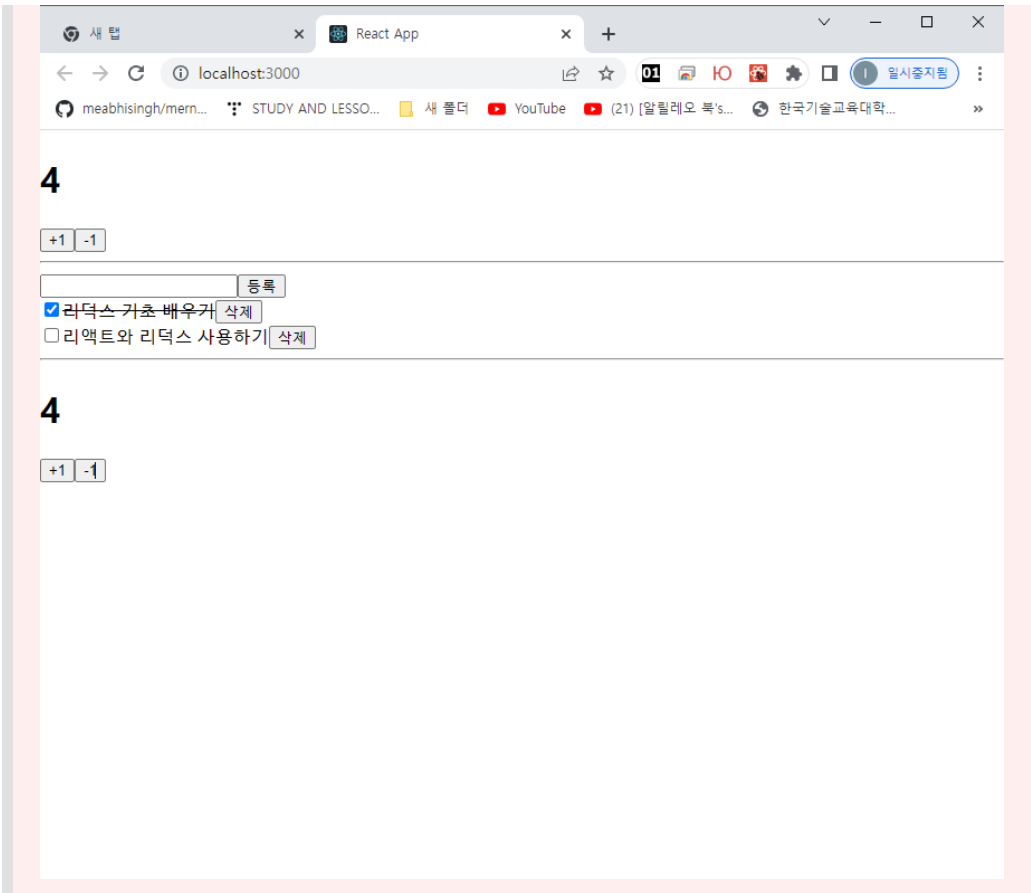
- 컴퍼넌트가 리랜더링할 때마다 onIncrease, onDecrease함수가 새롭게 생성이 된다.
- 이 상황을 막기 위해 useCallback함수로 액션을 디스패치하는 함수를 감싸 주는 것이 좋다.
- useDispatch를 사용할 때는 useCallback과 함께 사용하는 것이 좋다.

```
import { useCallback } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import Counter from '../components/Counter';
import { decrease, increase } from '../modules/counter';

const CounterContainerUseSelector = () => {

  const number = useSelector(state => state.counter.number);
  const dispatch = useDispatch();
  const onIncrease = useCallback(() => dispatch(increase()), [dispatch]);
  const onDecrease = useCallback(() => dispatch(decrease()), [dispatch]);

  return (
    <Counter
      number={number}
      onIncrease={onIncrease}
      onDecrease={onDecrease}
    />
  );

};

export default CounterContainerUseSelector;
```

src/App.js - useSelector

```
import CounterContainer from './containers/CounterContainer';
import CounterContainerUseSelector from
'./containers/CounterContainerUseSelector';
import TodosContainer from './containers/TodosContainer';
import TodosContainerUseSelector from './containers/TodosContainerUseSelector';

function App() {
  return (
    <div>
      <CounterContainer />
      <hr />
      {/* <Todos /> */}
      <TodosContainer />
      <hr/>
      <h3>useSelector</h3>
      <CounterContainerUseSelector />
      <hr/>
```

```
      </div>
    );
}

export default App;
```



17.7.3 `useStore`를 사용하여 리덕스 스토어 사용하기

- `useStore hooks`를 사용하면 컴퍼넌트 내부에서 리덕스 스토어 객체를 직접 사용할 수 있다.
- 사용법은 아래와 같지만 이를 사용해야 하는 상황은 흔치 않다.

```
const store = useStore();
store.dispatch({ type: 'SAMPLE_ACTION' });
store.getStore();
```

17.7.4 TodosContainer를 Hooks로 전환하기

- TodosContainer를 connect함수 대신 useSelector와 useDispatch Hooks를 사용하는 형태로 변경

container/TodosContainerUseSelector.js - Hooks로 전환

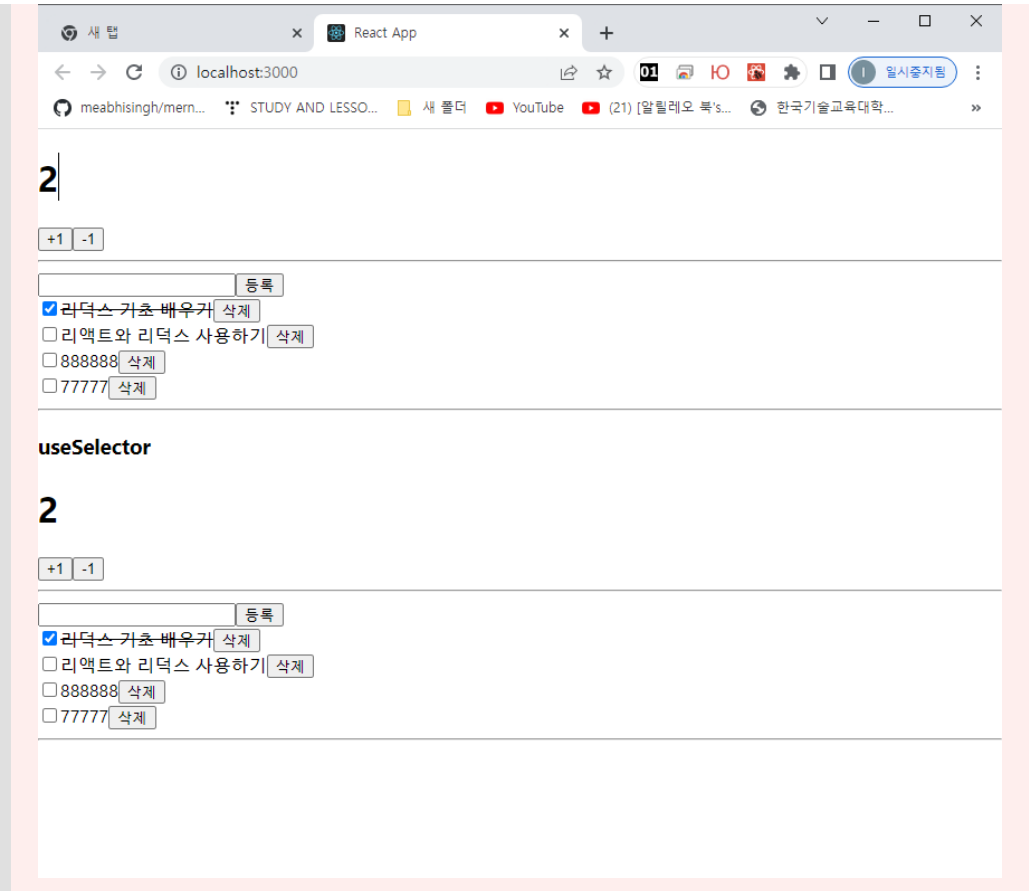
- useSelector를 사용할 때 비구조화 할당문법을 활용
- useDispatch를 사용할 때 각 액션을 디스패치하는 함ㅅ구를 작성

```
import { useDispatch, useSelector } from 'react-redux';
import { changeInput, insert, toggle, remove } from '../modules/todos';
import Todos from '../components/Todos';
import { useCallback } from 'react';
```

```
const TodosContainerUseSelector = () => {

  const { input, todos} = useSelector(({ todos }) => ({
    input: todos.input,
    todos: todos.todos
  }));
  const dispatch = useDispatch();
  const onChangeInput = useCallback(input => dispatch(changeInput(input)),
[dispatch]);
  const onInsert = useCallback(text => dispatch(insert(text)), [dispatch]);
  const onToggle = useCallback(id => dispatch(toggle(id)), [dispatch]);
  const onRemove = useCallback(id => dispatch(remove(id)), [dispatch]);

  return (
    <Todos
      input={input}
      todos={todos}
      onChangeInput={onChangeInput}
      onInsert={onInsert}
      onToggle={onToggle}
      onRemove={onRemove}
    />

  );
}

export default TodosContainerUseSelector;
```

src/App.js - useSelector

```
import CounterContainer from './containers/CounterContainer';
import CounterContainerUseSelector from
'./containers/CounterContainerUseSelector';
import TodosContainer from './containers/TodosContainer';
import TodosContainerUseSelector from './containers/TodosContainerUseSelector';

function App() {
  return (
    <div>
      <CounterContainer />
      <hr />
      {/* <Todos /> */}
      <TodosContainer />
      <hr/>
      <h3>useSelector</h3>
      <CounterContainerUseSelector />
      <hr/>
      <TodosContainerUseSelector />
      <hr/>
    </div>
  );
}

export default App;
```



17.75 useActions 유틸 Hook을 만들어서 사용하기

- useActions는 원래 react-redux에 내장할 계획이었지만 꼭 필요하지 않다고 판단하여 제외된 Hook
  - 대신 참고사이트 제공 : https://react-redux.js.org/next/api/hooks#recipe-useactions
- 이 Hook을 사용하면 여러 개의 액션을 사용할 경우 코드를 훨씬 간단하게 작성할 수 있따.
- src/lib/useActions.js 파일 작성하기

src/lib/useActions.js

```
import { useMemo } from 'react'
import { useDispatch } from 'react-redux'
import { bindActionCreators } from 'redux';

export default function useActions(actions, deps) {
  const dispatch = useDispatch();
  return useMemo(
    () => {
      if(Array.isArray(actions)) {
        return actions.map(a => bindActionCreators(a, dispatch));
      }
      return bindActionCreators(actions, dispatch);
    },
    // eslint-disable-next-line react-hooks/exhaustive-deps
    deps ? [dispatch, ...deps] : deps
  );
}
```

- useActions Hook은 액션생성함수를 액션을 디스패치하는 함수로 변환해 준다.
- 액션생성함수를 사용하여 액션객체를 만들고 이를 스토어에 디스패치하는 작업을 해주는 함수를 자동으로 생성한다.
- useActions는 첫 번째는 액션생성함수로 이루어진 배열, 두 번째는 deps배열이고 이 배열의 원소가 변경되면 애션을 디스패치하는 함수를 새로 생성한다.

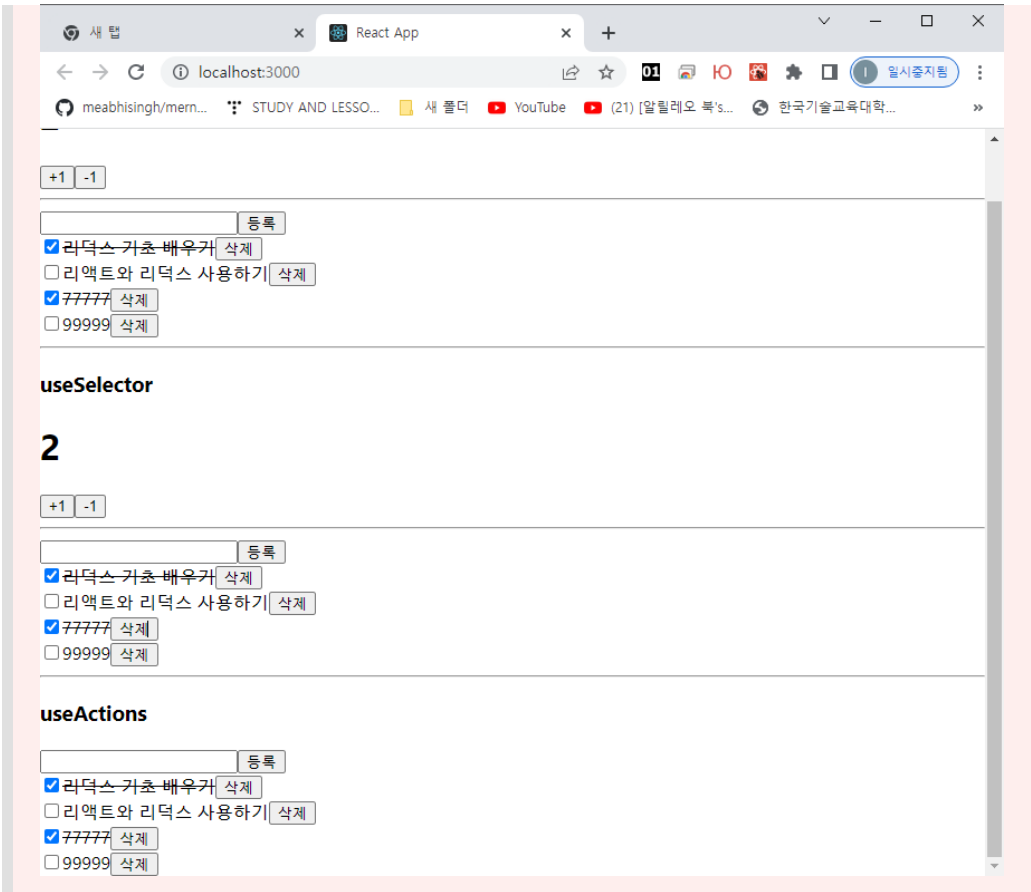
containers/TodoContainerUseActions.js

```
import {  useSelector } from 'react-redux';
import { changeInput, insert, toggle, remove } from '../modules/todos';
import Todos from '../components/Todos';
import useActions from '../lib/useActions';

const TodosContainerUseActions = () => {

  const { input, todos} = useSelector(({ todos }) => ({
    input: todos.input,
    todos: todos.todos
  }));

  const [onChangeInput,onInsert,  onToggle, onRemove] = useActions(
    [changeInput, insert, toggle, remove],
    []
  );

  return (
    <Todos
      input={input}
      todos={todos}
      onChangeInput={onChangeInput}
      onInsert={onInsert}
      onToggle={onToggle}
      onRemove={onRemove}
    />

  );
}

export default TodosContainerUseActions;
```

sr/App.js

```
import CounterContainer from './containers/CounterContainer';
import CounterContainerUseSelector from
'./containers/CounterContainerUseSelector';
import TodosContainer from './containers/TodosContainer';
import TodosContainerUseActions from './containers/TodosContainerUseActions';
import TodosContainerUseSelector from './containers/TodosContainerUseSelector';

function App() {
  return (
    <div>
      <CounterContainer />
      <hr />
      {/* <Todos /> */}
      <TodosContainer />
      <hr/>
```

```
        <h3>useSelector</h3>
        <CounterContainerUseSelector />
        <hr/>
        <TodosContainerUseSelector />
        <hr/>
        <h3>useActions</h3>
        <TodosContainerUseActions />
      </div>
    );
}

export default App;
```



17.7.6 connect함수와 주요 차이점

- 컨테이너 컴퍼넌트를 만들 때 connect함수를 사용해도 좋고 useSelector와 useDispatch를 사용해도 좋다.
- Hooks를 사용하여 컨테이너 컴퍼넌트를 만들 때 알아야할 차이점이 있다.
  1. connect함수를 사용할 경우, 해당 컨테이너 컴퍼넌트의 부모가 리랜더링될 때 해당 컴퍼넌트의 props가 변경되지 않았다면 리랜더링이 자동방지되어 성능이 최적화 된다.'
  2. useSelctor를 사용할 경우 최적화가 자동으로 되지 않기 때문에 최적호를 위해 React.useMemo를 컨테이너 컴포넌트에 사용해 주어야 한다.

containers/TodoContainerUseActions.js

```
(...)

export default React.memo(TodosContainerUseActions);
```