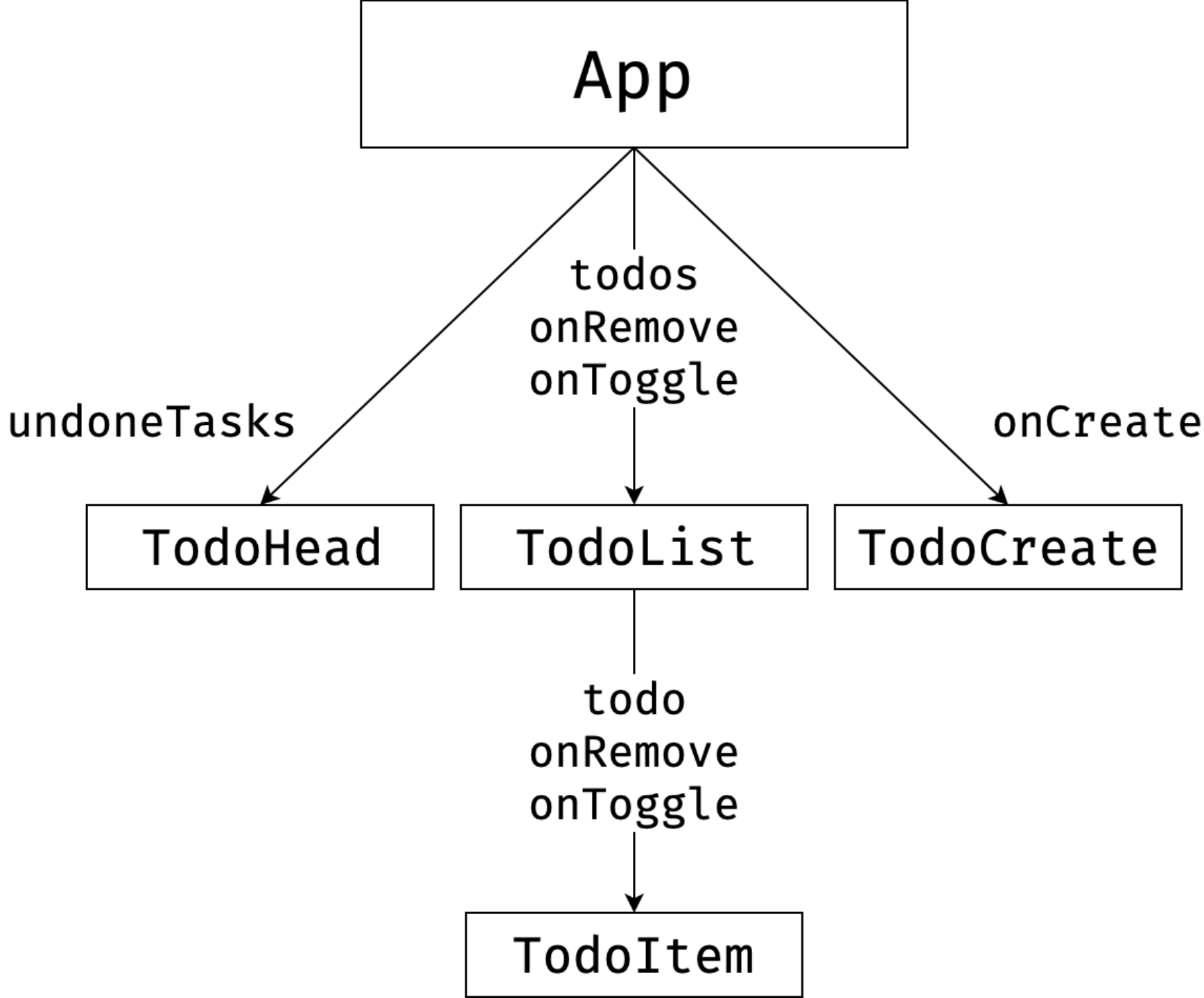
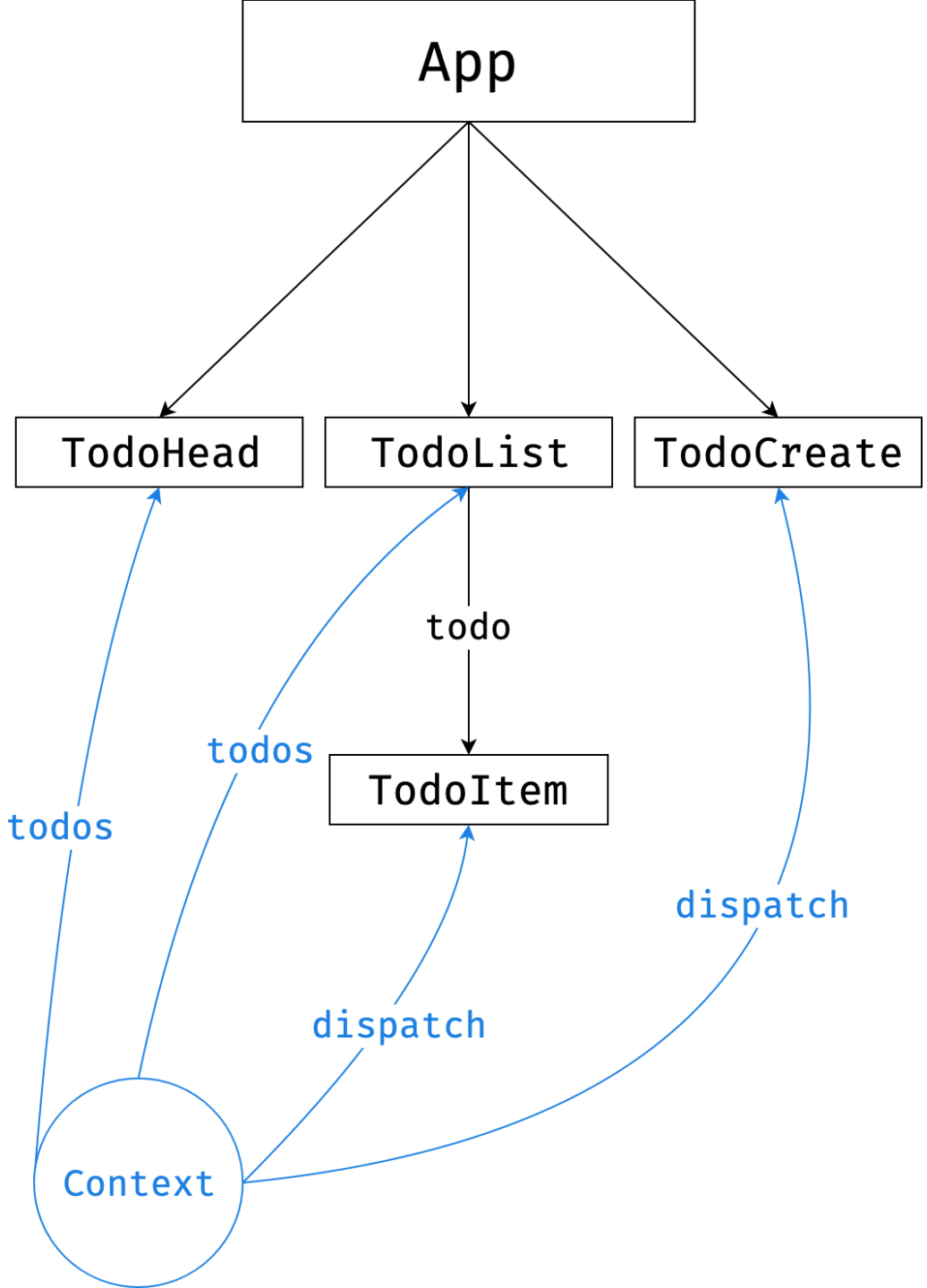
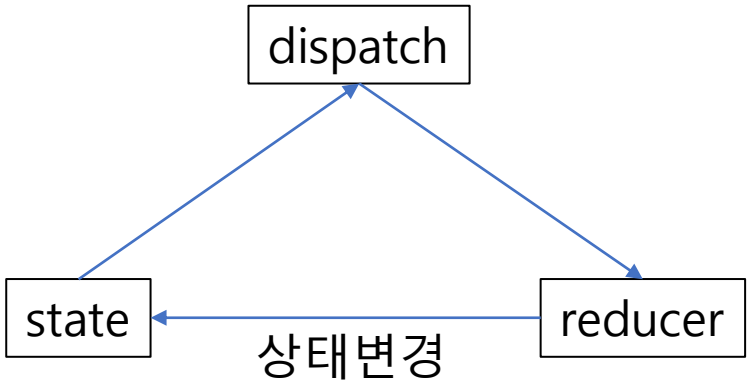
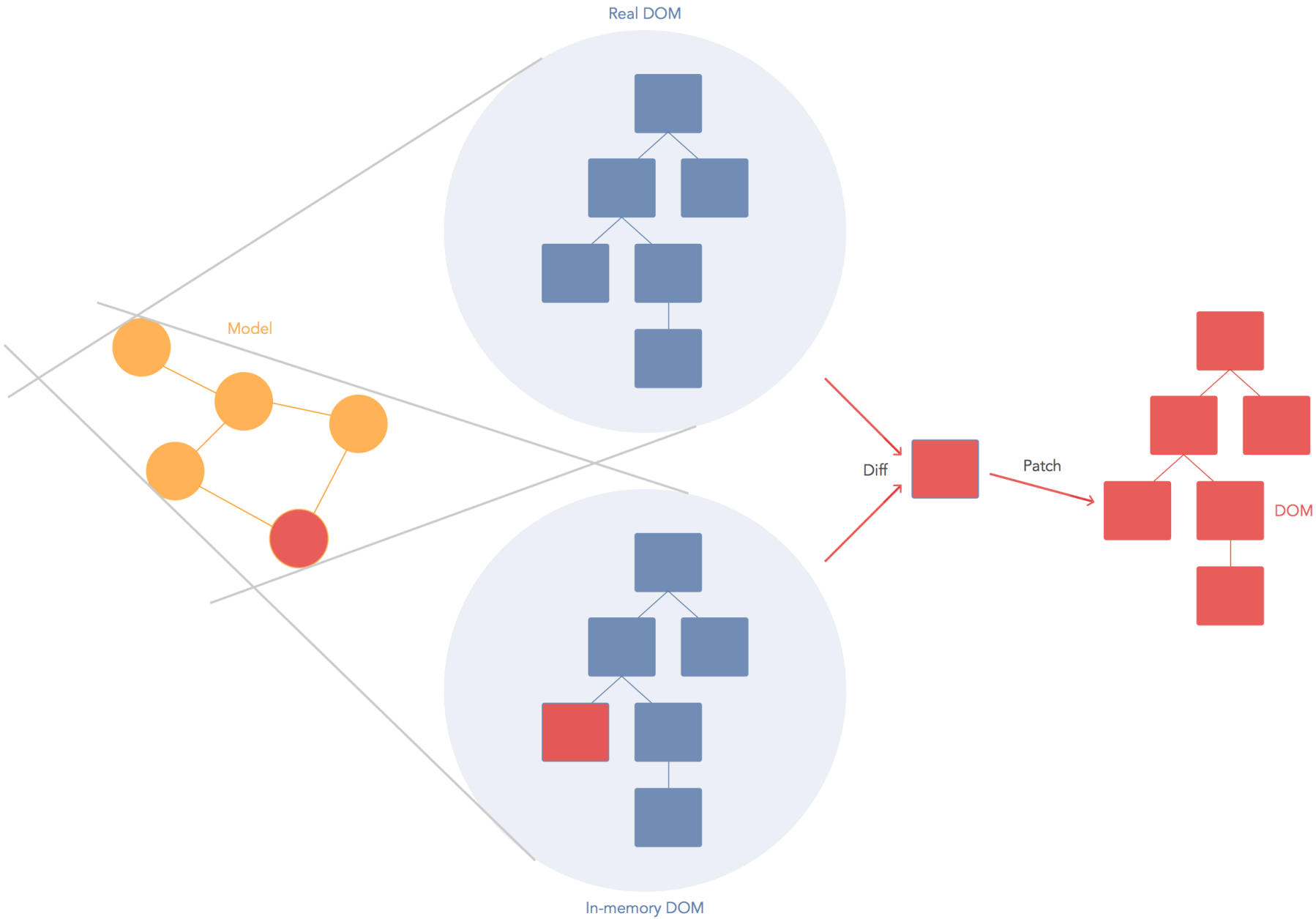

App
todos
onRemove
onToggle
undoneTasks
onCreate
TodoHead
TodoList
TodoCreate
todo
onRemove
onToggle
TodoItem


App
TodoHead
TodoList
TodoCreate
todo
TodoItem
todos
todos
dispatch
dispatch
Context


dispatch
state
reducer
상태변경


Real DOM
Model
Diff
Patch
DOM
In-memory DOM

리덕스는 리액트 생태계에서 가장 사용률이 높은 상태관리 라이브러리입니다.
리덕스를 사용하면 여러분이 만들게 될 컴포넌트들의 상태 관련 로직들을 다른 파일들로 분리시켜서 더욱 효율적으로 관리 할 수 있으며 글로벌 상태 관리도 손쉽게 할 수 있습니다.
우리가 이전에 배운 Context API 를 사용해도 글로벌 상태 관리를 할 수 있고 상태 관리 로직을 분리 할 수 있습니다.
특히, Context API 와 useReducer Hook 을 사용해서 개발하는 흐름은 리덕스를 사용하는 것과 매우 개발 방식이 유사합니다.
리덕스에서도 리듀서와 액션이라는 개념을 사용하거든요.

Redux는 Context API 가 지금의 형태로 사용방식이 개선되기도 전에, 그리고 useReducer 라는 Hook 이 존재하기도 전 부터 만들어진 라이브러리입니다.
사실 Context API 가 개선되기 전에는 프로젝트에서 글로벌 상태관리를 하는게 굉장히 까다로웠어요. 그래서 리덕스가 글로벌 상태 관리 용도로 많이 사용되어 왔었습니다.

리덕스를 여러분의 프로젝트에 써야 할지 말지 고민 할 때에는 다음 사항들을 고려해보세요.

1.프로젝트의 규모가 큰가?
    1. Yes: 리덕스
    2. No: Context API
2.비동기 작업을 자주 하게 되는가?
    1. Yes: 리덕스
    2. No: Context API
3.리덕스를 배워보니까 사용하는게 편한가?
    1. Yes: 리덕스
    2. No: Context API 또는 MobX

리덕스에서 사용되는 키워드

액션 (Action)

```
{
  type: "TOGGLE_VALUE"
}
```

액션 생성함수 (Action Creator)

```
export function addTodo(data) {
  return {
    type: "ADD_TODO",
    data
  };
}

// 화살표 함수로도 만들 수 있습니다.
export const changeInput = text => ({
  type: "CHANGE_INPUT",
  text
});
```

리듀서 (Reducer)

```
function reducer(state, action) {
  // 상태 업데이트 로직
  return alteredState;
}
```

리듀서는, 현재의 상태와, 전달 받은 액션을 참고하여 새로운 상태를 만들어서 반환합니다.
이 리듀서는 useReducer 를 사용할때 작성하는 리듀서와 똑같은 형태를 가지고 있습니다.

```
function counter(state, action) {
  switch (action.type) {
    case 'INCREASE':
      return state + 1;
    case 'DECREASE':
      return state - 1;
    default:
      return state;
  }
}
```

useReducer 에선 일반적으로 default: 부분에 throw new Error('Unhandled Action')과 같이 에러를 발생시키도록 처리하는게
일반적인 반면 리덕스의 리듀서에서는 기존 state를 그대로 반환하도록 작성해야합니다.

리덕스를 사용 할 때에는 여러개의 리듀서를 만들고 이를 합쳐서 루트 리듀서 (Root Reducer)를 만들 수 있습니다.
(루트 리듀서 안의 작은 리듀서들은 서브 리듀서라고 부릅니다.)

스토어 (Store)

리덕스에서는 한 애플리케이션당 하나의 스토어를 만들게 됩니다. 스토어 안에는, 현재의 앱 상태와, 리듀서가 들어가있고, 추가적으로 몇가지 내장 함수들이 있습니다.

디스패치 (dispatch)

디스패치는 스토어의 내장함수 중 하나입니다. 디스패치는 액션을 발생 시키는 것 이라고 이해하시면 됩니다.
dispatch 라는 함수에는 액션을 파라미터로 전달합니다.. dispatch(action) 이런식으로 말이죠.
그렇게 호출을 하면, 스토어는 리듀서 함수를 실행시켜서 해당 액션을 처리하는 로직이 있다면 액션을 참고하여 새로운 상태를 만들어줍니다.

구독 (subscribe)

구독 또한 스토어의 내장함수 중 하나입니다. subscribe 함수는, 함수 형태의 값을 파라미터로 받아옵니다.
subscribe 함수에 특정 함수를 전달해주면, 액션이 디스패치 되었을 때 마다 전달해준 함수가 호출됩니다.

리액트에서 리덕스를 사용하게 될 때 보통 이 함수를 직접 사용하는 일은 별로 없습니다.
그 대신에 react-redux 라는 라이브러리에서 제공하는 connect 함수 또는 useSelector Hook 을 사용하여 리덕스 스토어의 상태에 구독합니다.

리덕스의 3가지 규칙

하나의 애플리케이션 안에는 하나의 스토어가 있습니다

상태는 읽기전용 입니다.

변화를 일으키는 함수, 리듀서는 순수한 함수여야 합니다.