컴포넌트의 상태 업데이트 로직을 컴포넌트에서 분리

# useReduce

# reducer

- state 업데이트가 여러 이벤트 핸들러로 분산되는 경우 관리가 어려워짐
- state 업데이트하는 모든 로직을 reducer를 사용해 컴포넌트 외부로 단일함수로 통합해 관리
- 이벤트 핸들러에서 "action"을 전달
- 작성순서
  - state를 설정하는 것에서 action을 dispatch 함수로 전달하는 것으로 바꾸기.
  - reducer 함수 작성하기.
  - 컴포넌트에서 reducer 사용하기.

# useState에서 useReducer로 바꾸기

- state를 설정하는 것에서 action을 dispatch 함수로 전달하는 것으로 바꾸기.

```
    dispatch({
      type: "what_happend",
      //다른 필드들
     });
```

```
function handleAddTask(text) {
setTasks([
      ...tasks,
      {
        id: nextId++,
        text: text,
        done: false,
      },
    ]);
  }
```

```
function handleAddTask(text) {
   dispatch({
     type: "added",
     id: nextId++,
     text: text,
     done: false,
   });
 }
```

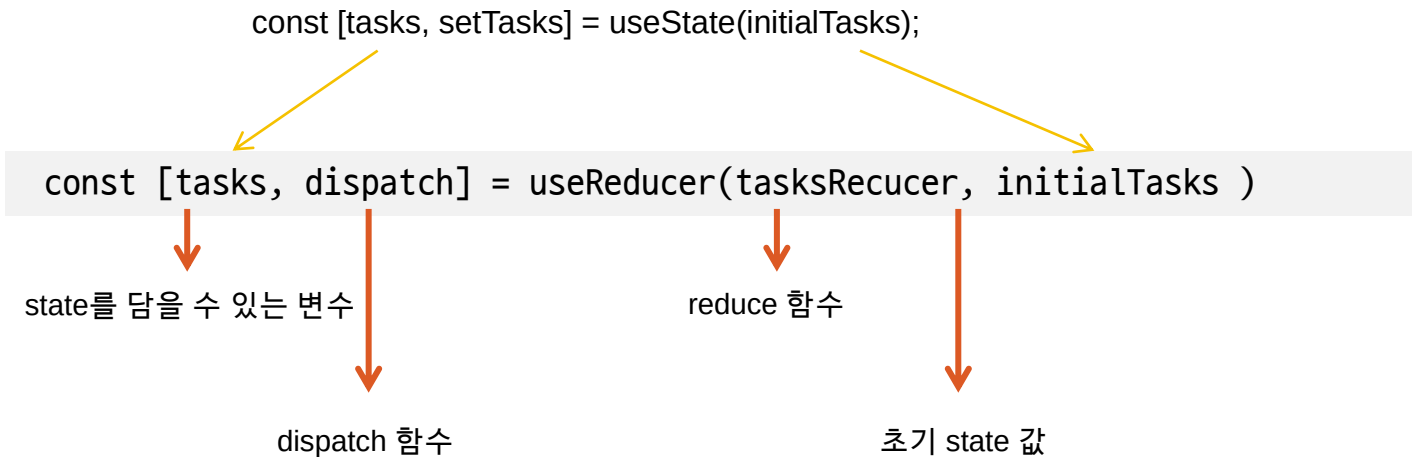
# useState에서 useReducer로 바꾸기

- reducer 함수 작성하기.

```
function tasksRecucer(tasks, action) {
  if (action.type === "added") {
    return [
      ...tasks,
      {
        id: action.id,
        text: action.text,
        done: false,
      },
    ];
  } else if (action.type === "changed") {
    return tasks.map((t) => {
      if (t.id === action.task.id) {
        return action.task;
      } else {
        return t;
      }
    });
  } else if (action.type === "deleted") {
    return tasks.filter((t) =>
                t.id !== action.id);
  } else {
    throw Error("Unknown action:"
                + action.type);
  }
}
```

# useState에서 useReducer로 바꾸기

- 컴포넌트에서 reducer 사용하기.



전역적으로 상태를 공유할 수 있도록 도와주는 도구

# Context API

# Context API

- 전역 데이터를 담고 있는 하나의 저장공간.
- 컴포넌트가 트리 상 아래에 위한 모든 곳에 데이터를 제공.
- Context를 사용하면 명시적으로 props를 전달해주지 않아도 부모 컴포넌트가 트리에 있는 어떤 자식 컴포넌트에서나 정보를 사용할 수 있음.
- Context 예시
  - 테마 지정하기
  - 로그인된 사용자 정보
  - 라우팅
  - 상태관리

# Context API

- 컨텍스트 생성
  - createContext() 함수의 인자로 초기값을 넘김

```
const themeDefault = { border : '10px solid green'}
const ThemeContext = createContext(themeDefault)
```

- 컨텍스트 사용

```
const themeContext = useContext(ThemeContext)
```

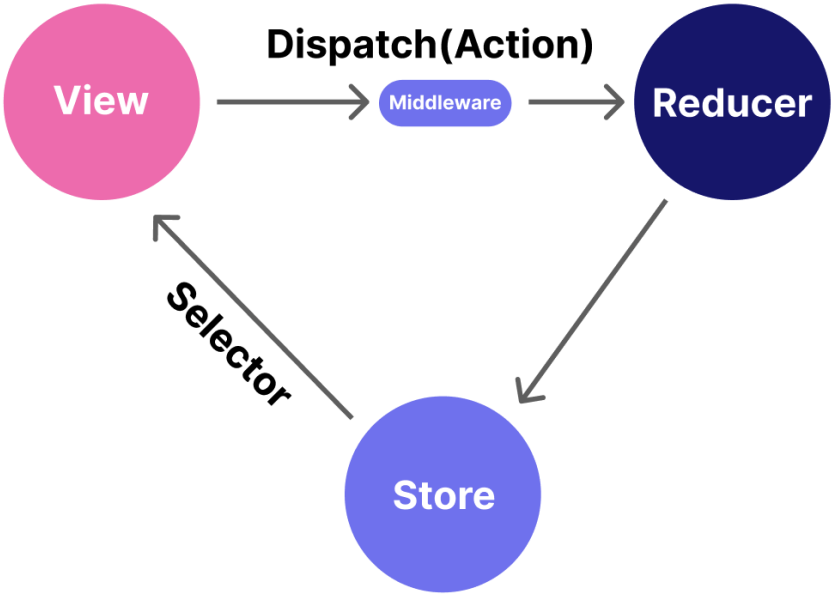
- 컨텍스트 제공

```
< ThemeContext.Provider value={themeDefault}>
앨리먼트
</ThemeContext.Provider>
```

상태관리자

# react-redux

# Redux란



# react-redux 적용

- 리액트 리덕스 설치

```
npm i redux react-redux
```

- reducer 생성

```
function reducer(currentState, action) {
  …
  return newState;
}
```

//state와 action을 인수로 받아서
// action에 따라 새로운 state를 생성해서 리턴

- store 생성하고 reducer 주입

```
import { legacy_createStore as createStore } from "redux";
const store = createStore(reducer)
```

//store의 인수로 reducer를 넣어줌

# react-redux 적용

- Provider로 감싸고 store 지정

```
import { Provider, useSelect, useDispatch, connect } from 'react-redux'
<Provider store={store}>                                    //컴포넌트를 Provider 로 래핑
```

- useSelect

```
const number = useSelector( (state) => state.number );      //store에 저장된 state 꺼내기
```

- useDispatch

```
const dispatch = useDispatch();

dispatch( { type:'PLUS' } )                                 //dispatch를 호출하여 state 값 변경
```

효율적인 **Redux** 개발을 위한 도구모음

# Redux Toolkit

# Redux Toolkit 이란

- 효율적인 Redux 개발을 위한 도구모음
- 저장소준비, 리듀서 정의 , 불변 업데이트 로직, 액션 생산자나 액션 타입을 직접 작성하지 않고 전체 상태, 'slice'을 만들어내는 기능에 해당하는 유틸리티 함수들이 제공
- Redux Toolkit이 나온 배경
  - 저장소를 설정하는 것이 너무 복잡하다"
  - "쓸만하게 되려면 너무 많은 패키지들을 더 설치해야 한다"
  - "보일러플레이트 코드를 너무 많이 필요로 한다"
- 기능별로 작은 스토어(slice)를 여러 개 만들면 이 slice를 합쳐서 리덕스가 요구하는 큰 스토어를 리덕스 툴킷이 알아서 만들어 줌

https://ko.redux.js.org/redux-toolkit/overview/

# Redux Toolkit 사용

- 리덕스 툴킷이 세팅된 개발 환경으로 app 만들기

```
>npx create-react-app example07_redux --template redux

Creating a new React app in D:\react_work\example07_redux.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template-redux...
```

# Redux Toolkit 사용

- createSlice
  - 초기값과 reduce 지정

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

```
const initialState = {   value: 0,   status: 'idle',  };
```

```
export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {       state.value += 1;     },
    decrement: (state) => {       state.value -= 1;     },
    incrementByAmount: (state, action) => {
      state.value += action.payload;
    },
  },
});

export const selectCount = (state) => state.counter.value;
```

# Redux Toolkit 사용

- 스토어 생성
  - 작은 슬라이스들을 모음

store.js

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

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
});
```

# Redux Toolkit 사용

- state 사용하기

```
import { useSelector, useDispatch } from 'react-redux';
import {  decrement,  increment,  incrementByAmount,  incrementAsync,
  incrementIfOdd,  selectCount, } from './counterSlice';
```

```
export function Counter() {
  const count = useSelector(selectCount);
  const dispatch = useDispatch();

  const incrementValue = Number(incrementAmount) || 0;

  return (
    <div>
      <div className={styles.row}>
        <button
          className={styles.button}
          aria-label="Decrement value"
          onClick={() => dispatch(decrement())}
        >
          -
        </button>
```

```
dispatch( { type:'counterSlice/up', step:2 } )
```

```
dispatch( countSlice.action.up(2) )
```