React

Redux

state 종류

- Local State: 각각의 컴포넌트가 소유하고 있는 상태를 의미. 이 상태는 해당 컴포넌트 내에서만 관리되고 사용됨
- Cross-Component State: 두 개 이상의 컴포넌트 간에 공유되는 상태를 의미하며 props를 통해 상태를 전달
- App-Wide State: 애플리케이션의 전체 영역에서 사용되는 상태. 여러 컴포넌트, 혹은 앱의 전체 영역에서 공유되어야 하는 데이터나 상태에 사용됨

=> Cross-Component와 App-Wide State 일 때
***Context API*나 *Redux*가 요구됨**

Context API

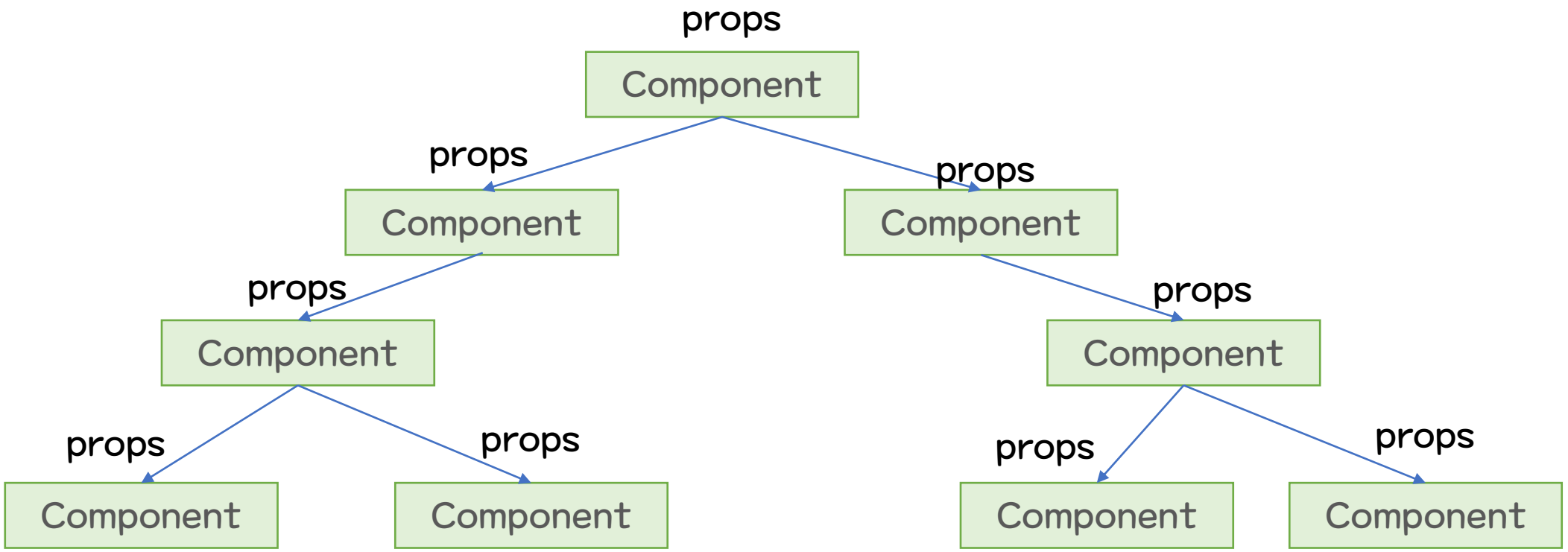
(정의만 참고하기!)

Context란

- 컴포넌트 트리 전체에 걸쳐 데이터를 공유하는 방법
- 일반적으로 React에서 데이터를 전달하려면 부모에서 자식으로 props를 통해 전달. => 자식이 많아지면 코드가 복잡해진다!

- Context API는 React의 내장 기능
- Context API를 사용하면 중간 컴포넌트들을 건너뛰고 데이터를 직접 전달할 수 있다

사용이유



컴포넌트 수가 많은 대형 프로젝트에서는 props를 많이 써야하는 경우가 발생

Redux in JS

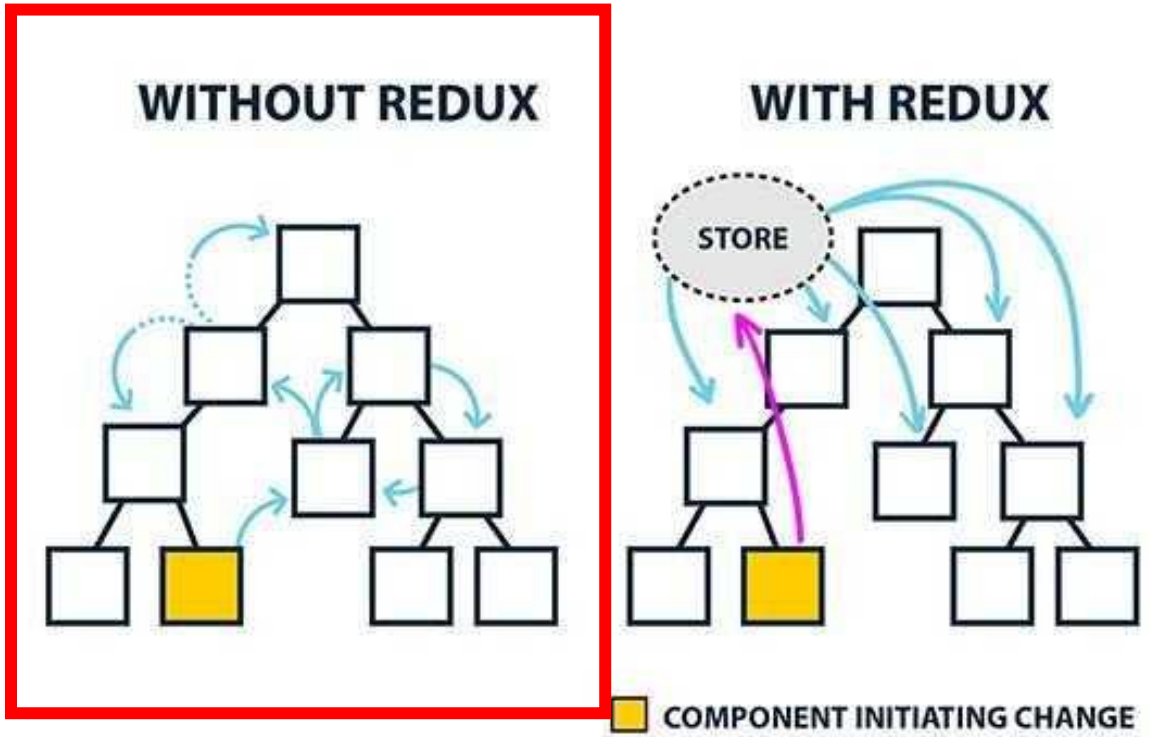
redux란?

- Javascript 상태관리 라이브러리
- 리액트를 배울 때 많이 나오는 용어지만, 꼭 리액트에 종속되는 개념은 아님
- 리액트의 상태 관리 라이브러리로 가장 많이 사용됨.

redux 사용 이유

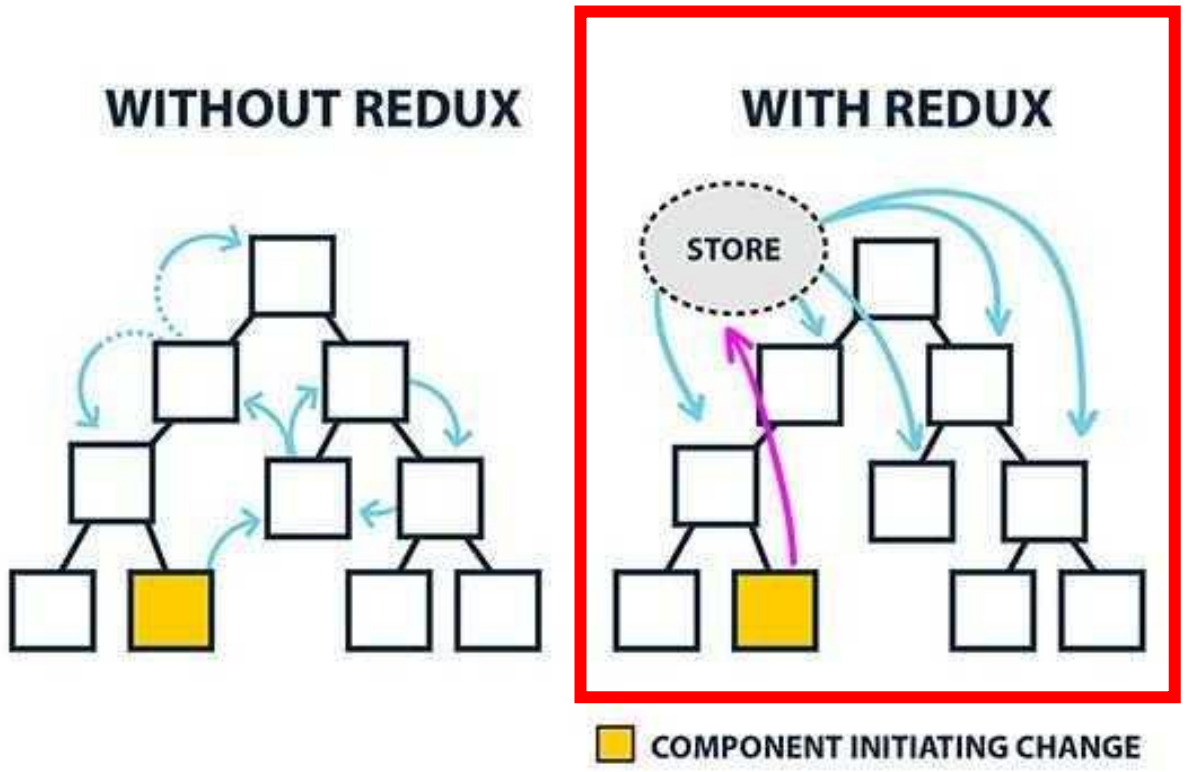
- 컴포넌트 수가 많은 대형 프로젝트에서는 state를 전해주기 위해 props를 엄청나게 많이 써야 하는 경우가 발생 함.

→ props 지옥…?



redux 사용 이유

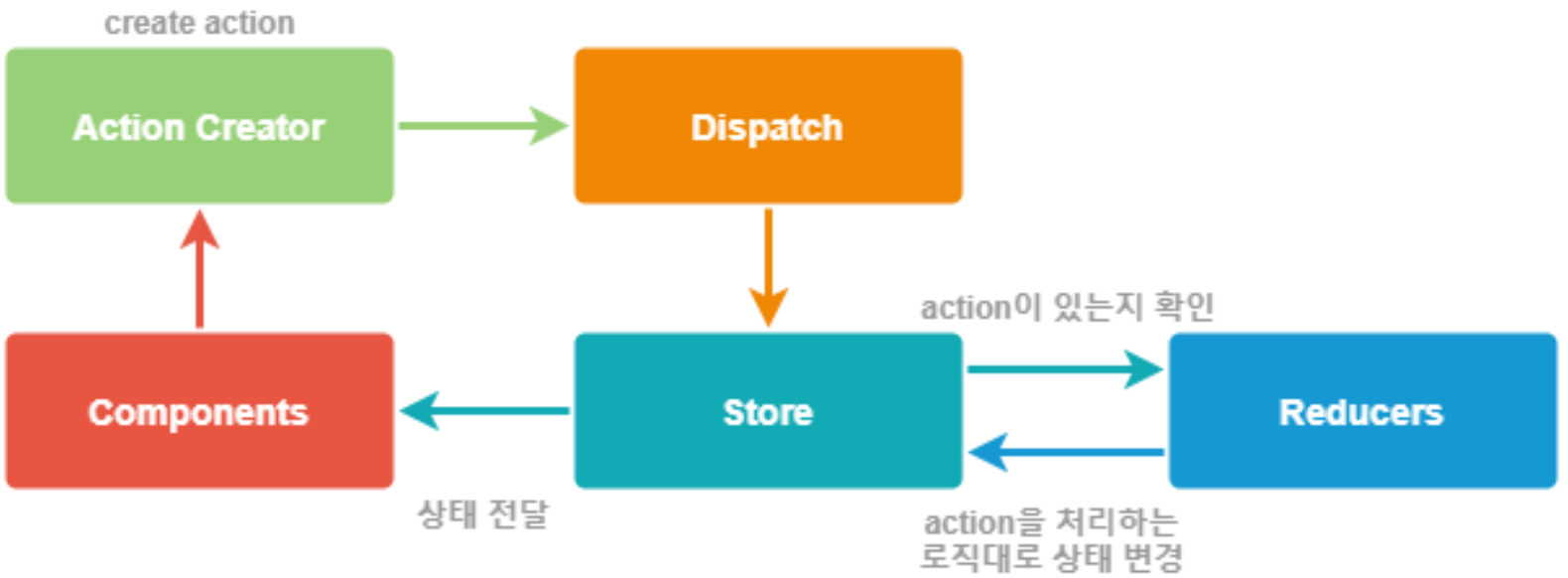
- 리덕스를 사용할 경우, 전역으로 상태를 관리할 수 있게 되어 state를 props로 전달.. 전달.. 전달.. 하지 않고, store라는 곳에서 언제든지 꺼내 쓸 수 있음!



redux 용어 정리

- 리덕스를 사용하기 위해 알아야 하는 용어는?

- Store (스토어)
- Action (액션)
- Reducer (리듀서)
- Dispatch (디스패치)



Store (스토어)

- Store는 상태가 관리되는 오직 하나의 공간
- 스토어 안에는 현재 애플리케이션 상태와 리듀서가 들어가 있음.
- 한 개의 프로젝트는 단 하나의 스토어만 가질 수 있음.
- 스토어에 있는 데이터는 컴포넌트에서 직접 조작하지 않음
  => 리듀서 함수 사용

Action (액션)

- 상태에 어떠한 변화가 필요하면 액션(action)이란 것이 발생
- Action은 컴포넌트에서 store에 운반할 데이터를 말함.
- Action은 하나의 **객체**로 표현됨.
- 리듀서가 수행할 작업을 설명

```
{
  type: 'CHANGE_INPUT',
  text: '안녕하세요'
}
```

```
{
  type:'INCREASE',
}
```

Dispatch (디스패치)

- 액션을 발생시키는 것
- dispatch(action) 과 같은 형태로 액션 객체를 파라미터로 넣어서 호출

Reducer (리듀서)

- 리듀서는 액션의 type에 따라 변화를 일으키는 함수
- 첫번째 매개 변수는 현재 상태값, 두번째 매개 변수는 Action값을 받음

```
const initialState = {
  counter: 1
};

function reducer(state=initialState, action){
  switch(action.type){
    case 'INCREMENT':
      return {
        counter: state.counter + 1
      };
    default:
      return state;
  }
}
```

- 항상 새로운 상태 객체를 반환
- Http 요청, 데이터 저장 같은 건 하면 안됨

Redux 사용

```
import { createStore } from "redux";
//리듀서는 데이터를 수정해주는 함수
//나의 데이터를 변경해줌
const reducer = (count = 0, action) => {
    switch (action.type) {
        case ADD:
            return count + 1;
        case MINUS:
            return count - 1;
        default:
            return count;
    }
};
//스토어 생성
const countStore = createStore(reducer);
console.log(countStore);
```

npm install redux

createStore() : store생성

Redux 사용

```
//getState()는 createStore로 생성된 저장소에서 사용되는 메소드
//최신상태의 값을 반환
const onChange = () => {
    number.innerText = countStore.getState();
};

//subscribe는 함수를 사용하며 데이터와 저장소가 변경될 때마다 함수를 실행
countStore.subscribe(onChange);
```

```
countStore.dispatch({ type: "ADD" });
countStore.dispatch({ type: "MINUS" });
```

dispatch() : 액션 발생

React Redux

redux 적용

- 모듈 설치하기

```
npm install redux react-redux @reduxjs/toolkit
```

@reduxjs/toolkit?

- Redux의 복잡성을 줄이기 위해 만들어진 도구
- 액션 생성, 리듀서, 미들웨어 등 Redux와 관련된 기능들을 효율적으로 구현

redux 적용

- 리덕스를 리액트에 적용하기

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

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
    <React.StrictMode>
        <Provider store={store}>
            <App />
        </Provider>
    </React.StrictMode>
);
```

redux 적용

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

```
const number = useSelector(state => state.number);
```

- useSelector
  - 리덕스 store의 상태 값을 조회하기 위한 hook 함수
  - 인자로 함수를 넘겨줘야 함
  - 그 함수는 state를 매개변수로 받을 수 있고, return 값은 원하는 state 변수 값 설정

redux 적용

```
const dispatch = useDispatch();
<button onClick={()=>{dispatch({type:'INCREASE'})}}>+1</button>
<button onClick={()=>{dispatch({type:'DECREASE'})}}>-1</button>
```

- useDispatch
  - Action을 발생시키는 dispatch 함수를 실행하는 hook 함수
  - 인자로 원하는 Action 객체를 넘겨줘야 함.

Redux Toolkit

redux toolkit

codingOn

configureStore()

- Redux 스토어를 생성하기 위한 함수.
- 여러 미들웨어와 리듀서를 쉽게 통합할 수 있으며, Redux DevTools 확장 프로그램과의 통합도 제공

createSlice()

- 리듀서와 액션을 함께 생성하는 함수
- 슬라이스라는 개념을 사용하여 액션 타입, 액션 생성 함수, 리듀서를 한 번에 정의

redux toolkit

```
const counterSlice = createSlice({
    name: "counter",
    initialState: initialCounterState,
    reducers: {
        increment(state) {
            state.counter++;
        },
        decrement(state) {
            state.counter--;
        },
        increase(state, action) {
            state.counter = state.counter + action.payload;
        },
    },
});
```

redux toolkit

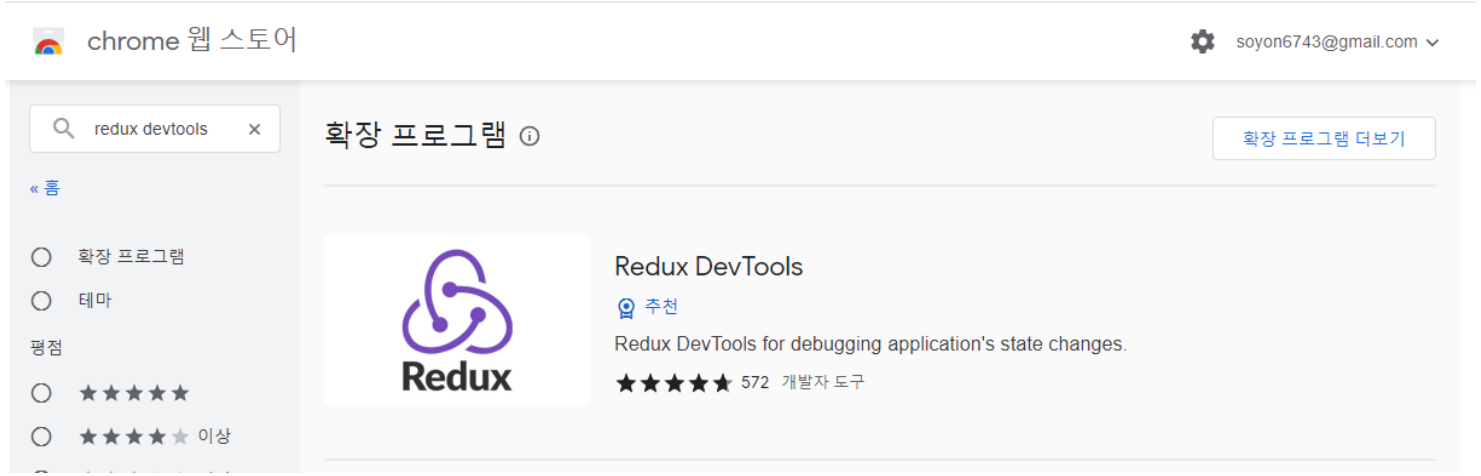
```
const store = configureStore({
    reducer: { counter: counterSlice.reducer, auth: authSlice.reducer },
});

export const counterActions = counterSlice.actions;
export const authActions = authSlice.actions;

export default store;
```

Chrome 에서 스토어 확인

- 크롬 확장 프로그램 Redux DevTools 설치



Chrome 에서 스토어 확인

codingOn

- 리액트 앱에 적용하기

```
npm install redux-devtools-extension
```

```
import { composeWithDevTools } from "redux-devtools-extension";
```

```
const store = configureStore({reducer: reducer}, composeWithDevTools());
```

Chrome 에서 스토어 확인

- 크롬 개발자 도구 〉 Redux 로 스토어에 저장된 상태 확인해보기