# Redux 소개 및 현 시스템 적용 방안 (Context API)

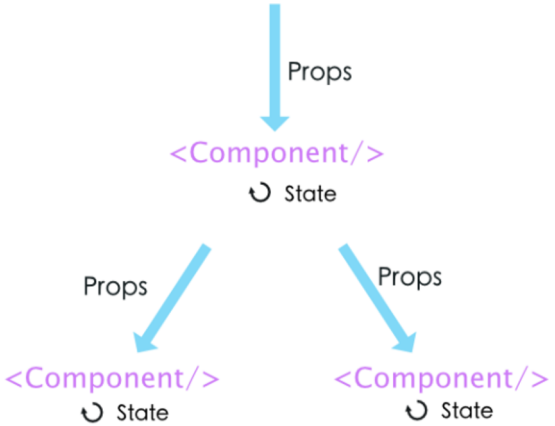
# 리덕스 등장 배경

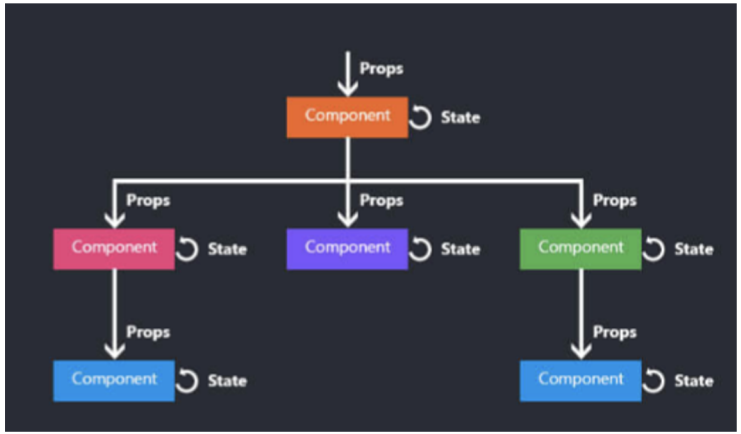
I. 너무 많은 상태 관리

II. 복잡한 데이터 흐름

III. 복잡한 코드

IV. 변화와 비동기의 혼용



# 리덕스

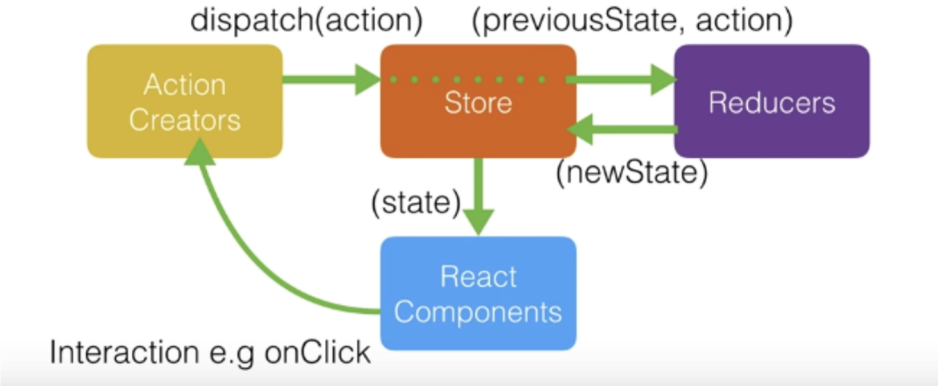
현재 가장 많이 사용되는 리액트 상태 관리 라이브러리

I. 컴포넌트의 상태 업데이트 관련 로직을 다른 파일로 분리시켜 더욱 효율적 관리 가능

II. 똑같은 상태를 공유해야 할때 여러 상태 컴포넌트를 거치지 않고 전달 및 업데이트 가능

III. 코드 유지 보수성과 작업 효율을 극대화

IV. 비동기 작업을 효과적으로 관리 가능



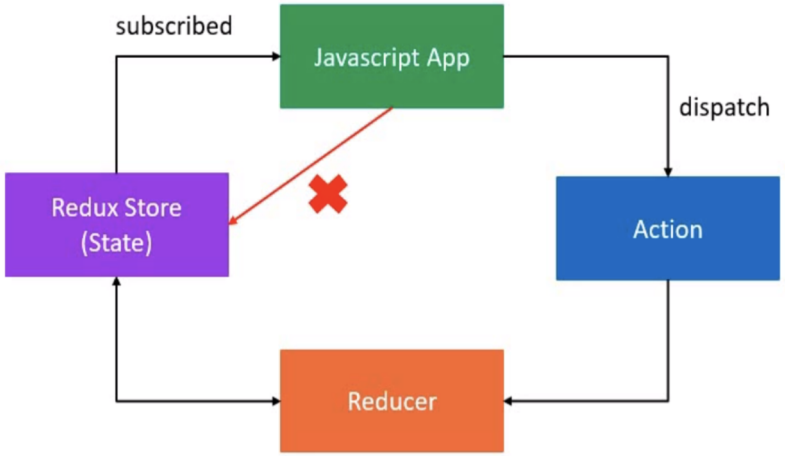
# 리덕스 작동 원리

I. Redux Store는 중앙에서 State를 관리하고 있고 오직 Reducer에 의해서만 변경됨

II. Redux에서 State에 어떤 변화를 일으켜야 할 때 Action을 Store에 요청(Dispatch)

III. Action은 객체 형태로 되어있고 상태를 변화 시킬 때 그 객체를 Reducer가 참조하여 변경

IV. Redux Store를 Subscribe하고 있는 Component 들은 변경된 State를 State를 Store에서 가져옴

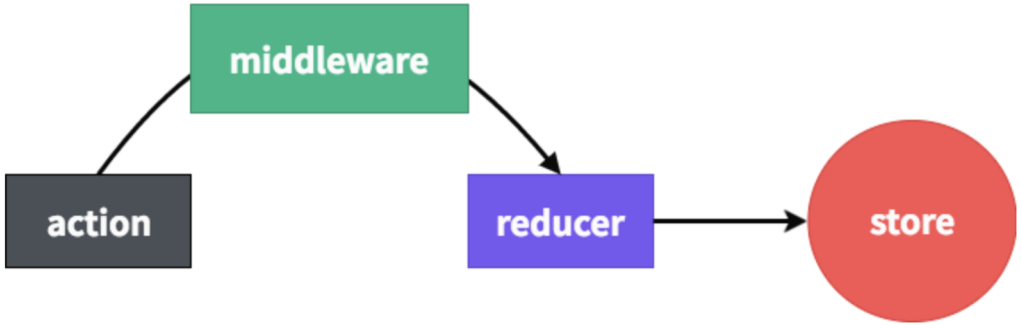


# 리덕스 미들웨어

액션과 리듀서 사이의 중간자

- 액션이 디스패치(dispatch) 되어서 리듀서에서 이를 처리하기전에 사전에 지정된 작업들을 설정

ex) 전달받은 액션에 기반하여 액션을 취소, 다른 종류의 액션들을 추가적으로 디스패치



# 리덕스를 사용해야 하는 이유

I. 데이터의 집중화로 인한 예측 가능

II. 단방향적 데이터 흐름을 통한 용이한 디버깅

III. 유연한 구현이 가능
- prop 전달이 매우 많은 경우나 디버깅 문제가 있다면 Redux 사용을 권장

IV. 미들웨어를 통한 비동기 작업 처리 및 state 관리 안정성 확보

# 현재 시스템에 도입하려는 목적

동일한 데이터를 props, state로 관리해야 하는 경우가 많기 때문

i. state를 전역으로 사용하고자 하는 목적

ii. 로그인 관련 uuid 또한 리덕스로 관리

검색 결과

i. 단순 state를 전역으로 하기 위한 거면 context API를 사용할 것
   - 리덕스보다 개발 과정이 더 간편함 ( 리덕스가 더 많은 기능을 제공 )

ii. 로그인 관련 uuid는 로컬스토리지에 저장하는 것이 좋음

# Context API

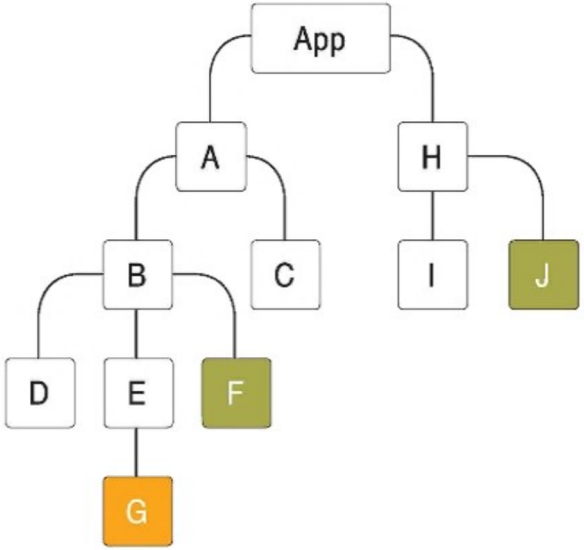
I. 리덕스 자체가 context API를 기반으로 개발됨

II. 단순 전역 state를 관리하는 목적

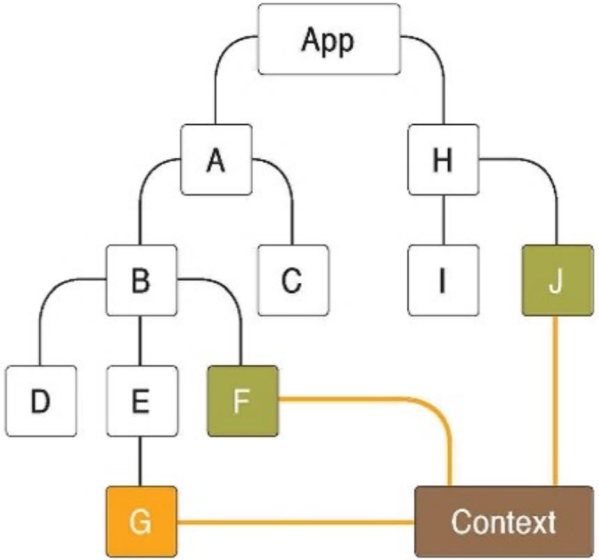
- 본 목적만 있을 시 코드 구현 단계에서 Context API 가 더욱 편리함

# Context API

기존 데이터 흐름

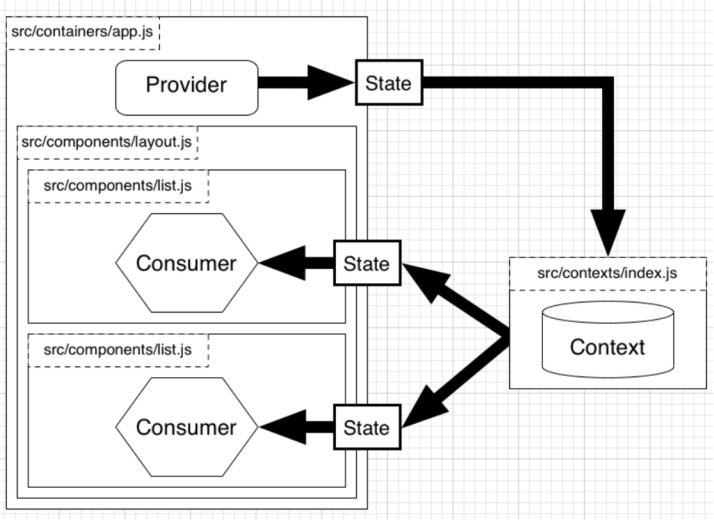



Context API 적용 시 흐름



# Context API 구성

클래스형 컴포넌트 사용할 경우




함수형 컴포넌트 사용할 경우

hooks 라이브러리 제공 (useContext)

더욱 간결해지는 코드

유지보수에 용이

Consumer 구현이 쉬워진 것

# 현 시스템에 도입 예정

email, url 등 여러 컴포넌트에 걸쳐서 이동하는 경우 도입