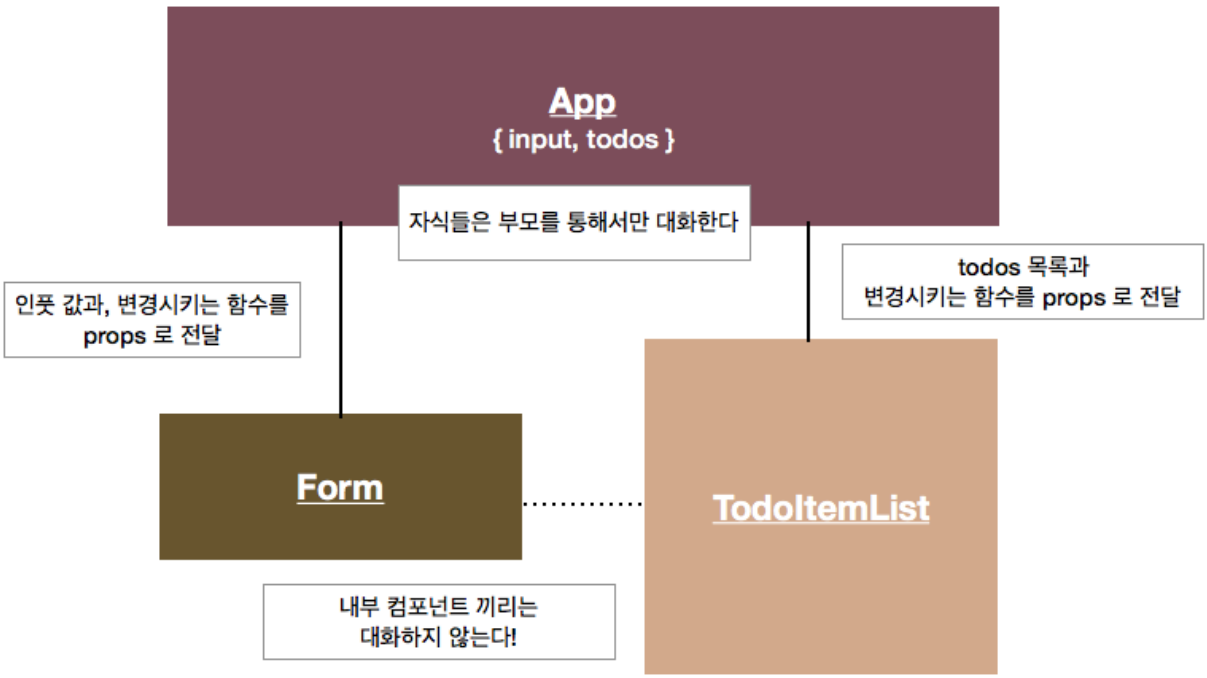
Redux

들어가기 전에

Redux는 가장 유명한 JavaScript 상태 관리 라이브러리입니다. React에서만 쓰려고 만들어진 것이 아니기에 JavaScript 어디에서든 사용할 수 있지만 가장 대표적으로 React와 쓰이고 있으며 저희가 React 를 기본 라이브러리로 잡고 있기 때문에 React 기반으로 설명드리겠습니다

React의 상태 관리

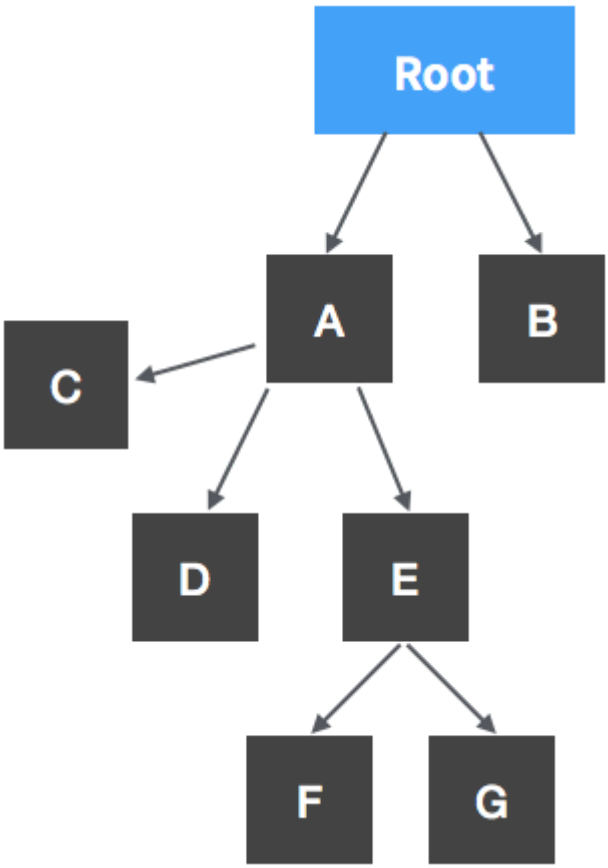
Redux를 알기 전에 React의 상태 관리 방법에 대해 알아보겠습니다.



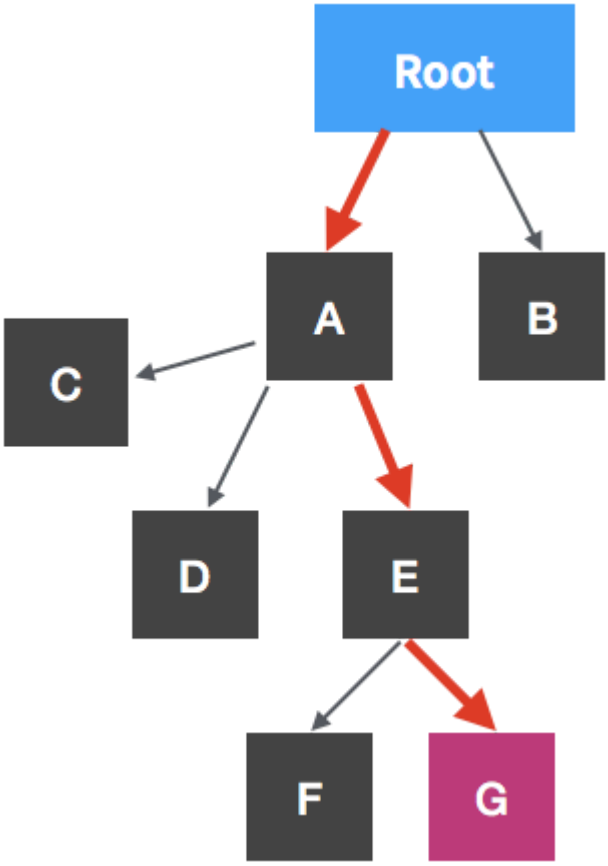
리액트 프로젝트에서는 대부분의 작업을 할 때 부모 컴포넌트가 중간자 역할을 합니다. 컴포넌트끼리 직접 소통하는 방법도 있지만 그렇게 하면 코드가 굉장히 꼬이기 때문에 절대 권장되지 않습니다. 예를 들어 TodoList 를 만들었다고 생각해보겠습니다. 부모 컴포넌트인 App

컴포넌트에서 input, todos 라는 변수와 input 값을 변경시킬 onChange, todo 아이템을 만들어 낼 onCreate 함수를 생성해 각 자식들에게 주는 방식으로 투두리스트가 만들어질 것입니다.

이러한 구조는 부모 컴포넌트에서 모든걸 관리하기 때문에 매우 직관적이며, 오히려 규모가 작으면 작을수록 관리하는 것도 꽤 편하게 느껴집니다. 하지만 규모가 커진다면 App 컴포넌트의 코드가 넘칠 정도로 길어질 수밖에 없고 이에 따라 유지보수도 매우 힘들어 집니다. 예를 들어 아래와 같은 프로젝트 구조가 있다고 생각해봅시다.



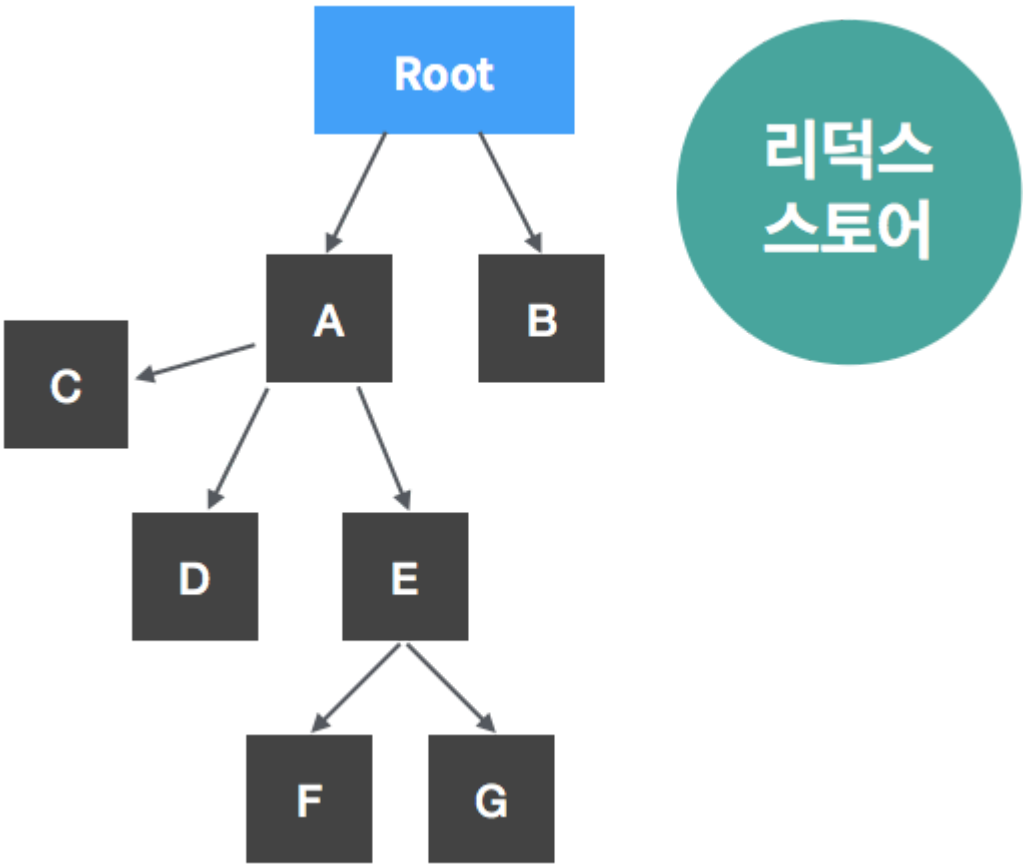
Root 컴포넌트에서 G 컴포넌트에게 어떠한 값을 전달해야 한다면 아래와 같이 이를 잇는 모든 컴포넌트를 거쳐야 합니다.



Redux!!

그럼 이제 리덕스로 넘어와 보겠습니다. 위에서 봤다시피 리액트의 state 관리는 상당히 힘든 구조를 가지고 있습니다. 하지만 Redux를 이용해 전역으로 상태를 관리한다면 정말 편해집니다.

리덕스에서 가장 중요한 개념은 **Store** 라고 생각합니다. 리덕스에서 스토어는 어느 컴포넌트에도 종속되지 않고 독립적인 구조를 가지고 있습니다. 아래 사진을 보면 바로 이해가 되실겁니다.



각 컴포넌트들은 스토어를 구독(subscribe)하고 있습니다. 유튜브 구독이랑 똑같습니다. 만약 G 컴포넌트에서 스토어의 count 라는 변수를 사용하고 있다면 다른 컴포넌트에서 count 변수를 업데이트 해도 G 컴포넌트에서 그대로 업데이트 된 값을 사용할 수 있게 됩니다.

그리고 여기서 스토어의 state를 업데이트 하는 것을 **dispatch** 라고 합니다. dispatch 시에는 반드시 action 이라는 객체를 스토어에게 던져주는데 이 객체 안에는 필수 값으로 **type**이 들어갑니다. 즉, 아래와 같은 구조를 띄게 되는 것입니다.

```
let action = { type: 'INCREMENT', ... }
dispatch(action)
```

type을 제외하고는 선택값이며 보통 변수값이 들어가고, 이 선택값을 payload 라고 부릅니다.

Reducer를 통해 상태 변화시키기

dispatch를 했으면 리덕스는 Reducer를 통해 상태를 업데이트 합니다. action 안에는 반드시 type 값이 있어야 한다고 했습니다. 그렇기 때문에 Reducer는 type에 따라 어떻게 상태를 업데이트 할 지 정의하고, 또한 상태를 업데이트 하는 함수입니다.

예를 들어 action: { type: 'INCREMENT', payload: 2 } 라고 하면 특정 변수에 2를 더해주는 작업을 수행하게 되겠죠. 이렇게 업데이트 로직을 정해주는 함수를 리듀서라고 합니다.

Reducer는 **state, action** 이라는 2가지 파라미터를 받습니다. action은 위에서 설명드린 action 그대로고, state는 **현재 상태**를 의미합니다. 즉, action에 따라 현재 state를 변경하면 되는거죠. 그리고 그렇게 특정 변수가 업데이트 되면 해당 변수를 사용하고 있던 컴포넌트는 자동으로 리렌더링을 하게 되는 것입니다.

Redux의 3가지 규칙

1. 하나의 어플리케이션 안에는 하나의 스토어가 있습니다.
2. 상태는 읽기전용 입니다.
    - 리액트에서 state를 업데이트 할 때는 setState를 사용하고, 배열을 업데이트 할 때는 배열 자체에 push를 하지 않고, concat 같은 함수를 사용해야 합니다. 리덕스 또한 마찬가지입니다. 기존의 상태를 건들이지 않고 새로운 상태를 생성하여 업데이트를 해주는 방식으로 해야합니다.
    - 리덕스에서 불변성을 유지해야 하는 이유는 내부적으로 데이터가 변경 되는 것을 감지하여 shallow equality 검사를 하기 때문입니다. 리덕스는 내부적으로 shallow equality 검사를 계속 진행하는데 이를 사용해 컴포넌트가 올바르게 업데이트 되려면 불변성이 필수입니다. 실제로 리덕스 공식 홈페이지에서 아래와 같은 문구를 확인할 수 있습니다.

> Redux's use of shallow equality checking requires immutability if any connected components are to be updated correctly.

3. 변화를 일으키는 함수, 리듀서는 순수한 함수여야 합니다.

단점

이렇게 좋아보이는데… 요즘 추세는 Redux를 지양하는 쪽으로 가고 있습니다. 물론 여전히 대부분의 기업에서 Redux를 쓰고 있고 있지만 유명한 기업들에서 Redux 보단 다른 상태관리 라이브러리들을 사용하기 시작했죠.

Redux는 위에서 설명드린 대로 상태 관리를 위한 라이브러리 입니다. 그리고 여기서 맹점은 **상태 관리만을 위한 라이브러리** 라는 것입니다. 상태 관리를 위해서만 만들어졌으며 프레임워크가 아닌 라이브러리이기 때문에 필요한 다른 기능들이 있을 경우 또 다른 라이브러리들을 가져와 사용해야 합니다. 예를 들어, 서버 데이터를 사용하기 위해서는 Redux-saga, Redux-thunk, Redux-toolkit 등과 같은 또 다른 미들웨어를 필수적으로 사용해야 하죠. 그리고 이렇게 미들웨어가 많아질수록 코드의 길이가 길어지게 됩니다.

즉, Redux를 이용해 서버 데이터를 관리하게 되면 **서버 데이터를 위한 로직이 너무 커지고, 그로 인해 Redux를 활용하기 위한 보일러 플레이트가 비대해 진다는 것입니다.**

그리고 이런 문제를 해결하기 위해 대체 기술들을 여러 기업들이 찾아보았고, 한때는 swr 이 React-Query 보다 앞서가기 시작했으나 현재는 뒤집힌 상황이 되었고 앞으로도 격차가 커지지 않을까 됩니다.