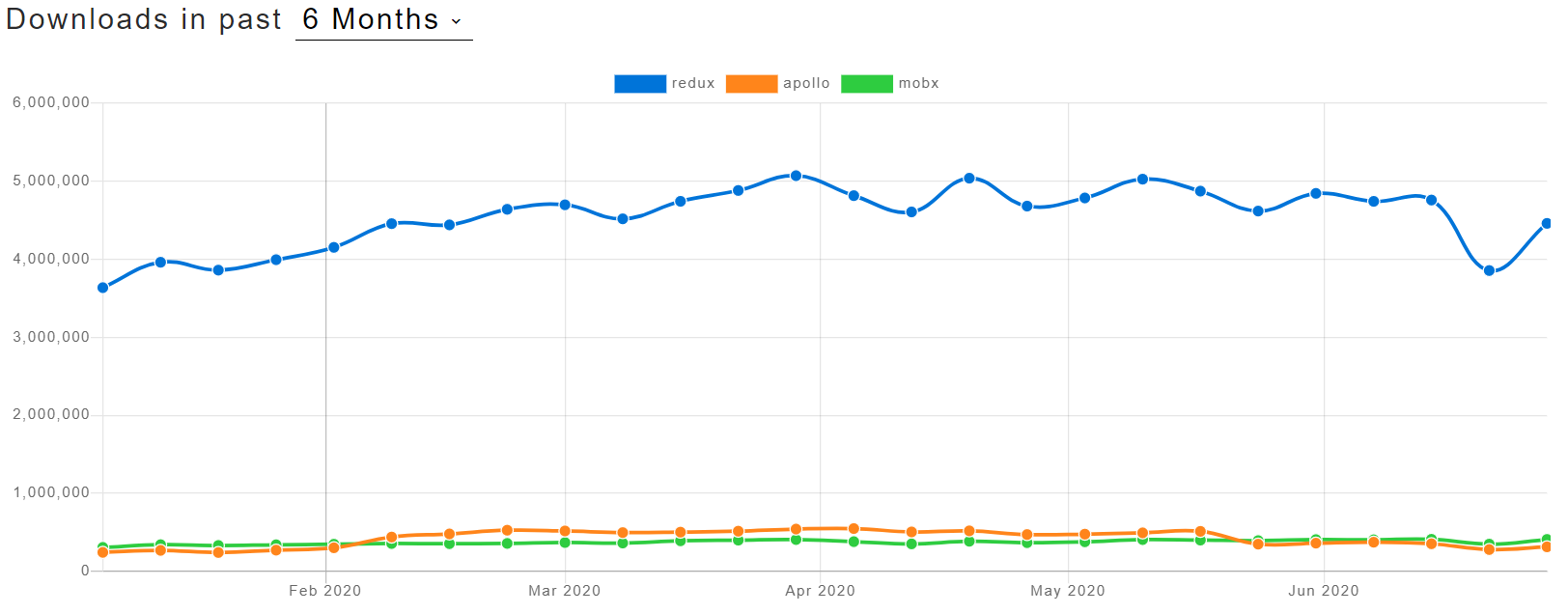
1. **리덕스의 개념은 무엇이고 왜 쓰는가?**

리덕스는 현재 가장 사용률이 높은 상태관리 라이브러리이다.



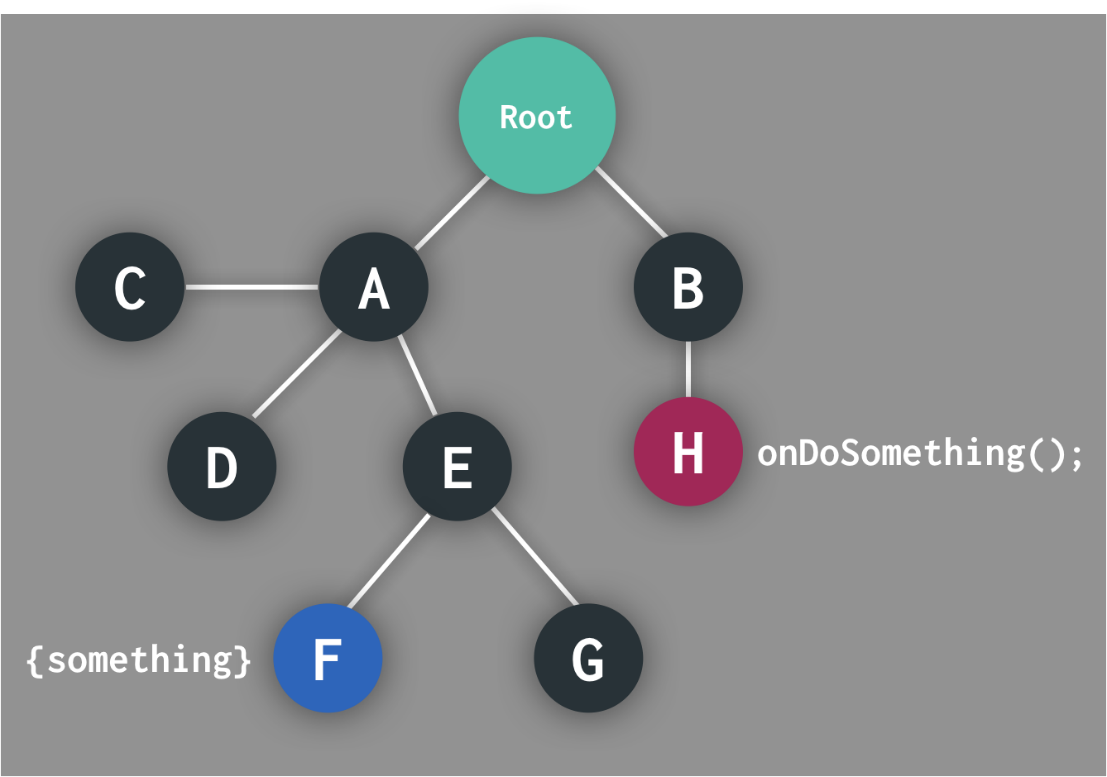
**출처 :** <https://www.npmtrends.com/redux-vs-apollo-vs-mobx>

리덕스를 왜 사용하는가 ?

리덕스를 사용하면, 우리가 만들게 될 컴포넌트들의 상태 관련 로직들을 다른 파일들로 분리시켜서 더욱 효율적으로 관리할 수 있다. 또한, 컴포넌트끼리 상태를 공유하게 될 때 여러 컴포넌트를 거치지 않고도 손쉽게 상태 값을 전달할 수 있다.

추가적으로, 리덕스의 미들웨어라는 기능을 통하여 비동기 작업, 로깅 등의 확정적인 작업들을 더욱 쉽게 할 수도 있게 해준다.

예시를 통해서 리덕스의 필요성을 몸소 느껴보자.

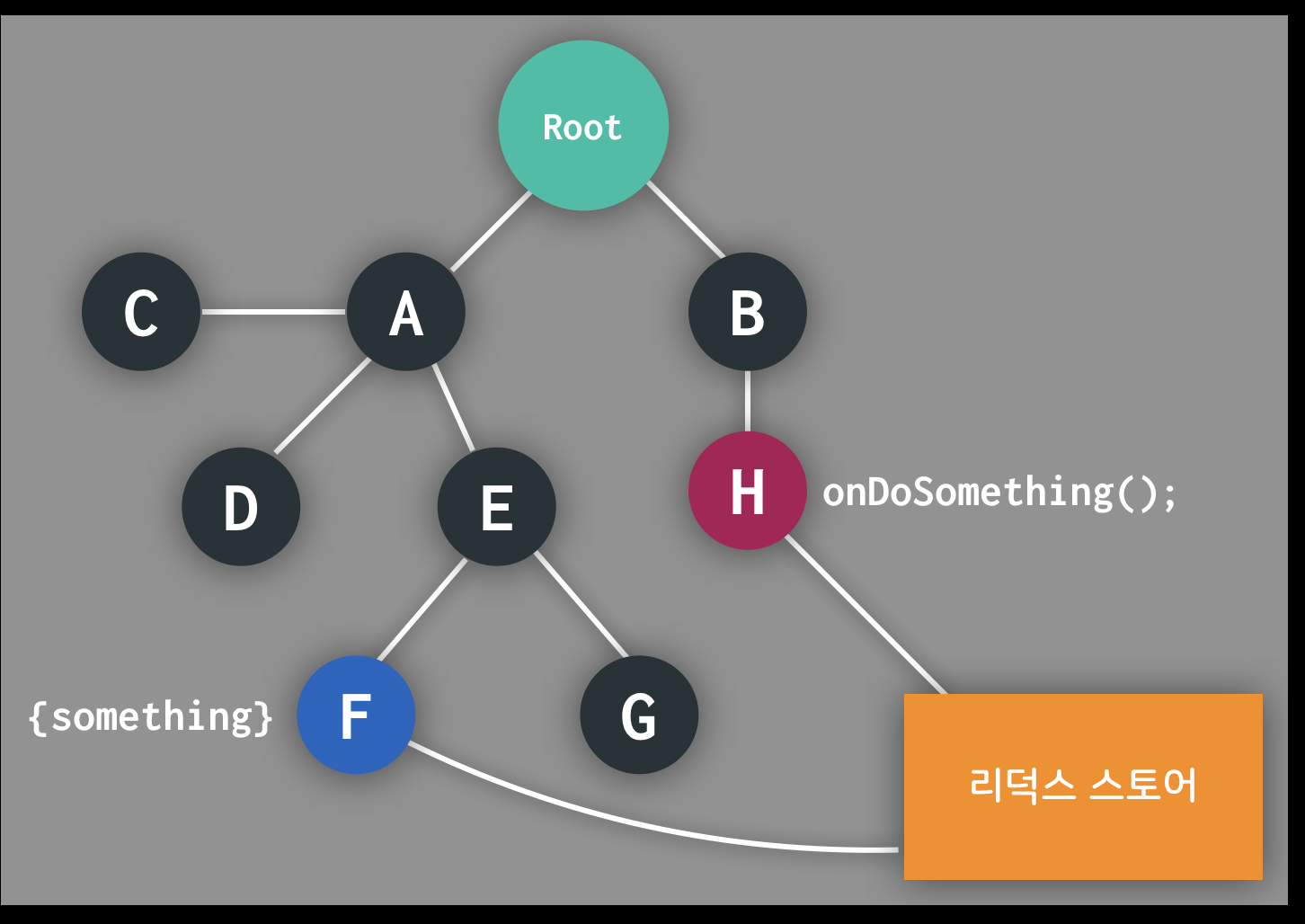


Root 컴포넌트에는 something 이라는 상태 값이 있고, onDoSomething이라는 함수가 something 값에 변화를 준다.

onDoSomething은 Root -> B -> H 로 전달되고, H 에서 이벤트가 발생하여 이 함수가 호출되면 something 이 Root -> A -> E -> F 로 전달된다.

props 가 필요한 곳으로 제대로 전달되게 하기 위하여, 실제로는 해당 props 를 사용하지 않는 컴포넌트를 거쳐가야 한다는 것은 리렌더링하게 될 때 비효율적이기도 하고, 굉장히 귀찮은 작업이기도 하다. 상위 컴포넌트에서 props 이름을 바꿔준다면 그 아래에도 쭉 바꿔줘야 하기 때문이다.

**리덕스가 있다면,** 다음과 같은 구조로 작업을 진행할 수 있게 된다.



앱이 지니고 있는 상태와, 상태 변화 로직이 들어있는 스토어를 통하여,

우리가 원하는 컴포넌트에 원하는 상태값과 함수를 직접 주입해줄 수 있게 된다.

이런 식으로, 더 쉬운 글로벌 상태 관리를 위하여 리덕스를 사용하기도 하며,

조금 더 체계적이고 편리한 상태 관리를 하기 위하여 사용을 하는데, 후자의 경우엔 실제로 사용을 해봐야 경험을 해볼 수 있을 것이다.

출처 : <https://velog.io/@velopert/Redux-1-%EC%86%8C%EA%B0%9C-%EB%B0%8F-%EA%B0%9C%EB%85%90%EC%A0%95%EB%A6%AC-zxjlta8ywt>

1. **hook 과 mobx는 무엇이고 왜 쓰는가 ?**

**Hook** : 함수(function) 컴포넌트에서 Class형 컴포넌트처럼 <state>와 <생명주기 기능(lifecycle features)>을 연동(hook into)할 수 있게 해주는 함수

배경 : 컴포넌트 간에 데이터 공유 시, 편의성을 위해 만들었다.

리액트 Data는 한 방향으로 흐른다.(one-way data flow).

ex) 부모 컴포넌트 -> 자식 컴포넌트로 props(properties: 클래스의 속성 값) 전달한다.

객체(object)의 상태값(state value)은 props{객체}에 담아 자식 컴포넌트에 전달한다.

여러 컴포넌트 간에 상태 값을 공유하는 경우, 데이터 전달을 위해

* 컴포넌트 구조를 바꿔야 하거나,
* 컴포넌트의 생명주기 (lifecycle) 설정을 별도로 해주거나
* “래퍼 지옥(wrapper hell)”을 볼 수 있음. (=분리하기 어려운 BigSize 컴포넌트)
* 때문에 컴포넌트들을 작게 만드는 것은 불가능하고, 테스트하기도 어렵다.

이러한 문제를 해결하고자 Hooks를 개발함.

목적 : Hook는 props, state, context, refs, 그리고 lifecycle와 같은 React 개념에 좀 더 직관적인 API를 제공.

출처 : <https://developthreefeet.tistory.com/1>

**Mobx** : 프론트엔드를 위한 어플리케이션 상태 관리 라이브러리이다. 주로 React와 함께 사용되며, 상태 관리 라이브러리라는 특성 때문에 종종 Redux와 비교된다.

Redux가 함수형 아이디어를 적용했다면, Mobx는 반응형 아이디어를 적용했다. Mobx에서는 반응형 프로그래밍 패러다임에서 필수적으로 언급되는 옵저버 패턴을 적극적으로 사용한다. 다만, Mobx에서는 오직 상태(데이터)만 관찰 가능하며, 상태가 변경되었을 때 반응한다.

한마디로, Mobx는 상태를 observable하게 관리할 수 있도록 돕는 라이브러리이다. 모든 상태변화로 일어나는 부분을 자동으로 추적한다.

**Mobx와 React를 함께 사용하는 이유**

React는 State의 변경사항을 기반으로 최소한의 UI를 업데이트하며 Mobx는 절대적으로 필요한 경우에만 state를 변경한다. 따라서 최소한의 state 변경과 최소한의 UI 업데이트는 프로젝트를 빠르고 효율적인 사용자 인터페이스를 제공할 수 있게 한다.

출처 : <https://blog.naver.com/PostView.nhn?blogId=httts&logNo=221741842238>

**3. webpack과 babel은 무엇이고 왜 쓰는가**

자바스크립트 코드가 많아지면 하나의 파일로 관리하는데 한계가 있다. 그렇다고 여러 개 파일을 브라우저에서 로딩하는 것은 그만큼 네트워크 비용을 치뤄야 하는 단점이 있다. 뿐만 아니라 각 파일은 서로의 스코프를 침범하지 않아야 하는데 잘못 작성할 경우 변수 충돌의 위험성도 있다.

함수 스코프를 사용하는 자바스크립트는 즉시호출함수(IIFE)를 사용해 모듈을 만들 수 있다. CommonJS나 AMD 스타일의 모듈 시스템을 사용하면 파일별로 모듈을 관리할 수도 있다. 그러나 여전히 브라우저에서는 파일 단위 모듈 시스템을 사용하는 것은 쉽지 않은 일이다. 모듈을 IIFE 스타일로 변경해주는 과정뿐만 아니라 하나의 파일로 묶어(bundled) 네트워크 비용을 최소화할 수 있는 방법이 웹 프론트엔드 개발 과정에는 필요하다. 따라서 WebPack이 등장하였다.

WebPack은 프로젝트의 구조를 분석하고 자바스크립트 모듈을 비롯한 관련 리소스들을 찾은 다음 이를 브라우저에서 이용할 수 있는 번들로 묶고 패킹하는 모듈 번들러(Module bundler)이다.

출처 : <http://jeonghwan-kim.github.io/js/2017/05/15/webpack.html>

Babel은 자바스크립트 컴파일러로서, 최신의 자바스크립트 코드를 아주 무난한 이전 단계의 자바스크립트 코드로 변환 가능하게 해주는 개발 도구, 즉 트랜스파일러(Transpiler)이다.

이와 같은 트랜스파일러(Transpiler)가 필요한 이유는, ES6+를 사용하여 프로젝트를 진행하려면 ES6+로 작성된 코드를 IE를 포함하여 모든 브라우저에서 문제 없이 작동할 만한 개발환경을 구축해야 하기 때문이다. 즉, Babel 패키지를 활용하면 자바스크립트 최신 문법으로 자유롭게 코딩하면, 그것과 웹브라우저 간의 호환을 Babel 패키지가 책임져준다.

출처 : <https://brunch.co.kr/@topherlee/29>