**1. 리덕스의 개념은 무엇이고 왜 쓰는가**

**“Redux is a predictable state container for JavaScript apps.”**

리덕스는 ‘자바스크립트 앱을 예측 가능한 상태 컨테이너’ 라고 공식 Repo에서 말하고 있습니다.말 그대로 리액트 뿐 아니라 순수 자바스크립트의 상태관리에도 리덕스를 이용할 수 있습니다. 리덕스가 리액트와의 조합이 좋은 이유는 리덕스가 리액트의 작동원리중 하나인 상태변화를 쉽게 예측할 수 있도록 도와주기 때문입니다.

모던 자바스크립트 싱글페이지 어플리케이션(Single Page Application)의 관리해야 할 상태들이 더 많아지고 복잡해지고 있습니다. 이 상태들에는 서버와의 통신 뿐만 아니라 로컬환경에서 생성되는 상태들 또한 포함됩니다. 그리고 최근에는 UI를 관리하는 상태들의 복잡성이 크게 증가했습니다. 이러한 복잡성은 변화와 비동기성 두가지 개념의 조합을 이해하기 힘들기 때문에 발생합니다. 리액트와 같은 프론트엔드 라이브러리나 프레임워크는 비동기 및 직접 DOM 조작을 모두 제거하여 뷰 레이어에서 이 문제를 해결하려고 합니다. 하지만 상태를 관리하고 데이터를 조작하는 것은 여전히 개발자들의 몫입니다. 여기서 리덕스가 유용합니다.

리덕스는 세 가지의 기본 원칙으로 설명할 수 있습니다.

**Single source of truth**

우리의 앱 전체상태는 단일스토어로 관리합니다. 단일스토어는 앱 전체를 쉽게 관리할 수 있는 범용성을 가지고, 또한 디버깅이나 개발자도구를 쉽게 사용할 수 있게 합니다.

**State is read-only**

상태는 ‘읽기 전용’ 이며, 상태를 변경할 수 있는 유일한 방법은 ‘액션 객체’를 이용하여 ‘액션’을 발생시키는 것입니다.

**Changes are made with pure functions**

리덕스의 리듀서는 순수 함수이어야 합니다. 여기서 순수함수란 어떤 함수에 동일한 파라미터를 주었을때 항상 같은 값을 리턴하고, 외부의 상태를 변경하지 않는 함수입니다. 리듀서는 이전의 상태와 액션을 받아서 다음 상태를 반환하는 순수한 함수입니다. 한개의 리듀서부터 시작해서 여러개의 리듀서들까지 한개의 상태트리에서 관리하게 됩니다.

출처 : <https://byseop.netlify.app/redux-01/>

2. hook 과 mobx는 무엇이고 왜 쓰는가

**Motivation**

**1. 컴포넌트 사이에서 상태와 관련된 로직을 재 사용하기 어렵다.**

React는 컴포넌트에 재사용 가능한 행동을 붙이는 방법을 제공하지 않는다(예를 들어, 스토어에 컴포넌트를 연결하는 것). 이전에는 이것을 해결하기 위해 render props 그리고 고차 컴포넌트와 같은 패턴에 익숙할 것이다. 그러나 이런 패턴을 사용할 때 컴포넌트를 재구성해야 하며 코드를 추적하기 어렵게 만든다. React 개발자 도구에서 전형적인 React 애플리케이션을 본다면, providers, consumers, HoC, render props 등 래퍼 지옥을 볼 가능성이 높다. React는 상태 관련 로직을 공유하기 위해 좀 더 좋은 기초 요소가 필요하다.

Hook를 사용하면 컴포넌트로 부터 상태 관련 로직을 추상화 할 수 있다. 이것은 독립적인 테스트와 재사용이 가능하다. Hook는 계층 변화 없이 상태 관련 로직을 재 사용할 수 있도록 한다. 이것은 많은 컴포넌트들이 공유하기 쉬워지는 역할을 한다.

**2. 복잡한 컴포넌트들은 이해하기 어렵다.**

우리는 때로 간단하게 시작했지만 유지하기 힘든 상태 관련 로직들과 사이드 이펙트가 있는 컴포넌트들을 유지해야 한다. 각 생명주기 메서드는 자주 관련없는 로직이 섞여있다. 예를 들어 componentDidMount, componentDidUpdate로 데이터를 가져오는 것을 수행할 수 있으나 같은 componentDidMount 메서드라도 이벤트 리스너를 설정하는 것과 같은 관계없는 일부 로직이 포함될 수 있으며, componentWillUnmount에서 cleanup을 수행하기도 한다. 함께 변경되는 상호 관련 코드는 분리되지만 이와 관련없는 코드들은 단일 메서드로 결합한다. 이로 인해 버그가 쉽게 발생한다.

위와 같은 사례 안에서 상태 관련 로직이 모든 공간에 있기 때문에 이러한 컴포넌트들을 작게 만드는 것은 불가능하다. 또한 테스트도 어렵다. 이것이 많은 사람이 React를 별도 상태 관리 라이브러리를 사용하는 이유이다. 그러나 상태 관리 라이브러리는 종종 너무 많은 추상화를 하고, 다른 파일들 사이에서 건너뛰기를 요구하기 때문에 컴포넌트 재 사용을 더욱 어렵게 한다.

이것을 해결하기 위해, 생명주기 메서드를 기반으로 쪼개는 데 초점을 맞추기 보단, Hook를 통해 로직을 기반을 둔 작은 함수로 컴포넌트를 나눌 수 있다(구독 설정 및 데이터를 불러오는 것과 같은 로직) 조금 더 예측 가능하도록 하기 위해 리듀서를 활용해 컴포넌트의 지역 상태 값을 관리하도록 할 수 있다.

**3. Class는 사람과 기계를 혼동 시킨다.**

Class는 코드의 재사용성과 코드 구성을 좀 더 어렵게 만들고 React를 배우는데 큰 진입 장벽이다. Javascript에서 대부분의 언어가 다르게 작동하는 this가 어떻게 작동하는지 알아야 한다. 또 이벤트 핸들러가 등록되는 방법을 기억해야 한다. 여전히 React안에서 함수와 Class 컴포넌트들을 구별하고 각 요소를 언제 사용하는지는 숙련된 React 개발자 사이에서도 의견이 일치하지 않는다.

Class 컴포넌트가 최적화를 더 느린 경로로 되돌리는 의도하지 않은 패턴을 장려할 수 있다는 것을 발견했다. Class 는 최근 사용되는 도구에도 많은 문제를 일으킨다. 예를 들어 Class는 잘 축소되지 않고, 핫 리로딩을 깨지기 쉽고 신뢰할 수 없게 만든다. 우리는 코드가 최적화 가능한 경로에서 유지될 가능성이 더 높은 API를 제공하고 싶다.

**4. 개념적으로 React는 컴포넌트는 항상 함수에 더 가깝다.**

**React Hooks의 등장**

이런 문제를 해결하기 위해서 Facebook에서는 2018년 10월 ReactConf 2018에서 Hooks이라는 새로운 방법을 제안되었습니다.

**Hook이란?**

Hook은 함수형 컴포넌트에서도 클래스형 컴포넌트의 기능을 사용할 수 있게 하는 기능입니다. 즉 기존에 함수형 컴포넌트에서 못하던상태값 관리, 컴포넌트의 생명 주기 함수를 이용할 수 있게 되었다.

**Hooks 처리시 장점**

함수형 컴포넌트에서는 원하는 기능을 함수로 만든 후 필요한 곳에 넣어주기만 하면 되기 때문에 로직의 재사용이 유연해 집니다.

클래스형 컴포넌트가 가지고 있던 복잡성, 재사용성의 단점들까지 해결됩니다.

출처 : <https://medium.com/@ehddnjs8989/react-hooks-%EC%82%AC%EC%9A%A9%EC%9D%B4%EC%9C%A0-ce03c66a53b0>

**Mobx이란?**

Redux와 또 다른 State관리 라이브러리이며 이글을 작성하는 목적의 라이브러리입니다. 기본적으로 객체지향 느낌이 강하며 Component와 State를 연결하는(Redux와 달리) 번잡한 보일러플레이트 코드들을 데코레이터(애노테이션)제공으로 깔끔하게 해결합니다.

**Observable**

Mobx에서 Rerendering 대상이 되는 state(상태, 값)를 관찰 대상(observable value)라고 칭하며 @observable 데코레이터로 지정한 State는 관찰대상으로 지정되고 그 State는 값이 변경될 때 마다 Rerendering됩니다. 이것이 사실 Mobx가 동작하는 가장 기본 개념입니다.

**Mobx의 장점**

**객체지향적**

보다 객체지향적입니다 ES6에서 추가된 Class를 이름뿐인 Class가 아니라 객체지향적으로 사용하고 개발하는 것을 권장하고 있습니다. 도메인모델로 분리됨으로 써 집중된 비즈니스 로직은 적절히 분산되고 도메인간의 상호작용은 message를 주고 받는 형태로 구현 할 수 있습니다.

**서버개발자들에게 친숙한 아키텍쳐**

Java Spring Framework와 유사한 아키텍쳐구조를 지향하고 있어 서버개발자들에게 보다 친숙하고 낮은 러닝커브를 제공, 장점을 그대로 적용할 수 있습니다.

**Decorator**

데코레이터(java 애노테이션과 유사하다고 보면 된다)를 제공하기 때문에 Redux를 사용할 때 React Component와 state를 연결 하기위한 mapStateToProps, Redux action을 연결을 위한 mapDispatchToProps 그리고 bindActionCreators…. 등등의 보일러플레이트 코드가 사라지고 데코레이터가 처리하기 때문에 너무나도 깔끔한 코드가 생성됩니다.

**캡슐화**

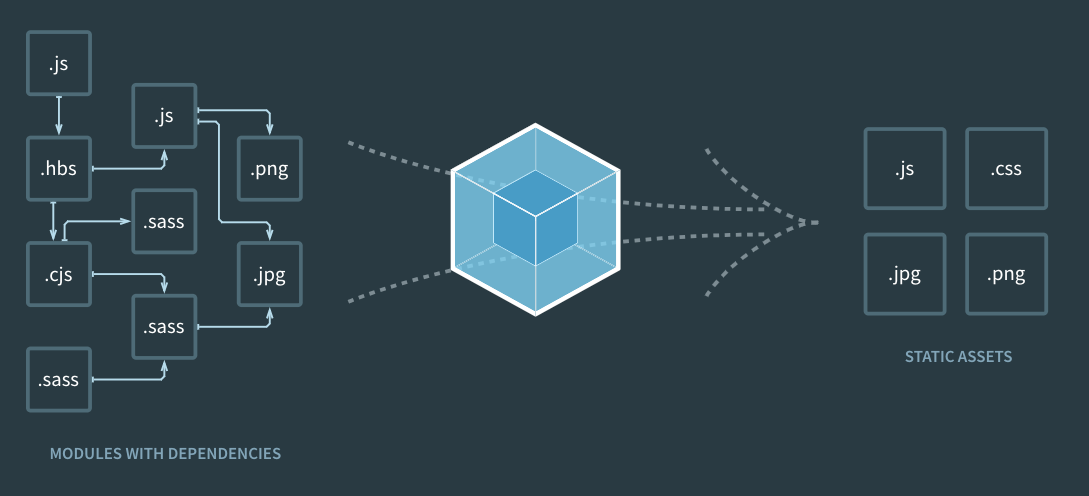
Mobx Configuration 설정으로 State를 오직 메소드를 통하여 변경할 수 있도록 Private하게 관리 할 수 있습니다.

**불변성 유지를 위한 노력이 불필요**

State의 불변성을 유지하기 위해서 번잡스러운 코드나 ImmutableJs같은 라이브러리를 따로 사용할 필요가 없습니다. 이것이 왜 장점이 되냐 하면 불변성을 유지하면 서 State를 변경하는 코드는 Object가 Depth가 깊어지게 되면 코드의 가독성이 매우 떨어집니다. 그래서 ImmutableJs 라이브러리를 사용하게 되는데 Redux와 같이 사용하게 될 경우 여러가지 설정이 필요하고 추가적인 라이브러리도 필요할 뿐 만 아니라 추가적인 학습도 동반 되어야 합니다.

출처 : <https://woowabros.github.io/experience/2019/01/02/kimcj-react-mobx.html>

3. webpack과 babel은 무엇이고 왜 쓰는가

**왜 Webpack이 등장하게 되었을까?**

규모가 있는 System에서는 많은 Javascript가 존재하게 됩니다. 이 많은 Javascript 파일을 하나의 파일로 관리하기엔 어려움이 있습니다.

그렇다면 여러 개로 호출하면 해결되는게 아닐까?

여러 개의 파일을 브라우져에서 로딩하는 것은 네트워크 비용이 그만큼 사용하여 반응속도가 느려지게 됩니다. 더 나아가 각 파일의 변수 충돌의 위험성도 존재하게 됩니다. 이를 해결하기 위해 Webpack이 등장하게 되었습니다.

**Webpack이란?**

현대 Javascript Application의 Static Module Bundler입니다. Webpack이 실행된다면 Dependencies Graph를 통해 필요한 형태의 하나 또는 여러개의 Bundle로 생성합니다. 번들링을 함으로써 파일은 하나로 합쳐지고 네트워킹 요청횟수는 줄어들게 됩니다. 그리고 중복된 소스코드도 최소화하고 모듈 개념을 사용하기 때문에 글로벌이 오염되지도 않습니다.

그리고 웹팩을 사용하면 자바스크립트 간의 종속성 뿐만 아니라 자바스크립트 내에서 필요한 css나 img와 같은 파일(.css, .jpg)도 번들링해서 하나의 파일로 합쳐줘서 네트워크 요청을 최소화 시킬 수 있습니다.

**Bundle이란?**

소프트웨어 및 일부 하드웨어와 함께 작동하는 데 필요한 모든 것을 포함하는 Package  
각각의 모듈들에 대해 의존성 관계를 파악하여 하나 또는 여러개의 그룹으로 볼 수 있습니다. 다른 Module Bundler도 많이 존재하지만 Webpack이 performance가 우수합니다.

출처 : <https://nesoy.github.io/articles/2019-02/Webpack>

**Babel이란?**

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 웹브라우저간의 호환을 Babel 패키지가 책임져준다.

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