09. REDUX Prior Art.

Redux has a mixed heritage. It is similar to some patterns and technologies, but is also different from them in important ways. We'll explore some of the similarities and the differences below.

9.1. Flux

Redux was inspired by several important qualities of Flux. Like Flux, Redux prescribes that you concentrate your model update logic in a certain layer of your application ("stores" in Flux, "reducers" in Redux). Instead of letting the application code directly mutate the data, both tell you to describe every mutation as a plain object called an "action".

Unlike Flux, **Redux does not have the concept of a Dispatcher.** This is because it relies on pure functions instead of event emitters, and pure functions are easy to compose and don't need an additional entity managing them. Depending on how you view Flux, you may see this as either a deviation or an implementation detail. Flux has often been described as (state, action) => state. In this sense, Redux is true to the Flux architecture, but makes it simpler thanks to pure functions.

Another important difference from Flux is that Redux assumes you never mutate your data. You can use plain objects and arrays for your state just fine, but mutating them inside the reducers is strongly discouraged. You should always return a new object, which is easy with the object spread operator proposal, or with a library like **Immutable**. https://facebook.github.io/immutable-js

While it is technically possible to write impure reducers that mutate the data for performance corner cases, we actively discourage you from doing this. Development features like time travel, record/replay, or hot reloading will break. Moreover it doesn't seem like immutability poses performance problems in most real apps, because, as **Om** (https://github.com/omcljs/om) demonstrates, even if you lose out on object allocation, you still win by avoiding expensive re-renders and re-calculations, as you know exactly what changed thanks to reducer purity.

For what it's worth, Flux's creators approve of Redux.

9.2. Elm.

Elm is a functional programming language inspired by Haskell and created by Evan Czaplicki. It enforces a "model view update" architecture, where the update has the following signature: (action, state) => state. Elm "updaters" serve the same purpose as reducers in Redux.

http://elm-lang.org/

Unlike Redux, **Elm is a language, so it is able to benefit from many things like enforced purity, static typing, out of the box immutability, and pattern matching** (using the case expression). Even if you don't plan to use Elm, you should read about the Elm architecture, and play with it. There is an interesting JavaScript library playground implementing similar ideas. We should look there for inspiration on Redux! One way that we can get closer to the static typing of Elm is by using a gradual typing solution like Flow.

9.3. Immutable

Immutable is a JavaScript library implementing persistent data structures. It is performant and has an idiomatic JavaScript API.

https://facebook.github.io/immutable-js

Immutable and most similar libraries are orthogonal to Redux. Feel free to use them together!

Redux doesn't care how you store the state—it can be a plain object, an Immutable object, or anything else. You'll probably want a (de)serialization mechanism for writing universal apps and hydrating their state from the server, but other than that, you can use any data storage library as long as it supports immutability. For example, it doesn't make sense to use Backbone for Redux state, because Backbone models are mutable.

Note that, even if your immutable library supports cursors, you shouldn't use them in a Redux app. The whole state tree should be considered read-only, and you should use Redux for updating the state, and subscribing to the updates. Therefore writing via cursor doesn't make sense for Redux. If your only use case for cursors is decoupling the state tree from the UI tree and gradually refining the cursors, you should look at selectors instead. Selectors are composable getter functions. See reselect for a really great and concise implementation of composable selectors.

9.4. Baobab

Baobab is another popular library implementing immutable API for updating plain JavaScript objects. While you can use it with Redux, there is little benefit in using them together.

https://github.com/Yomguithereal/baobab

Most of the functionality Baobab provides is related to updating the data with cursors, but Redux enforces that the only way to update the data is to dispatch an action. Therefore they solve the same problem differently, and don't complement each other.

Unlike Immutable, **Baobab doesn't yet implement any special efficient data structures under the hood**, so you don't really win anything from using it together with Redux. It's easier to just use plain objects in this case.

9.5. RxJS

RxJS is a superb way to manage the complexity of asynchronous apps. In fact there is an effort to create a library that models human-computer interaction as interdependent observables.

https://github.com/ReactiveX/RxJS

Does it make sense to use Redux together with RxJS? Sure! They work great together. For example, it is easy to expose a Redux store as an observable:

Example:

```
function toObservable(store) {
  return {
    subscribe({ next }) {
      const unsubscribe = store.subscribe(() =>
next(store.getState()))
      next(store.getState())
      return { unsubscribe }
    }
  }
}
```

Similarly, you can compose different asynchronous streams to turn them into actions before feeding them to **store.dispatch().**

The question is: do you really need Redux if you already use Rx? Maybe not. It's not hard to re-implement Redux in Rx. Some say it's a two-liner using Rx .scan() method. It may very well be!

If you're in doubt, check out the Redux source code (there isn't much going on there), as well as its ecosystem (for example, the developer tools). If you don't care too much about it and want to go with the reactive data flow all the way, you might want to explore something like Cycle instead, or even combine it with Redux. Let us know how it goes!