React 状态管理（4）：不同方式对比

现在我们了解了 React 的状态管理，也学习了 Redux 和 Mobx 这两个第三方状态管理工具，是时候来对比一下这几种方式的优缺点了。

Mobx 和 Redux 的比较

Mobx 和 Redux 的目标都是管理好应用状态，但是最根本的区别在于对数据的处理方式不同。

Redux 认为，数据的一致性很重要，为了保持数据的一致性，要求Store 中的数据尽量范式化，也就是减少一切不必要的冗余，为了限制对数据的修改，要求 Store 中数据是不可改的（Immutable），只能通过 action 触发 reducer 来更新 Store。

Mobx 也认为数据的一致性很重要，但是它认为解决问题的根本方法不是让数据范式化，而是不要给机会让数据变得不一致。所以，Mobx 鼓励数据干脆就“反范式化”，有冗余没问题，只要所有数据之间保持联动，改了一处，对应依赖这处的数据自动更新，那就不会发生数据不一致的问题。

值得一提的是，虽然 Mobx 最初的一个卖点就是直接修改数据，但是实践中大家还是发现这样无组织无纪律不好，所以后来 Mobx 还是提供了 action 的概念。和 Redux 的 action 有点不同，Mobx 中的 action 其实就是一个函数，不需要做 `dispatch`，调用就修改对应数据，在上面的代码中，`increment` 和 `decrement` 就是 action。

如果想强制要求使用 action，禁止直接修改 observable 数据，使用 Mobx 的 `configure`，如下：

```
import {configure} from 'mobx';

configure({enforceActions: true});
```

总结一下 Redux 和 Mobx 的区别，包括这些方面：

1. Redux 鼓励一个应用只用一个 Store，Mobx 鼓励使用多个 Store；
2. Redux 使用“拉”的方式使用数据，这一点和 React是一致的，但 Mobx 使用“推”的方式使用数据，和 RxJS 这样的工具走得更近；
3. Redux 鼓励数据范式化，减少冗余，Mobx 容许数据冗余，但同样能保持数据一致。

然后，被问起最多的问题就来了：我应该选用 Mobx 还是 Redux 呢?

问：你的应用是小而且简单，还是大而且复杂?

如果是前者，选择 Mobx；如果是后者，选择 Redux。

问：你想要快速开发应用，还是想要长期维护这个应用?

如果是前者，选择 Mobx；如果是后者，选择 Redux。

我们真的必须使用 Mobx 和 Redux 吗

当然不是!

首先我们要明白，Redux 和 Mobx 都是一个特定时期的产物，在 React 没有提供更好的状态管理方法之前，Redux 能够帮助使用 React 的开发者一把，Mobx 也能提供一种全新的状态管理理念。

不过，React 是在持续发展的，光是 Context API 的改进，几乎就可以取代 react-redux 和 mobx-react 的作用。实际上，react-redux 和 mobx-react 两者的实现都依赖于 React 的 Context 功能。

以 Redux 为例，它相较于 React Context 还有哪些特点呢?

Redux 有更清晰的数据流转过程，配合 Redux Devtools 的确方便 Debug，代价就是我们必须要写啰嗦的 action 和 reducer。如果我们觉得应用并不需要 Redux 这样的增强功能，那完全就可以直接使用 React 的 Context。

当然，React 的 Context 还是过于简单了一点，我建议开发者不要只关注 Redux，可以尝试一些更加轻量级的第三方管理工具，其中的佼佼者，我认为就是 unstated 。

最后，还是苦口婆心地对开发者们说出这句我曾说过不下一万遍的话：**不要因为某个工具或者技术炫酷或者热门而去用它，要根据自己的工作需要去选择工具和技术。**

小结

本小节比较了多种管理 React 状态的方式，读者应该明白：

1. Redux 和 Mobx的不同；
2. 很多场合 React 自身的状态管理就足够用，并不是必须要使用 Redux 或者 Mobx 这样的第三方管理工具；
3. 随着 React 的发展，开发者对第三方工具的依赖会越来越少。