MobX 是一个优秀的响应式状态管理库

MobX 是一个简单、可伸缩的响应式状态管理库。通过 MobX 你可以用最直观的方式修改状态，其他的一切 MobX 都会为你处理好（如自动更新UI），并且具有非常高的性能。

MobX 的设计哲学

MobX 的设计哲学概括起来就一句话：**Anything that can be derived from the application state, should be derived. Automatically. (任何可以从应用状态中派生的内容，都应当自动地被派生。)**

理解这句话的关键是搞清楚【派生】的含义是什么？在 MobX 中【派生】的含义比较广泛，包括：

1. **用户接口(UI)，如组件、页面、图表**
2. **派生数据(computed data)，如从数组中计算得到的数组长度**
3. **副作用(side effect)，如发送网络请求、设置定时任务、打印日志等**

我们平时常看到的状态响应模型，其中的响应就可以看做是状态的一种派生，MobX 将这种模型进行泛化，形成更通用的状态派生模型，接下来会详细介绍。

MobX 状态响应模型

状态响应模型概括起来主要包含三个要素：定义状态、响应状态、修改状态。

MobX 中通过`observable`来定义可观察状态， 它接受任意 JS 值（包括`Object`、`Array`、`Map`、`Set`），返回原始数据的代理对象，代理对象与原始数据具有相同的接口，你可以把代理对象当做原始数据使用。

```
// 定义状态
const store = observable({
  count: 0
});
```

MobX 中通过`autorun`来定义状态变化时要执行的响应操作，它接受一个函数。此后，每当`observable`中定义的状态发生变化时，MobX 都会立即执行该函数。

```
// 响应状态
autorun(() => {
  console.log("count:", store.count);
});
```

```

MobX 中修改状态和修改原始数据的方式没什么区别，这也是 MobX 的优点——符合直觉的操作方式。

```
// 修改状态
store.count += 1;
```

把上面的部分串起来就是一个最简单的 MobX 示例了（如下），示例中每次修改 count 的值时会自动打印一条日志，并且日志包含最新的 count 值。这个示例揭示了 MobX 中最核心的功能。

```
import { observable, autorun } from "mobx";

 // 1. 定义状态

 const store = observable({
       count: 0
});

 // 2. 响应状态

autorun(() => {
      console.log("count:", store.count);
 });

// 3. 修改状态

store.count += 1;
store.count += 1;
store.count += 1;
```
```

```
// 控制台输出

// count: 0
// count: 1
// count: 2
// count: 3


```