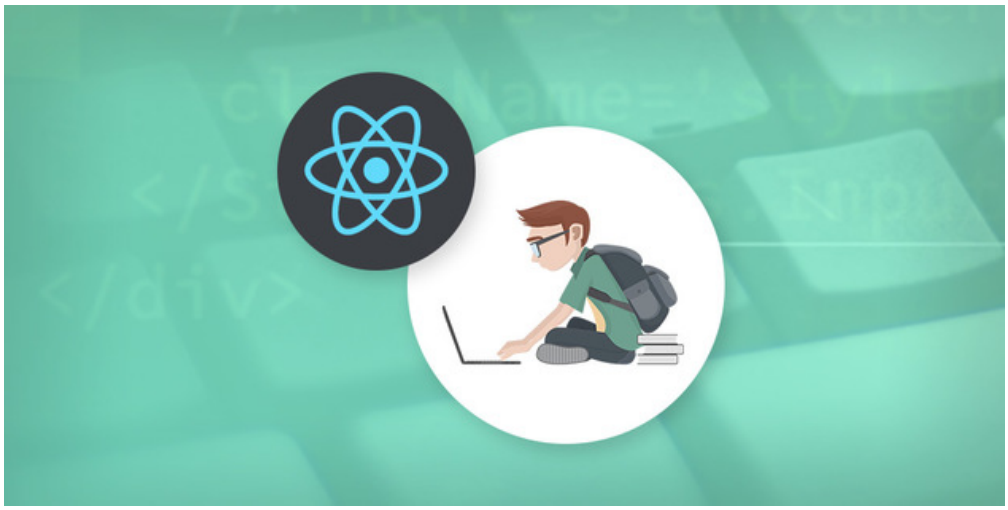
一篇是 Gianluca Guarini 写的 《Things nobody will tell you about React.js》，我将它译作 《那些入坑 React 前没有人会提醒你的事》，因为作者行文中明显带着对 React 的批判和失望。

另一篇则是 Facebook 员工，也是 Redux 作者的 Dan Abramov 针对上文的回复 《Hey, thanks for feedback!》。

1 引言



我为什么要选这篇文章呢？

我们团队最早在 2014 年中就确定了 React 作为未来的发展方向，那个时候很多人都还在感叹 Angular（那时候还是 Angular 1）是一个多么超前的框架，很多人甚至听都没有听说过 React。

在不到三年的时间里，React 社区迅速的发展壮大，许多 Angular、Ember、Knockout 等框架的拥趸，或主动或被动的都逐渐开始向 React 看齐。

站在 React 已经繁荣昌盛、无需四处布道宣传的今天，我们不妨冷静下来问问自己，React 真的是一个完美的框架吗？社区里一直不缺少吐槽的声音，这周我们就来看看，React 到底有哪些槽点。

2 内容概要

Gianluca Guarini 着重吐槽的点在于：

- React 项目文件组织规范不统一，社区中 Starter Kit 太多（100+），新手不知道该怎么组织文件
- 由于 React 只关心 View 层，开发者就要面临选择 mobx 还是 redux 的纠结，无论选择哪种都会带来一系列的问题（重新配置构建脚本，更新 eslint 规则等）
- 如果选了 mobx，会发现 mobx 无法保证自己的 store 不被外部更新（官方建议是加上特殊的前缀）
- 如果选了 redux，会发现要实现同样的功能需要写很多的重复代码（这也是为什么社区中有海量的 redux helper 存在）

- 路由用起来也很蛋疼，因为 React Router 几乎是社区中唯一的选择，但是这货版本更新太快，一不小心就用了废弃的 API
- 用 JSX 的时候总是要嵌很多没必要的 `div` 或 `span`
- 要上手一个 React 应用，要配置很多的构建工具和规则才能看到效果
- ...

Dan Abramov 的回复：

- 「React 16.0 引入的 Fiber 架构会导致现有代码全部需要重构」的说法是不对的，因为新的架构做到了向后兼容，而且 Facebook 内部超过 3 万个组件都能无痛迁移到新架构上
- 缺少统一脚手架的问题，可以通过 create-react-app 解决
- 觉得 redux 和 mobx 繁琐的话，对于刚刚上手的小应用不建议使用
- React Router 升级太频繁？2015 年发布的 1.0，2016 年 2 月发布的 2.0，2016 年 10 月发布的 3.0。虽然 4.0 紧接着 3.0 马上就发布了，但是 React Router 很早就已经公布了这样的升级计划。
- ...

3 精读

本次提出独到观点的同学有：@rccoder @Turbe Xue @Pines-Cheng @An Yan @淡苍 @黄子毅 @宾彬 @cisen @Bobo 精读由此归纳。

很高兴能看到不少新同学积极参与到精读的讨论中来，每一个人的声音都是社区发展的一份力量。

React 上手困难

很早之前我们去四处布道 React 的时候，都会强调 React 很简单，因为它的 public API 非常之少，React 完整的文档 1 个小时就能看完。

那么说「React 上手困难」又是从何谈起呢？参与精读的同学中有不少都有 Vue 的使用经验（包括本周吐槽文的作者），所以不免会把两个框架上手的难易程度放在心里做个对比。

都说没有对比就没有伤害，大家普遍的观点是 Vue 上手简单、文档清晰、构建工具完善、脚手架统一……再反观 React，虽然 Dan 在文章里做了不少解释，但引用 @An Yan 的原话，『他也只是在说「事情没有那么糟糕」』。

所以说，大家认为的 React 上手困难，很大程度上不是 React 本身，而是 React 附带的生态圈野蛮发展太快，导致新人再进入的时候普遍感觉无所适从。虽然官方的 create-react-app 缓解了这一问题，但还没有从根本程度上找到解法。

状态管理的迷思

在今时今日的前端圈子里，说 React 不说 Redux 就像说 Ruby 却不说 Rails 一样，总感觉缺点儿什么。

因为 React 将自己定位成 View 层的解决方案，所以对于中大型业务来说一个合适的状态管理方案是不可或缺的。从最早的 Backbone Model，到 Flux，再到 reflux、Redux，再到 mobx 和 redux-observable，你不得不感叹 React 社区的活力是多么强大。

然而当你真正开始做新项目架构的时候，你到底是选 Redux 还是 Mobx，疑惑是封装解决方案如 dva 呢？ @淡苍 认为，Redux 与 MobX，React 两大状态管理方案，各有千秋，Redux 崇尚自由，扩展性好，却也带来了繁琐，一个简单的异步请求都必须引入中间件才能解决，MobX 上手容易，Reactive 避免不必要的渲染，带来性能提升，但相对封闭，不利于业务抽象，缺少最佳实践。至于如何选择？根据具体场景与需求判断。

不难看出，想要做好基于 React 的前端架构，你不仅需要对自己的业务了如指掌，还需要对各种解决方案的特性以及适合怎样的业务形态了如指掌。在 React 社区，永远没有标准解决方案。

Redux 亦非万能解

Redux 在刚刚推出的时候凭借酷炫的 devtool 和时间旅行功能，瞬间俘获了不少工程师的心。

但当你真正开始使用 Redux 的时候，你会发现你不仅需要学习很多新的概念，如 reducer、store、dispatch、action 等，还有很多基础的问题都没有标准解法，最典型的例子就是异步 action。虽然 Redux 的 middleware 机制提供了实现异步 action 的可能性，但是对于小白来说去 dispatch 一个非 Object 类型的 action 之前需要先了解 thunk 的概念，还要给 Redux 添加一个 redux-thunk 中间件实属难题。

不仅如此，在前端工程中常见的表单处理，Redux 社区也一直没有给出完美的解法。前有简单的 util 工具 redux-form-utils，后有庞大复杂的 redux-form，还有 rc-component 实现的一套基于 HOC 的解决方案。若没有充分的了解和调研，你将如何选择？

这还没有提到最近非常火热的 redux-saga 和 redux-observable，虽然 Dan 说如果你不需要的话完全可以不用了解，但是如果你不了解他们的话怎么知道自己需不需要呢？

React 与 Vue 之争

Vue 之所以觉得入门简单，因为一开始就提供了 umd 的引入方式，这与传统 js 开发的习惯一致，以及 Avalon 多年布道的铺垫，大家可以很快接受一个不依赖于构建的 Vue。

React 因为引入了 JSX 概念，本可以以 umd 方式推广，但为了更好的 DX 所以上来就推荐大家使用 JSX，导致新手觉得门槛高。

React + Mobx 约等于一个复杂的 Vue，但这不是抛弃 React 的理由。为什么大家觉得 Vuex 比 Redux 更适合 Vue 呢？因为 Vuex 简单，而 Redux 麻烦，这已经将两个用户群划分开了。

一个简单的小公司，就是需要这种数据流简单，不需要编译，没有太多技术选型要考虑的框架，他们看中的是开发效率，可维护性并不是第一位，这点根本性的导致了这两类人永远也撮合不到一块。

而 Vue 就是解决了这个问题，帮助了那么多开发者，仅凭这点就非常值得称赞，而我们不应该从 React 维护性的角度去抨击谁好谁坏，因为站在我们的角度，大部分中小公司的开发者是不 care 的。

React 用户圈汇集了一批高端用户，他们不断探索技术选型，为开源社区迸发活力，如果大家都转向 Vue，这块摊子就死了，函数式、响应式编程的演进也会从框架的大统一而暂时终止，起码这是不利于技术进步的，也是不可能发生的。Vue 在自己的领域做好，将 React 敏捷思想借鉴过来，帮助更多适合场景的开发者，应该才是作者的目的。

小贴士：如何在开源社区优雅的撕逼

开源社区撕逼常有，各种嘴炮也吃充斥在社区里，甚至有人在 Github 上维护了一份开源社区撕逼历史。虽然说做技术的人有争论很正常，但是撕的有理有据令人信服的案例却不多。这次 Facebook 的员工 Dan Abramov 就做出了很好的表率。面对咄咄逼人的文章，逐条回复，不回避、不扯淡且态度保持克制，实属难能可贵。

3 总结

React 开发者们也不要因为产生了 Mobx 这种亲 Vue 派而产生焦虑，这也是对特定业务场景的权衡，未来更多更好的数据流方案还会继续诞生，技术社区对技术的优化永无止尽。

比如 mobx-state-tree 就是一种 redux 与 mobx 结合的大胆尝试，作者在很早之前也申明了，Mobx 一样可以做时间旅行，只要遵守一定的开发规范。

最后打个比方：安卓手机在不断进步，体验越来越逼近苹果，作为一个逼格高的用户，果断换苹果吧。但作为 java 开发人员的你，是否要为此换到 oc 流派呢？换，或者不换，其实都一样，安卓和苹果已经越来越像了。

> 讨论地址是：那些入坑 React 前没有人会提醒你的事 · Issue #13 · dt-fe/weekly

> 如果你想参与讨论，请点击这里，每周都有新的主题，每周五发布。