开篇词 | 这一次，真正吃透 React 知识链路与底层逻辑

2020/10/12 修言

在接下来的一段时间里，我们将一起深入 React 这个框架领域，完成从“小工”到“专家”的蜕变。

作为一名React 重度用户，与其说我对 React 源码、底层原理及周边生态有着较为深入的探究，不如说我对它们有着浓厚的兴趣。早期，我专注于性能优化和前端工程化，曾将线上大型应用性能提升率做到 40%，并基于 React 打造过团队新基建。此外，我还担任过多年一线前端面试官，积累了丰富的面试经验。

前端生涯至今，我从未停止过挑战自己的能力边界，始终乐于拥抱新的技术和工具，这不止让我保持了很好的职场竞争力，还使我深知新手从入门到精通过程中的痛点和难点。

作为一线开发者，我不认同技术圈时下盛行的“造名词”风气，痛恨故弄玄虚的“语言壁垒”——其实**技术本身在多数情况下都是一些简单且有趣的东西，人们越是试图神化它，越容易脱离技术本质**。这也是我在这个专栏中秉持和践行的一个原则。

学好 React，到底有多爽?

> **这世界上没有一个前端不想学好 React，如果有，那可能你还没想明白。**
>
> **——修·格拉底·鲁迅·言斯妥耶夫斯基**
>
> @拉勾教育

在过去的几年，“变化”始终是前端框架世界里的一号关键词：前有 jQuery 刚刚式微时各路神仙各显神通，后有 React/Vue/Angular 三分天下，如今又渐渐演变成了 React/Vue 两分天下。

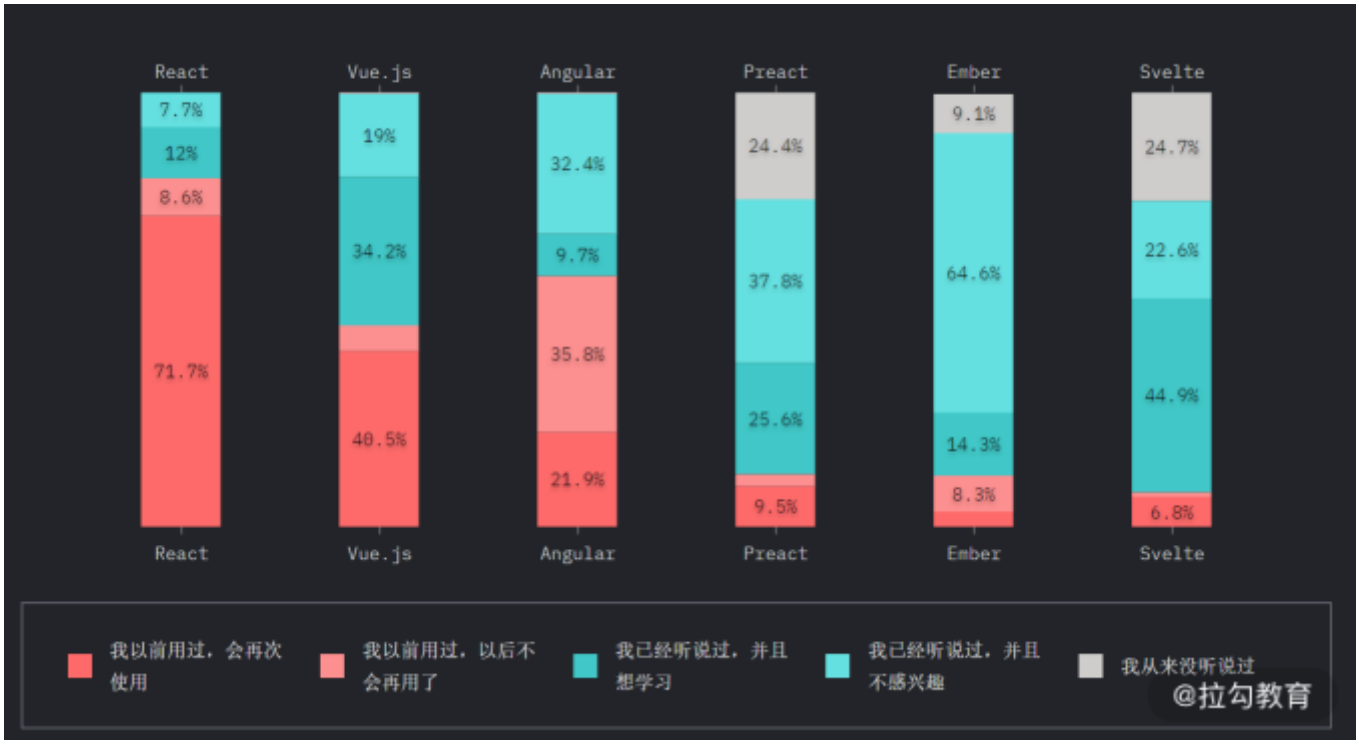
而反观框架本身，你会发现 Vue、React 乃至 Angular 之间不仅写法越来越像，甚至在设计层面也日渐趋同——它们似乎像是约好了一样，在齐刷刷地朝着 WebComponents 标准前进。因此在展望未来的前端框架时，我们有充分的理由相信，属于前端框架的一号关键词终有一日会从“变化”发展为“稳定”或“标准化”。

在这样的趋势下，站在任何主观视角去拉踩任何前端框架的行为都是不合适的。**学习者在意的不应是“哪个框架最牛”这样娱乐性的问题，而应该是学习的效用。**

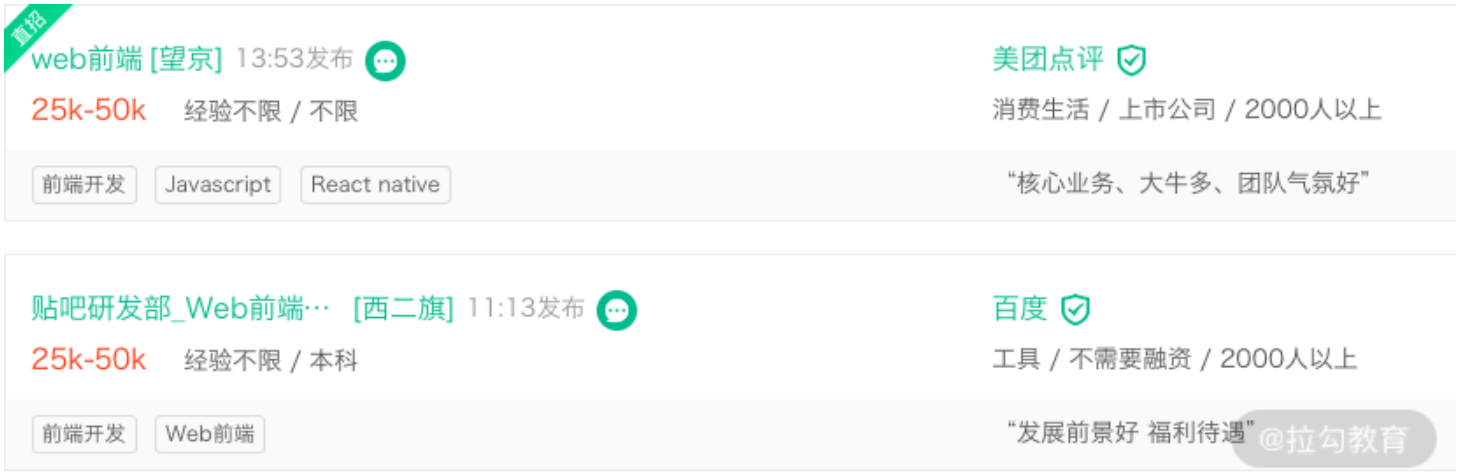
那么学习 React，将会带来什么样的效用呢？

利好个人职业生涯：大厂更喜欢 React

若单说岗位数量，我不敢妄言，但**在一二线的互联网大厂中，React 的绝对优势凸显无疑。**（比如，阿里就统一使用 React 作为底层技术栈，并且在内部紧密共建 React 生态。）国外的一份“2019 年度 JavaScript 趋势报告”中，React 也被评估为综合指数最高的前端框架：



在招聘上，大厂普遍青睐 React 人才，各种高薪职位中不乏“精通 React”“掌握 React”的字眼。作为前端，我们必须认识到这样一个现状：**大厂（包括国内、国外）更喜欢 React，当我们立下一个有朝一日进大厂的志愿时，就意味着必须先下定决心搞定 React。**



（信息来源：拉勾网）

强化项目实战能力：吃透 React，疑难杂症不在话下

面试时，React 相关的问题往往具备较高的区分度，能够在 React 方面脱颖而出的候选人并不多。很多时候，候选人似乎也确实不理解面试官“为什么要问得这么难”。比如常见的吐槽就有“我不读源码，不研究调用栈，用 React 写业务照样一把梭”这样的说辞。

确实，通过阅读 React 文档以及市面上一些“快速上手”“XX 实战”类型的学习材料，也能胜任一定的业务开发工作，但当业务复杂度攀升，“奇形怪状”的问题就会如雨后春笋般接连冒头。当你对 React 的运行机制不甚了解时，遇到这样的“疑难杂症”，就很容易懵掉。

面试环节的 React 深度考察，正是为了筛选出这些能够真正吃透 React、解决复杂问题的“高级玩家”：**对 React 的理解深度，将决定着你所能解决的实战问题复杂度的上限。**

普通开发者的“逆袭”机会：一个好的框架，就是最好的老师

这两年，许多中小型公司的前端工程师都面临着这样一个困境：**业务含金量不高，老板又不重视，技术专项难以提取，架构机会更是没有……好像永远都没办法破局，难道我这辈子就这样了吗**？

当然不是！当环境无法给我们提供优质的成长途径时，不妨自己尝试创造途径，比如：

- 深挖一个优质的前端框架，吃透其底层原理；
- 跟框架作者（React 团队）学架构思想、学编码规范、学设计模式。

React 正是一个优秀前端框架的典型，它在架构上融合了数据驱动视图、组件化、函数式编程、面向对象、Fiber 等经典设计“哲学”，在底层技术选型上涉及了 JSX、虚拟 DOM 等经典解决方案，在周边生态上至少涵盖了状态管理和前端路由两大领域的最佳实践。此外，它还自建了状态管理机制与事件系

统，创造性地在前端框架中引入了 Hooks 思想..... React 十年如一日的稳定输出背后，有太多值得我们去吸收和借鉴的东西。

这个专栏我将带你掌握目前行业里相对前沿且具有代表性的一套东西，也是真正能够在你的职业生涯里沉淀下来、发挥长期效用的“底层技能”。

React 为什么这么难学?

在实际的招聘过程中，我和同事都曾经不止一次地发过这样的感慨：**当下要想从社区招到一个符合预期的 React 开发，真的太太太太太太难了**。

不知道你有没有观察到一个比较有趣的现象：Vue 知识体系/原理的相关内容百花齐放，但 React 知识体系/原理的相关内容却屈指可数。

市面上以 React 为主题的进阶性内容，大部分是在教会一系列 API 的基础上，描述如何去实战一个具体的项目，即专精于“使用”；而为数不多的源码分析性内容，虽然试图去拆解“原理”，但却往往伴随着细化到逐行代码的知识粒度，对读者的时间、耐力和既有水平（提炼知识、抽象知识的能力）都提出了很高的要求。

这些现象的背后，和 React 令人望而却步的庞大知识体系、精密复杂的底层原理以及长长的知识链路是分不开的。平心而论，学透 React 很难，而我想帮你解决的，也正是这个“难”。

课程设计：串联知识链路，讲透底层逻辑

我分享技术内容有两年多了，一直将“接地气、说人话”作为写作的第一要务，这个专栏更是将“**把复杂的问题简单化、把琐碎的问题系统化**”作为课程设计的核心原则。它并非平铺直叙的学习笔记，而是一次我与你之间的对话。

我希望做一个能够将学习体验与知识深度中和到最佳状态，切实为你带来学习效用的专栏。为此，专栏在设计层面做了以下几件事情。

设计原则：贴着大厂面试逻辑做大纲，贴着源码讲原理

大厂的 React 面试不是走过场，更不是“造火箭”式的炫技，它是最有“效用导向”的一个学习依据。如果能够将大厂面试的逻辑利用充分，我们将实现面试和应用的双重突破。

贴着源码讲原理，绝不是带着你死磕源码，源码 !== 原理，源码是代码，而原理是逻辑，代码是繁杂冗长的，原理却可以是简洁清晰的。在一些场景下，源码确实能够成为一个不错的教具，但阅读源码不是抵达原理的唯一途径。因此，必要时我会提取对你理解问题有帮助的源码；也会在一些场景下选取其他的教具，确保你能够用正确且高效的姿势抵达知识的重点。

专栏所涉及的原理，**可以帮你解决实际工作中的大多数疑难杂症，也可以 Match 上大厂对资深前端工程师的技术深度的要求**。

对于体系性较强的知识：创建足够充分的上下文

之前曾经读到过木心关于红楼梦的书评，印象极深：“红楼梦中的诗词像是水中摇曳的水草，美极。若是捞出来看，就干巴巴了。”

同样的道理也适用于 React 的知识链路：**一些知识之所以难学，不是因为它有多复杂，而是因为理解它是需要上下文的**。你若把它放到正确的上下文里，可能想通这件事也就是一瞬间的工夫；但如果你的学习上下文是断裂的，那么知识点本身自然会变得“干巴巴”，难以下咽。

对于复杂度较高的知识：用现象向原理提问

考虑来学习这门专栏的同学的学习阶段参差不齐，我在讲解复杂原理时，会尽量遵循“**先提现象/问题，再挖原理**”这个顺序，将困难知识的学习坡度降至最低。专栏中有一些内容的前置知识，我写得比较细，一般也会提前标明这是“先导知识”，如果你是高端玩家，直接跳过即可。

整个专栏的结构规划思路如下。

- **模块一：基础夯实**。这部分内容涉及 React 的基本原理和源码，对大多数人普遍薄弱的、说不清楚的基础知识做深入浅出的讲解，帮你突破一些重点和难点。
- **模块二：核心原理**。这部分内容源于日常开发中的疑难杂症、大厂面试的压轴难题，呈现出框架的底层逻辑和源码设计，我将用最少的篇幅来提取尽量多的信息。如果你想要从事一些高级岗位，或者精通 React，那么这块的内容你肯定避不开，而面试官能够通过这些内容，对候选人的能力做一个评价，甚至是定级。
- **模块三：周边生态**。很多人用过 Redux、听说过 React-Router，但为什么要用它？其背后的工作原理、设计思想又是怎样的？专栏要讲的就是这部分比较有区分度的内容，面向使用过 React 全家桶，或者接触过还不具备熟练使用能力的前端工程师，解决你出了 Bug 却不知如何调试的问题。
- **模块四：生产实践**。对于一个优秀的前端应用来说，性能和设计模式是永恒的主题，性能决定用户体验，设计模式决定研发效率。这部分将结合我团队的实践经验以及当下行业里推崇的最佳实践，为你输出实战观点。对于这些最佳实践，你不仅要知道怎么做，还要理解“为什么这么做”。学完个模块可以强化你的实际应用能力，提升自主研发创新实践的线索和灵感。

《深入浅出搞定 React》大纲

开篇词 | 这一次，真正吃透 React 知识链路与底层逻辑

模块一：系统深入学习“基础知识”

1 JSX 代码是如何“摇身一变”成为 DOM 的?

2 为什么 React 16 要更改组件的生命周期? （上）

3 为什么 React 16 要更改组件的生命周期? （下）

4 数据是如何在 React 组件之间流动的? （上）

5 数据是如何在 React 组件之间流动的? （下）

6 React-Hooks 设计动机与工作模式（上）

7 React-Hooks 设计动机与工作模式（下）

模块二：刨根问底吃透“核心原理”

8 深入 React-Hooks 工作机制：“原则”的背后，是“原理”

9 真正理解虚拟 DOM：React 选它，真的是为了性能吗?

10 React 中的“栈调和”（Stack Reconciler）过程是怎样的?

11 setState 到底是同步的，还是异步的?

12 如何理解 Fiber 架构的迭代动机与设计思想?

13 ReactDOM.render 是如何串联渲染链路的？（上）

14 ReactDOM.render 是如何串联渲染链路的？（中）

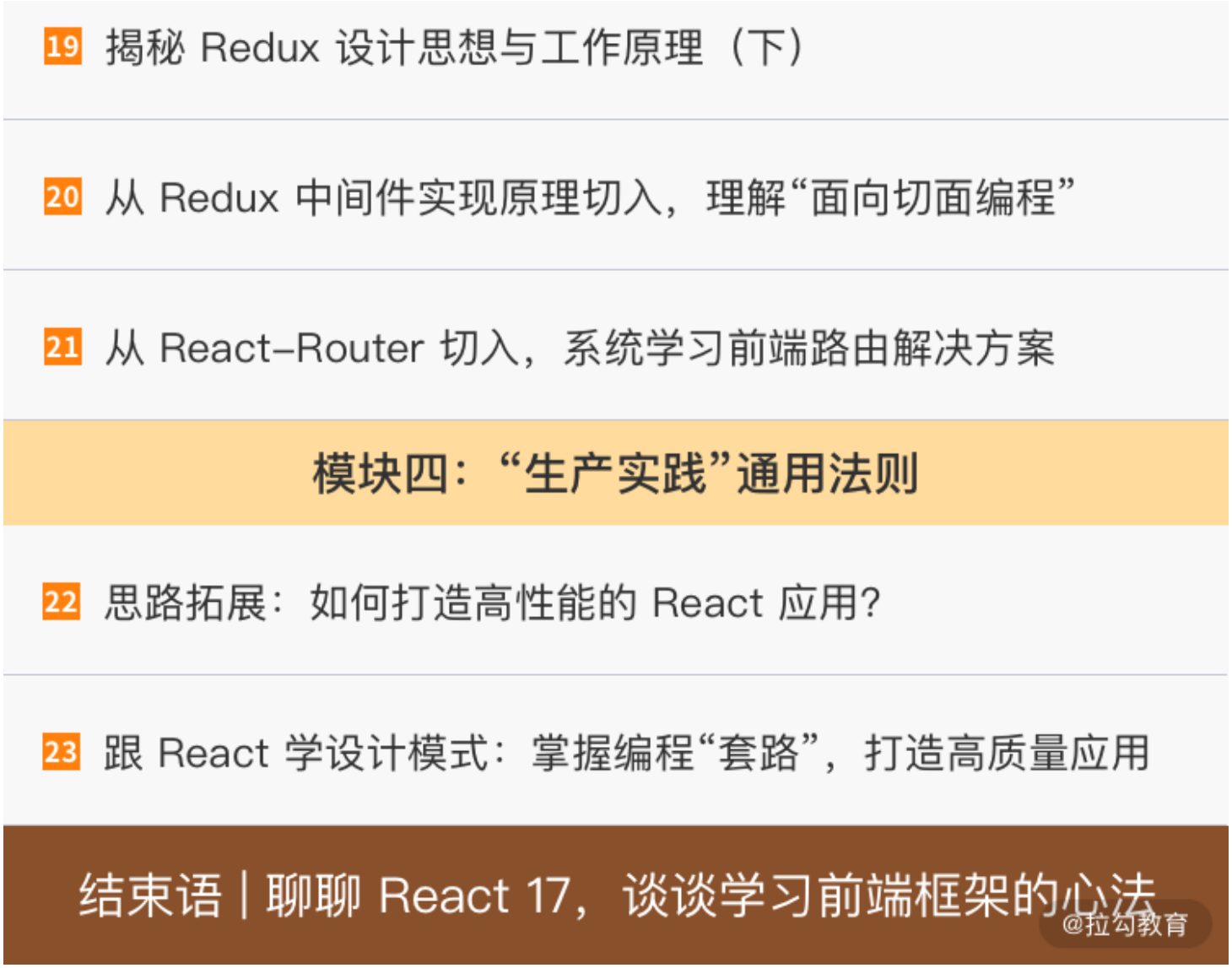
15 ReactDOM.render 是如何串联渲染链路的？（下）

16 剖析 Fiber 架构下 Concurrent 模式的实现原理

17 特别的事件系统：React 事件与 DOM 事件有何不同?

模块三：“周边生态”帮你拓宽技术视野

18 揭秘 Redux 设计思想与工作原理（上）



讲师寄语

我认为，学习的本质是重复，对于重要的知识，我会翻来覆去地说，想方设法让你记住它。所以，如果你在学习过程中发现某一块知识似曾相识或者早已埋下伏笔，多半意味着你发现了一个重难点，请牢牢抓住它。

为了将晦涩的知识转化为你手里实实在在的生产力，专栏的框架和内容也历经了多次的迭代和重构。每一次的果断推翻早期设想，每一次的重构表达逻辑，都是希望帮你更好地消化吸收这份知识。希望你也能够不吝耐心和智慧，顺利走完整个 React 学习曲线中最难的一段路。

到这里，我的故事就结束了，而你和我的故事才刚刚开始。欢迎在留言区分享你的前端经历，或者写写学习 React 过程中遇到的问题、想要学习的内容，让我们一起写出“我们”的故事，开启 React 奇幻之旅。