拓展：如何做好一个开源项目（上篇）

iView 的故事

毕业四年以来，我一直觉得自己是一个很幸运的人，幸运参与过创业，幸运一路有大牛带，幸运开源了 iView 项目。

2016 年初，我还是一名普通的前端工程师，那时候还是 Vue.js 1.x 的时代，知名度也远不如现在，在大部分人眼中，Vue.js 就是一个轻量级的 Angular。

我所在的公司是做 to B 业务的，与大部分公司一样，那时主导 jQuery，把 Vue.js 引入团队乃至整个公司，是一件很不易的事，因为不是你自己在用，你要说服所有人用，现在看来，当初的决定是很正确的。如果你有兴趣，欢迎阅读扩展阅读 1，那篇文章详细介绍了我的 Vue.js “推广史”。

到了 16 年 7 月，我们团队已经完全认可了 Vue.js + Webpack 的技术栈，一直践行至今。一个偶然的机会，公司举办了一个创新大赛，作为可视化团队，我提议做一套自己的组件库。当初也只是一个想法，抱着试试看的态度就报名了。比赛的目的只是呼吁大家创新，后续就没有动作了，但是我没有就此放下，从那时起，几乎全职开发 iView，两年半来，我每天的工作基本就是维护 iView。不得不说，一个成功的开源项目，必须有 leader 的认可和公司的支持，缺一不可。

那时不比现在，优秀的 Vue.js 组件库非常少，文献更是少得可怜。一开始做 iView 没什么头绪，只是规划了一张 MindNode 脑图，罗列了一期要做的所有组件。这里有一个插曲，因为这张 todo 的脑图

有贴在 GitHub 的 Readme，以至于一开始大部分 issues 都是问这个脑图是用什么软件做的，因为做的很好看，索性写了个提问须知，注明了脑图是用 MindNode 制作的。

起初对开源工作不是很了解，连一个编译工程都搭不起来，想了数日也搞不定。与大部分人一样，一开始也是要参考其它人的开源项目（就像现在很多人参考 iView 是一样的），那是我参考 Vux（移动端的 Vue.js 组件库，当时有 6k star），向作者捐了杯咖啡钱，顺便加了好友（因为是支付宝捐赠，能加好友），请教了工程的问题，这才一步步搭起来，现在想想真是幸运。

万事开头难，把地基搭好，剩下的都是添砖加瓦的事。差不多 16 年底，iView 一期完成了（43 个组件），也发布了第一个正式版，有了第一批用户（当然，主要还是自己公司），然而在那个时候，又做了一个在今天看来稍晚但正确的决定——支持 Vue.js 2。

虽然是开源项目，但主要还是先满足自己团队和公司的需求，那个时候所有的项目都是 Vue.js 1.x 的，还没有用 Vue.js 2，但 2.x 已经发布有一段时间了，以至于在 2017 年初，GitHub 有很多 issues 问什么时候支持 2.x。很长一段时间没有升级到 2.x，主要因为要支持公司的项目，而我当时也觉得升级到 2.x 是一个很耗时的工作，就没有支持。后来，有一个人提交了一个非常大的 PR，将所有组件都支持了 2.x，而且只用了一个周末的时间。这件事让我意识到支持 Vue.js 2.x 的重要性了，于是花了 2 周时间研究，并升级到了 2.x（那个 pr 没有直接合并，而是参考，因为很多地方有 bug）。接下来的几个月，都在不断维护新的 iView，使用者也不断增多，社区又重新活跃起来。

如果当初没有及时转型（现在看来仍然是晚了），还在一味地维护 Vue.js 1.x 版本，那 iView 也许早就完了。有了好的开头，就要不放弃，不抛弃，坚持做下去，认真处理每一个 PR，因为质量是一个开源组件库最重要的，但并不是每一个 PR 都适合 merge，即使作者付出了很多时间提交这个 PR，也会有质量问题。

有了不错的口碑，在 2018 年继续完善，目前已经是同类产品里功能最丰富的组件库，而且在 18 年的 7 月 28 日（iView 两周年）成功举行了 3.0 发布会，一切都在朝着更好的一面发展着.....

这就是我的 iView 开源的故事，一个成功的开源项目，或许带有一点小幸运，因为如今的同类开源产品已经数不胜数了，不过确实有很多值得分享的地方，如果你打算做一个开源项目，希望下面的些许经验能够帮助到你。

不要盲目造轮子

每一个做开源项目的开发者，都是有目的的，如果你做一个开源项目没有任何目的，那你的目的多半是要造个轮子来提升下技术。

中国当前的开源环境和氛围虽不是很好，“拿来主义”者居多，但相比几年前，已是不错了，更多的人喜欢把自己杰出的代码开源出来。目前多数“正经”的开源项目，都是 KPI 导向的，不能说项目不好，只是一定时间后，可能就不维护了，而开源项目不仅仅要解决问题，持续维护也是观望着最在意的。

相比 React，Vue.js 的组件库开源项目要多的多，每隔一段时间就能看到几个新的，可能是因为 Vue.js 更容易上手，开发者很活跃。既然已经有这么的同类项目了，那为什么还要持续造轮子呢？因为中国的开发者就是喜欢自己折腾一个，而不是去维护一个现成的。

就以 Vue.js 组件库的开源现象来说，源源不断的轮子，主要还是市面上已有的组件库不能完全满足自己的业务，主要还是 UI 层面的，单论功能，市面上成熟的组件库足够完善，甚至超出你的业务需求。不好说这种现象是好是坏，好的是从头开发一个组件库，对技术的提升还是有的，它会督促你学习别人的代码，来改良自己的代码，否则造一个还不如别人的轮子也就没意义了。对于公司来说，也有了自己的 UI 组件库和规范（虽然很多只是 fork 后改的），感觉用自己的东西，对内对外都是一件很自豪的事。不好的就是，这是一种程序员

向产品或 UI 或 leader 的妥协，因为没办法说服他们使用市面上成熟的组件库，否则，完全可以把造轮子的精力用于维护一个成熟的项目。

造轮子这个词，似乎成了前端圈的代名词，的确，前端的造轮子能力是极强的，但不能盲目造轮子。如果你只是仿照其它开源项目练习，那就不说了，如果是想认真做一个开源项目，而不是造个轮子玩玩，那一定要构思好你要做的东西。

一般来说，在决定做一个开源项目前，都是要做市场调研的，你要很清楚的知道，自己做的项目弥补了同类产品的哪些不足，或者有哪些新的特性，因为它们是用户选择你的开源项目的主要依据，否则内容都一样，为什么不选一个成熟的呢。

如果某个开源项目已经很不错，但你希望基于它进行改造，但不是以 PR 的形式，那你可以在开源协议允许的前提下，fork 后，基于某个 release 独立维护。

每个开源项目都有一个核心的功能，或是解决用户的核心痛点，比如 iView 就是解决用户建站的问题，相比同类产品，它的特点就是组件丰富程度最高，功能也是最全面的，生态完善，有技术支持渠道。如果你的开源项目解决的问题是其它任何开源项目没有的，那用户很有可能会使用。但对于相同功能的，用户更倾向于选择还在维护的、star 数多的（star 多，说明这个项目的关注度高，更活跃）。所以，在做开源时，你不仅要知道自己产品的核心技术和特性，还要了解市面同类产品，去其糟粕，取其精华，不断更新和修复 bug，逐渐就会获得第一批用户。

做了东西要用

虽然我这两年的工作基本全是在维护 iView 开源项目，但偶尔也会穿插做几个项目，因为使用了，才能更好地了解自己的开源项目。

你可能会问，我每天都在维护 iView，还有我不了解的吗？还真是，维护和使用完全不一样，在开发组件库时，往往只聚焦在某个组件上，我们定义了 API，然后通过文档告诉使用者怎么用，有些功能是实现了，然而维护者只知道提供了这个功能，却不知在实际项目中好不好用，能用和好用是两个概念。

很多 feature，是自己用了才提炼出来的。比如 iView 的 [单元格组件 Cell (https://www.iviewui.com/components/cell)](https://www.iviewui.com/components/cell) 和 [相对时间组件 Time (https://www.iviewui.com/components/time)](https://www.iviewui.com/components/time)，都是我在做项目时发现并增加的，没有项目的支持，可能觉得这个组件没什么意义，完全应该由使用者在业务里自己写，作为基础组件后，使用体验确实好很多。

如果你足够重视你的开源项目，应该亲自使用并且安利别人使用来收集反馈，对于组件库来说，细节是很重要的，而很多细节与我们的用户习惯有关，比如 iView 在 3.0 版本开始，对按钮 Button 组件提供了一个新的 props： to，用于指定跳转路径，之前版本如果点击按钮跳转，必须监听它的 @click 事件，然后通过 vue-router 的编程式导航（也就是通过 JS）跳转，如果不亲自使用，绝对不知道这种原先的跳转模式写起来有多麻烦。如果你足够注重细节，这个 Button 组件在跳转时，还应该支持键盘的 Command 键（Windows 为 Ctrl，要做兼容）在新窗口打开链接。这些细节都与使用习惯息息相关，所以，一定要多用用自己的开源项目，在一个个小细节都处理好后，你会的开源项目自然会蒸蒸日上。

第一批用户

开源项目做好后，要获得第一批使用者。现在的环境，大多是公司或团队主导做开源，个人的很少，所以你的公司或团队自然就是第一批用户，做开源的主要目的，也是服务他们。

第一批用户也可以算是开源项目的小白鼠，一开始说服全公司使用，还是比较困难的，可以先小范围推广使用。公司都希望自己的产品稳定，而新的开源项目前期必定会有不少 bug，在经历几个小项目试水后，再尝试向更多的团队推广。因为有了成功案例，又是“自家”的开源项目，给自己人推广还是比较容易的。

在第一批用户的推广中，你的 leader 可能会起到决定性作用。你的开源项目，八成市场上已经有同类的了，不得不承认，即使让“自己人”做技术选型，也更期望选市面上成熟的，这种情况，就需要你的 leader 出来“拍板”了，否则，你的开源项目也许永远不会有第一批用户来试错。记得当时我在团队推广 Vue.js 时，起初也是很多质疑，有一部分人坚持使用 jQuery + 前端模板，得到 leader 的认可后，试水了几个项目，大家都能感受到 Vue.js 和 webpack 带来的高效开发体验，从此就没人再用 jQuery 了。不过呢，你最好确保你的开源项目质量还不错，否则这锅就得 leader 替你背了。

与市面上同类成熟的开源项目相比，你的开源项目最大的优势就是你能为第一批用户提供优质的“售后”服务。如果是同事的问题，你可以很快直接解答；如果是通过 GitHub 提交的 issues（前期不会很多了），尽量在半小时内回复，这样使用者会觉得你确实在用心维护这个开源项目。通过前期的口碑积累，你的第一批用户也成为了最忠实的用户，这时可以组建微信或 QQ 群，更直接地提供“售后”，而这第一批忠实用户，会成为后期推广的重要人脉。

生态

生态（ecosystem）不是与生俱来的，当你的开源项目有了一定的规模后，可以考虑发展生态体系。比如 iView，起初只是一个组件库，后续逐渐提供了 iview-project 工程、具有 GUI 的 iView CLI（这个概念可比 Vue CLI 3 早了一年）、后台模板 iview-admin、支持 Vue CLI 3 的 vue-cli-plugin-iview，以及业务组件 iview-area 和 markdown 编辑器 iview-editor，再到后来支持 SSR、TS。

完善的生态体系对于新用户来说，可以最快速搭建产品，减少学习和开发成本；对于观望者（正在决定是否使用的人）来说，更愿意选择生态完善的开源项目。所以，你的开源项目生态越完善，使用者也会越多。

生态的另一个好处，就是让用户产生依赖。最典型的例子就是 Vue 全家桶，一般刚接触的人，只会用个 Vue.js，再后来 vue-router、vuex 都是必须的了，再后来搭配一个三方的组件库，比如 iView，各种业务都能轻松应对。一旦用户对你的生态足够依赖，就很难更换技术栈了，因为生态的深入，更换成本很大，这也是很多企业的老项目所谓的“历史原因”。

生态的建设，不一定都是官方的行为，但是最核心的还是要自己维护，用户既然选择你的开源项目，也就意味着信任你的技术实力，放心用你的生态。项目到了一定规模，自然有不少第三方的开发者一起建设生态，这些 contributors 都是最有价值的开发者，尽量联系他们，一起来贡献更多的代码。

对于 Vue.js 组件库来说，生态一般分为脚手架、后台模板和业务组件。最新的 Vue CLI 3 提供了插件机制，现在的主流做法都是提供一个类似 vue-cli-plugin-iview (https://github.com/iview/vue-cli-plugin-iview) 的插件，很少有单独提供自己的工程了，在文档里，要推荐使用者用 Vue CLI 3 来管理项目，享受 Vue 的生态。后台模板是开箱即用的，默认配置好了路由、权限管理、多语言、登录等常规的后台系统功能，使用者 down 下来后，稍作修改就能很快开发自己的后台管理系统，主流的 UI 组件库，都会提供自家的后台模板，当然也有第三方专注在做后台模板的。最后一类业务组件，比如城市级联选择器 iview-area (https://github.com/iview/iview-area)，它基于 iView 的基础组件开发，但又不是基础组件，所以不能归到 iView 里，只能作为独立组件单独维护，业务组件理论上使用者可以自己封装，但是重复性的工作，还是交给社区做吧，这就是开源。

下一篇，将介绍：

- 持续运营
- 国际化
- 让更多的人参与
- 让 Robot 来做“坏人”
- 赞助与商业化

扩展阅读

- 2016我的心路历程：从 Vue 到 Webpack 到 iView (https://juejin.im/post/5880108f8d6d810058ae0def)