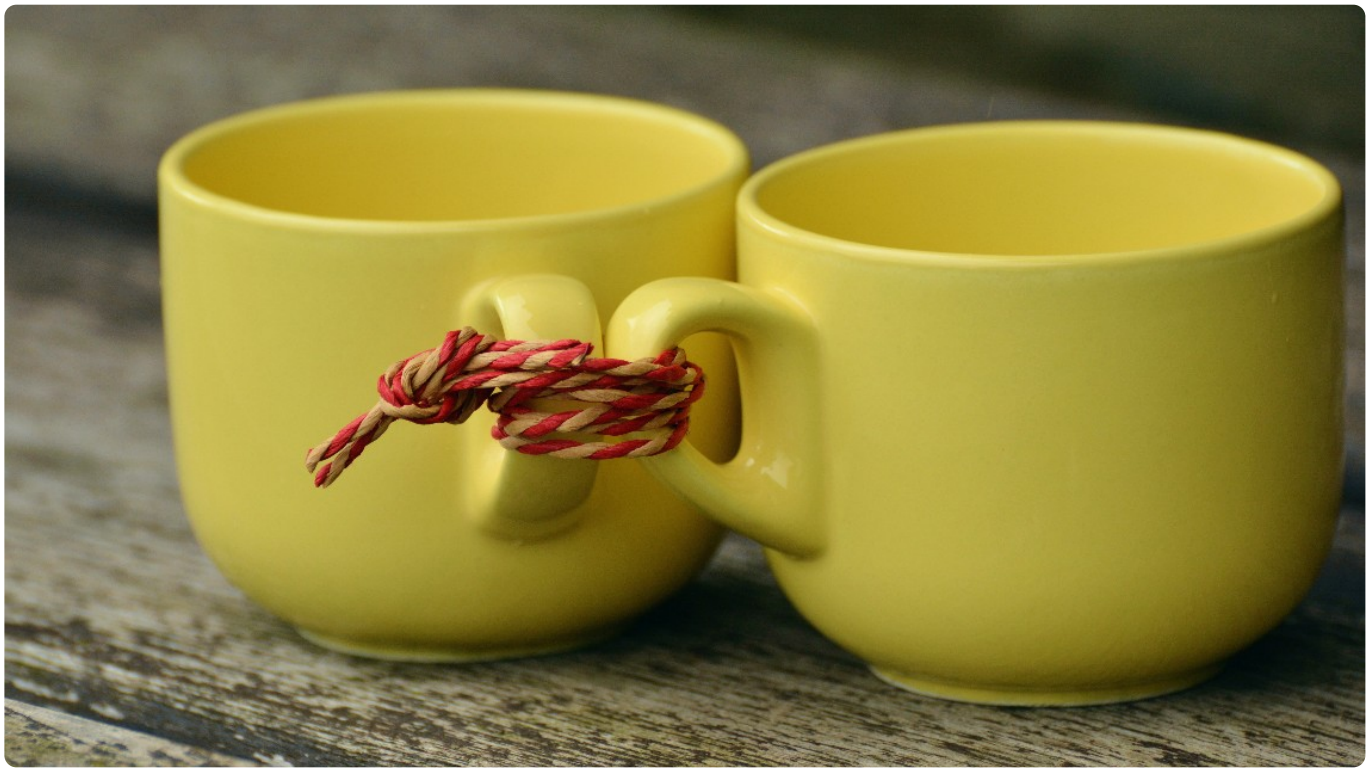
15 | 开源：从Phabricator的开源历程看开源利弊

2019-09-25 葛俊



讲述：葛俊
时长 16:38 大小 15.25M

你好，我是葛俊。今天，我来和你聊聊开源这个话题。

从克莉丝汀· 彼得森（Christine Peterson）1998 年提出“开源”这个名词，到今天已经 21 年了。可以说，在这些年里开源改变了软件开发世界。如今，开源覆盖了 IDE、移动端开发、前后端开发、运维、服务治理、AI 等众多领域的项目。比如，GitHub 上 2018 年最流行的前十个项目，包括 VSCode、React Native、Angular、Ansible、Kubernetes、TensorFlow 等，对这些领域都有覆盖。

项目名称	支撑公司/机构	星数
Microsoft/vscode	Microsoft	19K
facebook/react-native	Facebook	10K
tensorflow/tensorflow	Google	9.3K
angular/angular-cli	Google	8.8K
MicrosoftDocs/azure-docs	Microsoft	7.8K
angular/angular	Google	7.6K
ansible/ansible	Ansible	7.5K
kubernetes/kubernetes	Google	6.5K
npm/npm	NPM	6.1K
DefinitelyTyped/DefinitelyTyped	Microsoft	6.0K

从使用者的角度看，开源软件的价值不言自明。可以说，99% 的科技公司都在使用开源软件。

从贡献者的角度看，前十个项目中有 8 个项目的背后都有公司做支撑。毫无疑问，开源对公司来说也有吸引力的。但是，很多公司并没有开源，尤其是国内做开源的公司更是比较少，原因是什么呢？其实，就是因为开源有很多坑。

我在 Facebook 时参与了 Phabricator 开源的全过程，见证了其为公司带来的好处，比如因为模块化带来的代码质量提升、从开源社区获得的资源支持，也见证了开源的一些弊端，比如因为和开源社区目标不一致而带来的运维成本增加，以及最终导致的项目 Fork。

Phabricator 的整个开源过程

一般来说，开源一个项目的流程包括以下九步：

1. 公司 / 员工对某项目有开源的意愿；
2. 权衡利弊决定是否开源，以及后续的维护计划；
3. 法律和信息安全方面的审核；
4. 选择 License；
5. 选择 Contributor License Agreement；
6. 选择版本控制代码服务商（比如，GitHub、GitLab、BitBucket 等）；

7. 代码模块化，与公司代码分离；
8. 正式开源，发布信息；
9. 项目维护和持续开发。

接下来，我就以 Phabricator 的开源过程为例，帮助你理解公司进行开源的利弊，以及使用它来提高研发效能的一些原则和实践。

Facebook 决定对 Phabricator 进行开源

Phabricator 源自 Facebook 内部对代码审查的需求，后逐渐发展为软件开发的一个 Web 工具套件，包括代码审查、代码仓托管、缺陷跟踪、项目管理、团队协作等应用程序。

开源之前，Phabricator 主要由开发工具团队维护并增加新功能，其他开发人员也会向其贡献代码。它发展得非常快，为 Facebook 的开发和质量保障提供了很大帮助。但，Phabricator 有个问题就是，经常会出现严重的性能问题。具体来说就是，Phabricator 的速度会随着时间推移而逐步下降，每隔一年左右，就会达到让开发人员无法忍受的地步。

导致这个性能问题的主要原因有两个：

> 第一， Phabricator 和 Facebook.com 在同一个代码仓，共享 Facebook.com 的底层代码库，但开发人员对 Phabricator 的响应速度要求比用户对 Facebook.com 要高。
>
> 第二，非开发工具团队的开发人员在增加功能的时候，因为不了解全貌，所以在贡献代码的同时，往往会降低 Phabricator 的速度。一个功能降低的速度看不出来，但累积起来，总体速度的下降就很明显了。

所以每隔一年左右，我们就需要对 Phabricator 做一次重构，来提高响应速度。

2010 年年中的时候，这个性能问题再度爆发了，开发工具团队决定认真思考有没有更好的解决办法，从根本上解决这个问题。经过仔细分析，我们得出的解决方案就是和 Facebook.com 解耦。

正好这个时候，开源社区对 Phabricator 的代码审核功能非常感兴趣。我们认为开源 Phabricator 或许是一个可行的办法，同时调研结果显示开源 Phabricator 有以下好处：

1. Phabricator 是一个内部工具，不是面向用户的产品，开源非但不会影响公司的核心竞争力，还可以提高影响力。
2. 开源自然而然地就会把它从主代码仓剥离出来，实现与 Facebook.com 的解耦，实现提速。
3. 开源意味着代码从此要公开出去，更多的人可以看到。这就给 Phabricator 的开发人员带来压力，让他们更关注产品质量。这样一来，Phabricator 的性能就会更有保障。
4. 可以利用开源社区的开发资源。

当然，开源 Phabricator 也有缺点：

1. 开源之后，Phabricator 势必要支持更加通用的开发场景，这可能就会影响对 Facebook 特有场景的支持。
2. 开源之后，代码不会像在内部那样容易管控，灵活性会降低。

经过分析，我们认为可以使用插件的形式，从技术上解决对 Facebook 特有开发场景的支持问题。也就是说，在 Facebook 内部创建一个单独的代码仓作为插件，集成到开源的 Phabricator 之中。

而对 Phabricator 的代码管控问题，我们可以把它放到 Facebook 组织之下，从而保留比较强的管控力。

所以综合分析，内部工具团队以及上一级的基础平台团队，决定这一次的重构目标是开源 Phabricator。

开源准备工作

在确认了开源 Phabricator 之后，我们还要完成一些非开发的准备工作。

第一，法律和信息安全方面的审核。

这一步主要是确认此项目的开源，会不会使公司面临法律和信息安全方面的风险，由公司的律师团队和安全专家操作。一般来说，法律风险重点关注是否会泄露自己公司以及第三方公司知识产权；信息安全方面关注是否会暴露公司的安全漏洞。

第二，选择授权协议。

授权协议包括开源软件授权协议（Open-source License）和开源贡献协议两种。

其中，开源软件授权协议，指的是使用者享有的权利和受到的限制，比如 GPL、MIT、Apache 等协议。Phabricator 选择的是 Apache 2.0。这里，有两个工具可以帮助你做出选择，分别是“怎样选择开源协议？”和“开源指南”。

开源贡献协议，指的是对软件贡献者权力的限定，目的是赋予开发者对开源项目贡献代码的权力，并赋予项目管理者按照软件授权协议去发布软件。它包括 CLA（Contributor License Agreement）和 DCO（Developer Certificate of Origin）两种。Phabricator 选择的是 CLA。关于这个协议的选择，你可以参考“CLA 和 DCO 的区别”这篇文章。

因为具体选择哪个协议与法律有关，所以我只给出了参考链接，如果你的公司需要开源项目，推荐你去咨询律师。

第三，选择版本控制代码服务商。当时我们选择的是开源方面最流行的 GitHub。

开源具体步骤

完成了准备工作之后，剩下的就是正式的开发工作了。这部分工作主要包括以下三步。

第一步，把 Phabricator 代码和 Facebook 代码解耦。

我们做了一次比较彻底的重构，把分散在各处的代码，集中到 5 个代码仓里，分别是底层的 API 库 Libphutil、网站应用集 Phabricator、客户端 Arcanist、文档系统 Diviner 以及 Facebook 内部功能插件模块，完成了 Phabricator 的模块化。

第二步，进一步优化性能。

针对代码的性能，尤其是底层的 API 库，我们进行了很多优化。因为开源以后只需要支持通用的开发场景，所以我们不必考虑原来在 Facebook 代码仓的复杂调用，更容易去针对性地提高性能。

第三步，支持功能定制。

功能定制是开源的主要难点。除了解耦，我们还需要保证在解耦之后，仍然能够灵活地添加 Facebook 开发人员需要的定制需求。主要有以下三种方法：

Phabricator 提供对象的字段（Field）、类、库 3 个级别的扩展，我们主要采用库级别的扩展实现对 Facebook 的定制功能。

Phabricator 提供接口，供 Facebook 内部工具调用。

Facebook 内部工具代码提供接口，供 Phabricator 调用。

这样一来，我们就实现了 Phabricator 和 Facebook 其他内部工具的无缝集成。

除此之外，为了把 Phabricator 的部署从 Facebook 内部工具拆分出来，我们还需要完成以下工作：

数据库的迁移，即把 Phabricator 的相关数据从 Facebook 的数据库中迁移出来。

Phabricator 的部署。开源前 Phabricator 属于内部工具网站的一部分，所以不需要单独部署，但开源后我们需要给它重新设计和实现一套部署系统。

完成这些开发工作后，Phabricator 不仅从 Facebook 中剥离了出来，还显著提高了代码质量，比如模块化更好、注释更清晰、性能更好等。这些正是开源为 Facebook 带来的重要好处。同时，因为参与开源项目可以回馈社区并提升个人影响力，所以公司内部的 Phabricator 开发人员也都热情高涨。

但开源也意味着，我们需要投入额外的精力去实现 Phabricator 与其他内部工具的无缝集成，才不会影响 Facebook 开发者的使用体验。这，也是开源要付出的代价。

开源初期发展

完成开发工作后，Facebook 正式对外宣布了 Phabricator 的开源，同时正式切换到新部署的 Phabricator 集群。整个切换过程比较顺利，只是在一开始的时候，Phabricator 和其他工具间的联动出现了一些 Bug，修复之后就稳定下来了。

于是，Phabricator 也就开始进入开源的代码仓和内部的插件代码仓同时开发的阶段。针对 Facebook 的内部需求，我们尽量把它通用化，放到开源的代码仓中实现；实在需要定制的，才会放到 Facebook 的插件代码仓中。

这时，我的一位同事从 Facebook 离职，去了开源社区全职为 Phabricator 工作。他还创立了一家公司，致力于 Phabricator 的商用。于是，我们在开源社区也有了更强大的资源支持。

从 2011 年年初开源到 2013 年年底我离开 Phabricator 项目，Facebook 和开源社区对 Phabricator 的发展目标是一致的，所以一直在合力增加 Facebook 需要的功能，合作得非常好。总的来说，我们的确充分利用了开源社区开发者对 Phabricator 的贡献。同时，业界的很多著名公司开始使用 Phabricator，包括 Uber、Pinterest、Airbnb 等，提升了 Facebook 的声望。

Fork

2014 年开始，开源社区支持的公司越来越多，而它们的使用场景和 Facebook 不太一样，也就是说 Facebook 要想继续使用 Phabricator 的最新版本，就必须花费较大成本进行版本更新及维护。

而因为 Facebook 在 Phabricator 的使用上累积了非常多的数据，所以每一次数据库的 Shema 变动，都会带来非常麻烦的数据迁移工作，常常需要 DBA 的帮助才能实现不中断服务的版本更新。

考虑到这些新增功能对 Facebook 用处不大，而维护的成本又很高，所以 2014 年下半年 Facebook 决定停止使用外部开源的 Pabricator，重新在公司内部自己维护一套 Fork 的 Phabricator 代码。这样一来，开源版的 Phabricator 引入新功能的时候，Facebook 只在需要的情况下，才会参考外部的实现在内部引入。

其实，公司和开源社区目标不一致的现象比较普遍。在我看来，这可以算是开源项目的第一大坑。Facebook 对 Phabricator 采取的措施是内部 Fork，让开源社区继续自由发展，既然不能从开源社区得到资源，就把代码挪回公司内部获取完全的管控和自由度。这是处理目标不一致问题的第一种方法，你也可以借鉴。

第二个办法是，对代码仓强管控，但结果往往是开源社区 Fork 一个新项目重起炉灶，和第一种方法的结果其实差不多。

除了 Fork 之外，还有第三种办法，就是采用不同的分支来支持不同的目标。这样的好处是，公司依然可以获得开源社区的资源支持，坏处是分支管理、版本管理繁琐，也缺乏 Fork 的灵活性。

以上，就是 Facebook 开源 Phabricator 到最终 Fork 的全过程。我在这其中讲述了 Facebook 处理具体开源问题的一些方法，你也可以借鉴到自己的项目中。

开源对公司的利弊

这里，为了帮助你加深理解，我把开源对一个公司的利弊做了总结整理，如下表所示。

开源的好处	1. 提高代码质量。一是，开源通常需要把代码进行模块化；二是，代码开源之后会面对外部社区，自然会激发开发者更注意代码质量。
	2. 得到开源社区的免费帮助。
	3. 提高程序员积极性。开发人员一般都愿意在开源项目工作，对个人成长和在开源社区的知名度都有好处。
	4. 提高公司声望。对Facebook来说，提高声望最大的好处在于招聘方面，可以增加对人才的吸引力。
	5. 回报开源社区。
开源的缺点和挑战	1. 定制有挑战。没有开源的时候，公司对代码有完全的掌控权，同时可以很方便地与原有系统集成。
	2. 内外协调有挑战。我在Phabricator项目时，因为开源社区的对接人是关系很好的老同事，所以当时协调的风险对我们来说不是很大。
	3. 开源社区和公司对项目发展目标不一致。

总的来说，我认为开源在以下两种情况下最为有利：

第 1 种情况是大公司。不难发现，上面列举的各种好处对大公司比较明显，提高公司声誉就是典型例子。所以，2018 年 GitHub 前十名开源项目中，除了 NPM 和 Ansible 外，其他的 8 个项目都是由 Microsoft、Facebook、Google 三个大公司支撑。

第 2 种情况，需要通过开源获取影响力从而扩展业务的公司。比如，Docker 公司在开源 Docker 项目之前，名气并没有多大，但是把 Docker 项目开源之后，一下子就变成了明星企业，在改变了 Pass 发展格局的同时，也改变了自己的命运。

另外，从适合开源的项目的角度来看，平台、基础设施、工具等（比如，Phabricator 以及 2018 年 GitHub 前十名开源项目）适合开源，而业务层的项目因为通用性不强，不适合开源。

小结

开源正在改变软件开发的格局，选择开源自己的项目对公司来说也是有利有弊。所以，今天我以 Phabricator 的开源过程为例，和你分享了开源一个项目涉及哪些步骤，在这其中获得的好处以及需要付出的代价。

开源对公司的好处，主要表现在提高代码质量、得到开源社区的免费帮助、提高开发者的积极性、提高公司声誉、回报社区等。而缺点和挑战，主要包括定制困难、内外协调，以及版本维护等。如果你的公司或者团队在考虑是否开源，可以将这些利弊作为参考。

在我看来，Phabricator 算是开源的一个成功案例。因为在整个过程中，我们充分利用了开源带来的好处，而且在开源过程中投入的开发资源，即使不开源也是需要的。

一定程度上讲，开源 Phabricator 的过程，也体现了 Facebook 的实用主义。在需要开源的时候，放手去开源；在发现维护的性价比不好时，就果断 Fork。虽然从个人情感的角度说，我不愿意看到 Phabricator 在 Facebook 内部最后 Fork 了，但理智地看这的确是一个很好的决定。

思考题

跟硅谷相比，国内公司参与开源的非常少。你觉得主要原因是什么，将来的趋势又会是什么样的呢？

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再见！

上一篇　14 | 质量与速度的均衡：让“唯快不破”快得更持久

下一篇　16 | 高效上云：如何用云计算来提高效能?

精选留言 (5)

写留言

Geek_1988
2019-09-25

我想到以下两点原因：
1.目前国内公司业务型的代码较多，平台、工具型的代码较少，所以不适合开源
2.并且国内版权环境较差，开源代码容易被窃取，弊大于利

不过情况正在改善，华为，百度，腾讯，阿里等大公司已经开源了不少代码，为开源社...
展开

作者回复: 分析得不错!

2

李双
2019-09-25

学习

展开

1

兴国
2019-09-27

1. 绝大多数公司都会先以存活下来，盈利为目标。
2. 大部分公司会把代码看的比较重，是公司发展的核心竞争力
3. 整体的氛围和意识还没有达到

不过目前开源的原来越多了

展开

作者回复: 是这样的!

刘丹
2019-09-27

做开源项目需要投入。国外很多开源项目都有企业或基金赞助，开发者有时是赞助企业的正式员工，业余参加者也是有工资的。国内因为起步晚等原因环境不同，中小型企业在解决生存问题之前，基本都是索取为主；少数大型互联网公司没有温饱问题，在逐步参加或开源一些项目。

展开

作者回复: 分析得很全面。国外的确很多开源有企业或者基金赞助。而且开源贡献者如果做的不错会很受欢迎。举一个明显的例子就是git方面的专家。我亲耳听过一个代码托管起家的公司高管很自豪的跟我说“开源社区的几个主要贡献者有几个在我们公司，其他公司没法比”之类的话。

日拱一卒
2019-09-26

开源一般是公司发展到一定规模后会去考虑的事情，其中工程能力、代码质量、版权意识都可能阻碍公司去开源的因素。

展开

作者回复: 是的。小公司一般是因为开源跟业务开展有直接关系