目　录

致谢

当前文档 《github 开源贡献指南中文版本》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-05-01。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/opensource-contribute-guide-chinese

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

介绍

这里是 `github opensource guide` 的中文版本

怎么给开源项目做贡献

为何要给开源项目做贡献？

> 给*[freenode]*做贡献帮助我学到了很多后来在大学和实际工作中用到的技能，我觉得给开源项目工作对我的帮助和对项目本身的帮助相差无几。

——@errietta [“Why I love contributing to open source software”]

给开源项目做贡献可以说是在你能想象的领域上学习，传授，累计经验的最有效的方式！为什么人们要给开源项目做贡献，原因太多了！

提高现有技能

不管是写代码，用户界面的设计，图形设计，写作，或者是组织，如果你想找点练习做一做，在开源项目上你总能找到能胜任的任务。

认识和你有同样爱好的人

气氛融洽开放的开源项目会让人数年之后仍然不忘回来看看（项目进展）。许多人在参与开源项目的过程中结识了一生挚友，友谊在开会的互相照面
和深夜的线上闲聊中渐渐形成。

寻找导师或者教导别人

和他人一起合作一个项目意味着你得解释你是怎么做事情的，同时寻求他人的帮助。学习和传授知识的体验对每个参与其中的人来说都是令人愉快的体验。

做一个公开的产品帮你赢得声誉（和职业机会）

从开源项目的定义可以知道，你所有的工作成果都应该公开，这意味着你免费获得了一个想众人展示你能力的机会。

学习他人的技能

开源项目给参与其中的人们提供了锻炼领导力和管理能力的机会，比如解决冲突，组织团队的成员，辨别工作的轻重缓急。

你有权做出改动，就算是很小的

你不需要成为那种一直在给开源做贡献的人。你有在网站上看见手误吗，而且希望有人能修正它。在一个开源项目中，你自己就可以做到。开源帮助人们在生活和
对世界的体验上感觉到更有力量，这本身确实是意见可喜的事情。

贡献是什么意思

> 如果你是一个刚刚开始的开源贡献者，这个过程可能会让你觉得很吓人。如何找到正确的项目？你不知道怎么贡献代码怎么办？如果改错了怎么办？

不必担心！有很多参与开源项目的方法，和一些让你走出困境的小技巧。

你不需要贡献代码

对开源贡献的一个普遍的误解就是你得贡献代码。实际上，通常和代码无关的部分才是最容易忽视的。通过参与非代码部分的贡献会给项目带来巨大的帮助。

> 我因为对 *CocoaPods*的贡献而著名，但是大部分人都不知道我在这个工具本身的开发上并没有做实质性的工作。我花在这个项目上的主要时间都用来整理文档和宣传品牌。

——-@orta

即使你是一个开发者，非代码部分的贡献是一个很好的方式让你参与一个项目和认识社区的成员。建立这种关系会给你从事项目其他部分工作的机会。

> 我第一次接触 *Python* 开发团队（又叫做 *python-dev*）是在*2002*年*6*月*17*号我给邮件列表发邮件让他们接受我的补丁的时候。我迅速的发现的这个项目的缺陷，并决定负责组织团队的邮件摘要。这给了我一个很好的机会去询问他们对一个话题的解释，但是实际上更关键的是当有人指出什么的时候我能意识到那是不是需要修复的 *bug* 。

——-@brettcannon

你喜欢策划活动吗?

- 组织关于项目的研讨会或者线下聚会，就像 @fzamperin
- 组织项目会议（如果他们有的话）
- 帮助社区成员找到合适的会议主题并提交一份发言建议。

你喜欢设计吗？

- 重构项目的布局以增加其易用性
- 组织用户使用调查来重构项目的导航或者菜单
- 把样式指南放在一起以此来帮助项目有一致的视觉设计
- 设计 t-shirt 或者 新的logo，就像是hapijs的贡献者们做的一样

你喜欢写作吗？

- 编写或者改善项目的文档
- 创建一个文件夹展示项目怎样使用的例子
- 给项目编写教程，就像pypa的贡献者们做的一样
- 为项目的文档做翻译

> 认真的说，*[*文档*]*真的是重要得一逼。目前*Babel*的文档已经很棒了，这也是其杀手锏的特性之一。当然，还有一些章节需要大家的完善，即使是随便在哪儿增加一个段落都很感激。

你喜欢组织吗？

- 连接重复的 issue，或者为某个iuuse添加新的标签来让事情变得井然有序
- 检查打开的 issue，就像@nzakas给eslint做的那样
- 在最近打开的 issue 中问一些解释性（原文是 clarifying questions的问题使讨论向前推进

你喜欢写代码吗？

- 找一个 issue 去解决。就像@dianjin给Leaflet做的那样
- 询问项目所有者你是否可以帮忙写一个新的功能
- 使配置项目的过程自动化
- 改善工具链和测试

你喜欢帮助别人吗?

- 在诸如 Stack Overflow（这里有一个关于Postgres例子，或者reddit上回答关于项目的问题
- 在打开的 issue 中回答人们的问题
- Help moderate the discussion boards or conversation channels(待翻译)

你喜欢帮助改善别人的代码吗?

- 审查别人提交的代码
- 写一个关于项目如何使用的教程
- 帮助其他的贡献者，就像在Rust项目上@ereichert为@bronzdoc做的那样

你不一定只能给软件项目做贡献！

虽然开源一般指的是软件项目，实际上你可以在任何项目上进行协作。有很多书籍，经验贴，列表，课程都是开源项目。
比如:

- @sindresorhus策划了 awesome 系列的列表
- @h5bp持有一份关于前端常见面试问题的列表
- @stuartlynn 和 @nicole-a-tesla制作了一份关于[海雀的有趣现象集锦]

即使你不是你个软件工程师，给一个文档性质的项目做贡献也会让你迈进开源世界的大门。在没有代码的项目上做事通常没那么吓人（相比与有代码的项目来说），而且这个写作的过程会让你积累更多自信和经验。

投身于一个新项目

> 如果你打开一个项目的 *issue tracker* 。里面的东西可能让你觉得不解，不只是你有这样的感觉。这些工具需要很多的隐性知识，但是人们会帮助你搞清楚，你也可以问他们问题。

——@shaunagm “How to Contribute to Open Source”

对于那种不仅仅是修复一个手误的工作，给开源项目做贡献就像是在一个聚会上走近一群陌生人一样。当他们正在热火朝天的讨论金鱼的时候，你插进去开始讲骆驼,他们会像你投来异样的眼光。在你带着你的见解盲目的加入讨论之前，首先研究一下他们到底在讨论什么。这样会增加你的观点被注意到和听取的机会。

分析一个开源项目

每一个开源社区都不一样。

在一个项目上花费数年的事件会让你对这个项目了如指掌。但是当你迁移到另外一个项目中时，你会发现他们的词汇，规范和讨论的风格完全不一样。

话虽如此，很多开源项目还是遵循一个相似的组织结构。理解不同社区的角色和总体的进程会帮助你很快的融入一个新的项目。

一个经典的开源项目会有这样几类人：

- 作者： 创建该项目的人或者组织
- 所有者： 对该组织或者仓库有行政权的人（通常和原始的作者不是一个人）
- 维护者： 那些负责宣传项目，管理项目组织的贡献者（ 他们也可能是作者或者所有者）
- 贡献者： 每个给项目做出或多或少贡献的人
- 社区成员： 使用项目的人。他们可能在关于项目方向上的讨论中积极发表自己的观点

更大型的项目可能有针对不同工作的子社区或者工作组，比如工具链，工作分配，打造社区的舒适度和事件管理。查看项目网站上的“团队”页面，或者存放管理文档的仓库寻找这些信息。

一个项目也会有他自己的文档，这些文件放在项仓库目的一级目录。

- LISENCE：由于开源项目的定义，每个开源项目都要有一个开源协议。如果一个项目没有一个开源协议，那么他就不是开源项目。
- README：README文件是给社区的新成员的使用手册。它解释了为什么这个项目是有用的和怎么开始使用这个项目。
- CONTRIBUTING：READMEj文件是帮助人们使用项目的，而CONTRIBUTING文档是帮助人们对项目做贡献。但是不是每个项目都有CONTRIBUTING文件，那么有这个文件就标志着这是一个开放的项目。
- CODE_OF_CONDUCT：行动守则制定了参与者行为的基本规则，帮组促进了社区的友好，开放的氛围。但是不是所有项目都有 CODE_OF_CONDUCT 文件，如果有的话那就标志着这是一个开放的项目。
- Other documentation：还可能有其他的文档，比如教程，预览，或者管理政策，尤其是在大型项目中会出现。

最后，开源项目使用下面这些工具来管理讨论。在你阅读本文的过程中，你会对开源社区思考和工作的方式有一个总体的映像。

- Issue tracker: 人们用来讨论和项目相关的问题的地方
- Pull requests: 人们用来讨论和审查正在进行中的修改。
- Discussion forums or mailing lists（论坛或者邮件列表）： 有些项目会用不同的频道对应不同的讨论主题（比如说，“我怎样才能…” 或者 “你们对于…的看法是”，而不是 bug 报告或者功能请求。另外一些项目
  直接用 Issue tracker 进行所有话题的讨论。
- Synchronous chat channel（匿名的聊天频道）：有些项目用聊天频道（比如Slack或者IRC)来进行随意的讨论，合作，或者快速的修改。

找一个项目来做贡献

> 现在你一*ing*知道开源项目是怎么工作的了，是时候找个项目然后开始贡献了！

如果你从来没有给开源项目做过贡献，那么从美国前总统约翰·肯尼迪的名言之中吸取一点建议吧：

> 不要问你的国家能为你做什么，先问问自己你能为自己的国家做什么。

给开源项目做贡献可以发生在任何级别的任何项目。你不需要过分在意你的第一次贡献会是什么，或是以什么形式。

相反，从你已经在用的项目或者你想用的项目开始。你贡献最积极的项目正好是那些你发现你会是不是来看一下的项目。

在那些项目中，尽管释放你的本能，去发现那些你觉得可以做的更好或者做的不同的东西。

开源世界不是一个排他性的俱乐部，它正是有那些像你一样的人创造的。“开源组织”是一个把世界上所有问题看成可以解决的梦幻之地（此处待重译）

你可以浏览一下项目的 README 文档，找找有没有挂掉的链接或者手误。或者你是一个新用户，而且你发现什么了东西不对，或者一个你觉得应该放在文档中的 Issue ，与其直接忽视或者找别人修复它，还不如自己动手把他改过来。这就是开源的含义啦！

> *28%*的不固定的贡献者所做的都是在文档上，比如更正手误，重新排版或者提供一种语言的翻译版本。

你还可以用以下的资源来帮助你寻找项目。

- GitHub Explore
- First Timers Only
- Your First PR
- CodeTriage
- 24 Pull Requests
- Up For Grabs

一个在你贡献之前的清单

当你发现了一个你想要贡献的项目的时候，对项目做一快速的浏览来保证这个项目适合接受你的贡献，否则你的工作得不到应有的回应。

这里提供了一份评估一个项目是否适合新的贡献者的清单

检查开源的定义

- [ ] 他有一份开源协议吗？通常情况下是一个在项目根目录下的叫 LISENCE 的文件。
项目接受贡献者的活跃程度

查看 master 分支上的提交活动。在github上，你可以在仓库的主页上看到这个信息

- [ ] 最近一次提交是什么时候

- [ ] 项目目前有多少贡献者

- [ ] 人们提交的频率是怎样的？（在 Github ，你可以通过点击顶部的 “commit” 来找到。

接下来，查看项目的 issue 。

- [ ] 目前有多少 issue 。

- [ ] Do maintainers respond quickly to issues when they are opened?
项目维护者对打开的 issue 回复的速度如何？

- [ ] 在 issue 中的讨论是否热烈。

issue 都是在最近的吗？

issue 被关闭了吗（在 Github ，在 issue 页面点击 “closed” 标签查看关闭的 issue 。

对项目的 pull request 做同样的检查。

目前有多少 pull request？

项目维护者对打开的 pull request 回复的速度如何？

在 pull request 中的讨论是否热烈？

pull request 都是最近的吗？

最近一次的 pull request 被合并是什么时候？（在 Github ，在 pull request 页面点击 “closed” 标签查看被关闭的 pull request。

项目是否足够开放

如果一个项目是友好和开放的那么意味着他们很乐意接受新的贡献者。

项目维护者对 issue 中的问题的回复时候有帮助？

在 issue ，论坛，聊天室（比如 IRC 或者 Slack）中的人们是不是乐于助人。

pull request会被审查吗？

项目维护者对贡献者的 pull request 表示感谢了吗？

不管何时当你看到核心开发者做出的长篇大论式的，总结性的发言。不放思考他们总结是建设性的吗？而且在保持礼貌的同事一步一步把讨论引向一个结论。如果你看到了讨论过程中出现摩擦，那么让人叹息的是他们把精力浪费在了争吵而不是开发上面。

— @kfogel, *Producing OSS*

如何提交贡献？

假如你已经找到了一个你喜欢的项目，而且你已经准备好做一次贡献。终于！是时候谈谈怎么正确做出贡献啦！

高效率的沟通

不管你是一个一次性的贡献者还是想要加入社区，和他人合作是你在参与开源项目过程中会培养的一项重要技能。

[作为一个新的贡献者]，我很快意识到如果我想关掉 issue 的话我得问一些问题。我浏览了一下代码架构，当我对项目有了基本的把握之后，我便询问我下一步该做什么。最后，当我了解了我所需要的所有细节之后，我能够解决那个 issue 了。

— @shubheksha, A Beginner's Very Bumpy Journey Through The World of Open Source

在你打开一个 issue 或者 发起一个 pull request 或者在聊天室问一个问题之前，把下面这些要点记清楚以此来更好的表达你的想法。

给出上下文帮助人们快速了解你提出的东西。如果你遇到了一个问题，解释你想做什么和怎样重重现该问题，如果你是在表达一个新的想法，解释一下为什么你觉得对项目来说这个想法是有用的（而不仅仅是对你而言）

> ? "当我做甲的时候，乙为什么不出现"
>
> ? "这个啥啥啥出问题了，麻烦修复它"

提前做好功课 无知是没问题的，但是告诉别人你已经尽力了。在寻求帮助之前，一定要先看看 README 文件，文档，issue（开着的关着的都要看），邮件列表，在网上也找一找。当你展示除了一种想要学习的态度的时候别人会很乐意帮助你。

> ? "我不确定怎么实现这个，我查看了帮助文档但是没有找到相关的内容"

> ? “我怎样做才能啥啥啥”

保持你的请求简短清晰。就像是发邮件一样，每一次贡献，不管是多么简单或者多么有帮助，都需要有人审查。很多项目提问的数量远远多于提供帮助的人。保持简洁，你会增加别人帮助你的概率。

> ? “我想写一个 *API* 使用教程”
>
> ? “当我有一天下高速加气的时候突然想到了关于我们正在做的事情的一个牛逼的点子，在我解释之前，让我先展示…”

保持所有交流都是公开的 就算私戳项目的维护者是很诱人的，但是除非你要分享一些敏感信息（比如一个有关安全的 issue 或者是严重违反守则的行为，否则就不要这么做。当你让对话保持公开，更多人可以从你们的对话中学到更多。讨论本身也是一种对项目的贡献。

> ? (对于评论)”@-maintainer 你好！我们应该怎么处理这个 *PR?*”
>
> ? (对于邮件) “你好, 不好意思通过邮件打扰你，但是我想问一下你是否有时间审查一下我的*PR*”

可以问问题（但是一定要耐心！） 从某种程度上来说，每个人都是某个项目的新人，即使是对于有经验的贡献者，当他们刚开始接触一个项目的时候也要费点力气。同样，即使是长时间维护项目的人也不是对项目的所有细节都了如指掌。如果你想让他们对你有耐心的话你首先得对他们有耐心。

> ? “麻烦你看一下这个错误。我采取了你的建议！这是输出。”
>
> ? “为什么你没解决我的问题，这不是你的项目吗？”

尊重社区的决定 你的想法可能和社区优先考虑的事情或者说看问题的视角不一样。他们可能会给你反馈或者拒绝你的想法。然而你应该和他们讨论然后寻求妥协，维护者会比你在决定方向上话费更多时间，如果你不同意他们的方向，你可以一直在你 fork 的仓库上工作，或者自己创建一个新项目。

> ? “你没能支持我想要的特性我很失望，但是就像你解释的那样，它只会对一部分的用户游泳，我知道为什么。感谢你聆听我的建议”
>
> ? “为啥那么你不支持我的需求呢？这简直没法儿接受！”

总之，保持优雅的状态 开源项目是由来自全世界的协作者一起创造的。这意味着开源协作的背景是多语言，多文化，跨地理位置，跨时间区的。除此之外，用键盘敲出来的文字无法传达音调和情感。所以在交谈中要呈现出善意的一面。礼貌的在一个想法上表达不同看法，或者询问更多细节，表明自己的立场，都是可以的。努力让网络空间变得更美好。

收集背景信息

在你做任何事之前，快速检查一下你的想法还没在别处被讨论过。浏览项目的 README 文件，issues（开着的关闭的），邮件列表， Stack Overflow。你不需要话费数小时去浏览所有的信息，但是对一两个关键词的快速搜索也会大有帮助。

如果你在别的地方找不到你的问题，你就可以搞事情了。如果项目是在 Github 上的，以可以通过开 issue 或者发 pull request 和别人交流。

- **Issues** 就是发起一次交谈或者对话。
- **Pull requests** 验证某一种解决方案

- **For lightweight communication**，比如一个 clarifying or how-to question(待翻译)，在 Stack Overflow 上提问， IRC， Slack，或者其他的聊天频道，如果你所在项目有的话。

在你开 issue 或者 pull request 之前，查看项目的贡献文档（通常是一个叫 CONTRIBUTING 的文件，或者就在 README 里面），看看里面有没有你要的信息。举个例子，他们可能会让你遵照一个模板，或者要求你包含一个测试。

如果你想要做做一次比较大的贡献，先开一个 issue 问一下。最好是 watch 这个项目（在 Github 上，你可以点击 “watch” 这样你可以接受所有对话的通知），然后认识一下社区的成员，因为你的工作并不会被他们接受。

You’ll learn *a lot* from taking a single project you actively use, “watching” it on GitHub and reading every issue and PR.

— @gaearon on joining projects

开一个 Issue

在以下的情况你就应该开一个 issue

- 报告一个你自己解决不了的错误

- 讨论一个高级别的话题或者想法（比如社区，vision（这个自己体会。。），政策）
- 提出一个新功能或者其他的关于项目的想法

在 issue 中交流的小贴士:

- 如果你看到了一个开着的 **issue** ，而且你想解决他 在 issue 中评论让人们知道你在尝试解决他，这样别人就不会重复你的工作了。
- 如果一个 **issue** 才打开片刻，有可能这个 issue 别人已经在解决了，或者早就已经搞定了，所以在你开始动手之前在相应 issue 的评论里面问一下比较好。
- 如果你打开了一个 **issue** ，但是最后自己解决了， 在 issue 的评论里面告诉别人，然后关掉这个 issue 。甚至以文档的形式把你的成果展示出来也是对项目的一种贡献。

发一个 pull request

在以下的情况下你就可以发一个 PR 了。

- 提交一个小问题的修复（比如手误，挂掉的链接，或者明显的错误）
- 准备实现一个早就有人提过的需求，或者是解决在某个 issue 中讨论的问题

一个 pull request 不需要是现在已经搞定了的工作。通常最好是在这之前就发起开一个 pull request ，这样别人可以查看你的工作情况，或者对你现在的进度给予反馈。只要在标题行打上 WIP （正在进行中）的标签就行了。你可以稍后添加更多的信息。

如果项目是在 Github上的话，这里展示了如果提交了个 pull request:

- 复刻仓库 然后克隆到本地。通过把它添加到 remote 就把你的本地仓库和远程的“上游”仓库链接起来了。要经常从你的“上游仓库”拉取代码以此来保证同步从而当你提交你的 pull request 的时候，合并冲突就变得更容易了。（从这里查看更多教程
- 创建分支 用来编辑代码。
- 引用任何相关的 **issue** 或者在你的 PR 顺便附上相关信息（比如 “关掉 issue #37”）
- 包括修改之前和之后的截图 如果你的改动包含 HTML/CSS 文件。拖放图片到 pull request 的正文。
- 测试你的改动！ 在你的改动上运行已经存在的测试，有必要的话创建新的测试。不管之前有没有测试，都要保证你的改动不会破坏项目已有的功能。
- 按照项目的风格改动 将你的能力发挥到最好。这意味着会使用和你自己仓库不一样的缩进，分好和注释风格。但是这会方便维护者合并，其他人在以后也好理解。

当你提交一次贡献的时候发生了什么

你做到了！恭喜成为一个开源贡献者。而我们希望这仅仅是开始！

当你提交你的 PR 之后，可能会发生以下几种情况。

？ 你并没有得到回应

在你做贡献之前你还满怀希望的检查了标志项目活跃的要求。即使是一个活跃的项目，然后还是有可能你的贡献得不到回应。

如果你在一周之内都没有得到回应，你可以有礼貌的找别人帮你审查。如果你知道帮你审查贡献的合适人选的名字，你可以@他们。

不要私戳那个人；要时刻记住公开交流对开源项目来说是必要的。

如果这样还没人理你，那么就可能不会有人理你了，永远。这让人感到不爽，但是别因为这打击到你。每个人都可能会遇到这种情况！你没得到回复的原因有很多，包括你不能控制的私人原因。尝试着找另外一个项目做贡献。总之，在社区其他人还没参与和相应进来的时候你就不要话太多的事情在某个问题上面。

？ 有人想改动你的PR

被要求改动你的贡献是很常见的，要么是对你的想法，要么是对你的代码。

当有人想改动你的PR的时候，务必回复！因为他们花时间审查了你的代码。你开个PR就跑路是不好的！如果你不知道怎么改，好好研究一下问题所在，如果需要的话可以寻求帮助

如果你没时间再搞某个 `issue` 了（举个例子，如果对话已经过去数月了，而且你的情况也有所改变），让维护者知道从而他们就不会等着你的反馈了。可能另外某个人会开心的接手你的工作。

？ 你的贡献被拒绝了

到最后你的贡献不一定会被接受。如果你也没在这上面花太多功夫那是最好，如果你不确定为什么没有接受，你有完美的理由去询问维护者给你反馈和解释。不要争吵或者怀恨在心。如果你不认同它的看法你大可 `fork` 一份搞自己的版本。

？ 你的贡献被接受了！

万岁！你已经完成了一个开源贡献！

你做到了！

不管你是已经完成了你的第一个开源贡献还是在寻找贡献的新途径，我们都希望你可以立即行动起来。即使你的贡献可能不会被接受，但是不要忘记当维护者花时间帮助你的时候要说声谢谢。开源世界就是由无数像你这样的人创造的：一个 `issue` ，`pull request`，评论和每一次的庆祝。

创建一个开源项目

什么是开源，为什么要开源

那么你正准备拥抱开源吗？恭喜你，开源世界欣赏你的贡献。接下来让我们聊聊什么是开源，我们为什么要开源。

“开源”意味着什么？

当一个项目开源后，意味着 不论什么目的，所有人都可以浏览，使用，修改和分发你的项目。 这些权限都是来自于开源协议.

开源非常的强大。因为它降低了使用的门槛，使新奇的思想得到快速的传播。

来理解它如何工作，想象下你的朋友正在吃便当，这时你带来了樱桃派。

- 每个人都会想要樱桃派（使用）
- 这个派引起了一场轰动！周围的人会想知道你的烹饪方法（浏览）
- 有一位朋友Alex是一名糕点师，他会建议少放一点糖（修改）
- 另外一位朋友Lisa要求使用它作为下个星期的晚餐（分发）

同样的，闭源就像是你去餐厅必须付钱才能吃樱桃派。但是，餐厅不会告诉你樱桃派的烹饪方法。如

果你恰好抄袭了他们的派，并以你自己的名义出售，那么餐厅将会采取行动抵制你。

人们为什么要将他们的工作开源？

我从开源使用和协作中获得的最有价值的经验之一来自我与其他面临许多相同问题的开发者建立的关系。

— @kentcdodds, “How getting into Open Source has been awesome for me”

这里列举了很多理由 来解释为什么个人或者组织想要开源自己的项目。下面列举了部分：

- 协作： 开源项目欢迎所有人参与。例如， Exercism是一个有超过350人协作开发的练习编程的平台。
- 采用和重新混合：
  任何人可以出于几乎任何目的使用开源项目。人们甚至可以将开源项目用于构建其他的项目。例如， WordPress是基于开源项目 b2构建的。

- 透明度： 所有人都可以检查开源项目中存在的问题。透明度对于政府（如 @bozhobg/bulgaria-got-a-law-requiring-open-source-98bf626cf70a">保加利亚或者 美国），产业调整（如银行业或者医疗健康行业），和软件安全（如 Let's Encrypt）。

  不仅仅是可以开源软件，你可以开源一切，从数据集到书籍。通过浏览 GitHub Explore 你可以知道什么东西可以被开源。

开源是否意味着免费?

开源最吸引之处就是它不用花钱。然而免费只是开源的价值的一个副产品。

因为 开源协议要求开源项目可以被任何人出于几乎任何目的使用，修改和分享，这些项目一般都是免费的。如果有些开源项目需要付费使用，任何人都可以合法地使用其免费版。

结果是大多数开源项目都是免费的。但免费并不属于开源定义的一部分。开源项目可以通过双重许可协议或者其它的方法进行间接收费，同时不违背开源的官方定义。

我应该发起属于自己的开源项目吗?

答案是肯定的，因为不论结果是什么，发起一个属于自己的开源项目是学习开源最好的方法。

如果你还没有开源过一个项目，你可能会因为没有人关注或者别人的说辞而紧张。如果真是这样的话，你并不孤独！

开源与其他有创意的活动是一样的，无论是写作还是画画。你可能会害怕向世界分享你的工作，但练习是唯一让你变得更好的方法，即使你没有一位听众。

如果你不确信，那么请花一点时间想想你的目标可能是什么。

设定你的目标

目标可以帮助你知道该做什么，不因该说什么和需要从他人那里获得哪些帮助。请开始问自己，我为什么要开源这个项目？

这个问题没有一个正确的答案。你可能为一个简单的项目设定了多个目标，或者不同的项目有不同的目标。

如果你唯一的目的是炫耀你的工作，你可能甚至不想将它贡献出去，甚至不会在README中说明。另一方面，如果你想贡献自己的项目，你将会花更多的时间在书写简洁明了的文档上，使新来的参与者感到欢迎。

在某一时刻，我创建了一个自定义的UIAlertView，并且决定开源。因此我对进行了一些修改使其更加动态灵活，同时上传到GitHub。我编写了一份技术文档以便其他开发者将UIAlertView用于他们的项目中。或许没有人使用这个项目，因为这是一个简单的项目。但是我为自己的贡献感到开心。

— @mavris, “Self-taught Software Developers: Why Open Source is important to us”

随着你的项目的发展，你的社区可能不仅需要你提供的代码。回复issues，审查代码和传播你的项目在一个开源项目中都是非常重要的任务。

虽然你花费在非编码上的时间取决于项目的规模和范围，但你应准备好作为维护者来自己解决问题或者向他人寻求帮助。

如果你参与了公司的开源项目， 确保你的项目拥有它所需要的内部资源。当项目启动后，你会想知道由谁负责维护和在你的社区如何分享这些任务。

如果你需要为项目的宣传，操作和维护准备一笔专用预算或者人员配置，那么尽早开始讨论。

一旦你开源项目后，最重要的是你要考虑到项目周围社区的贡献和能力。你不必担心一些不是你公司的贡献者参与到项目的关键部分。

— @captainsafia, “So you wanna open source a project, eh?”

为其他的项目做贡

如果你的目标是想学习如何与他人一起协作或者了解开源是如何工作的，那么你可以考虑为一个已存在的项目做贡献。开始参与你曾经使用过和喜爱的项目。为项目做贡献就像修改错别字或者更新文档一样简单。

如果你不知道如何开始做一个贡献者，那么可以阅读我们的Github开源项目贡献指南。

发起属于你的开源项目

如果没有充足的时间来开源你的工作，你可以开发一个想法，一个正在进行的工作或者多年后将被关闭的资源。

一般来说，当你发现有人对你的工作反馈了一些有建设性的观点后，你应该开源你的项目。

无论你决定开源你项目的哪个阶段，每个项目都应该包含这些文档：

- opensource license
- README
- opensource guidelines
- code of conduct

作为一名维护者，这些组合将会有助于你表达想法，管理贡献和保护每个人的合法权益（包括你自己的）。他们大大增加了你获得积极经验的机会。

如果你的项目在GitHub上，将这些文件按上面推荐的命名方式放在你的根目录，这样对你的读者会一目了然。

选择协议

开源协议可以保障他人对你的项目进行使用，复制，修改和贡献时不会产生影响。它还保护你免受法律的困扰。当你发起一个开源项目时必须选择一个协议。

法律工作很乏味。好消息是你可以在你的仓库中使用一个已经存在的开源协议。这样你只花了很少的时间，但很好的保护了你的工作。

MIT, Apache 2.0, and GPLv3 都是非常流行的开源协议，但是 还有其它的开源协议 可供你选择。

当你GitHub上创建了一个新项目，你可以选择许可协议。包括可以使你的GitHub项目开源的协议。

如果你还有其它的疑问或者与开源项目相关的法律问题，请来这里。

编写README

READMEs不仅解释了如何使用你的项目，他们还解释了你的项目为什么重要，以及用户可以用它做什么。

在你的README中尽量要回答以下的问题：

- 这个项目是做什么的？
- 为什么这个项目有用？
- 我该如何开始?
- 如果我需要使用它，我能从哪里获得更多帮助。

你可以用README去回答其它的问题，像你如何处理贡献，项目的目标是什么，开源协议的相关信息。如果你的项目不想接受贡献，或者你的项目不能用于产品，你就可以将这些写在README中。

一份好的文档意味着会吸引更多的用户，收到更少的支持请求，得到更多的贡献。（···）请记住你的读者们不是你。参与同一个项目的开发者们有着完全不同的经历。

— @limedaring, “Writing So Your Words Are Read (video)”

有时候，人们不会去编写README。因为他们觉得项目还没有完成或者他们不想要贡献。这些都是非常好的为什么要编写README的理由。

为了获得更多的灵感，可以尝试使用 @18F’s “编写可阅读的READMEs” 或者 @PurpleBooth’s README 模板去编写一份README。

当你的根目录中包含README文件后，README就会显示在GitHub仓库的首页上。

编写你的贡献指南

一份CONTRIBUTING文件能否告诉你的粉丝如何参与你的项目。例如，文件中可能会包含如下信息：

- 如何报告bug（尽量使用 issue 和 pull request 目标）
- 如何提议一个新特性
- 如何建立你的开发环境和运行测试

另外技术清单和一份CONTRIBUTING文件是一个你向贡献者传达你的期望的机会。如:

- 你渴望得到什么类型的贡献
- 项目的发展路线或者期望
- 贡献者应该如何联系你

使用温暖，友好的语气，并提供具体的建议（如写作文档或做一个网站）可以很大程度上让新来者感到欢迎和兴奋参与。

例如，Active Admin starts its contributing guide with:

> 首先，感谢你考虑为*Active Admin*做贡献。就是因为有了像您这样的人让*Active Admin*成为了一个伟大的工具。

在项目的早期，你的CONTRIBUTING文件会比较简单。为了做出贡献，你应该总是解释如何报告bugs或者文件issues和一些技术要求（像测试）。

过了一段时间，你肯会把频繁出现的提问添加到CONTRIBUTING文件中。写下这些信息意味着会有更少的人再重复向你提相同的问题。

想获得更多书写CONTRIBUTING文件的帮助，请查阅 @nayafia's 贡献指南模板 or @mozilla's "如何创建 CONTRIBUTING.md".

在README中附上CONTRIBUTING文件的链接，这样会让跟多的人看到。如果你 将CONTRIBUTING文件放在项目的仓库中,GitHub会自动链接你的文件当贡献者创建一条issue或者打开一个pull request。

制定行为规则

我们有过这样的经历，我们面临什么是滥用，或者作为一名维护者试图解释为什么有些事必须按一定的方式，或者作为一名用户提出简单的问题。(…)
一份行为规则会变成一份简单的参考和可链接的表示你的团队提出的建设性的话语非常认真的文档。

— @mlynch, “Making Open Source a Happier Place”

最后，一份行为规则帮助你为你项目的参与者建立了行为准则。如果你为一个社区或者一家公司发起一个开源项目，它是非常有价值的。一份行为规则授权你促成健康，有建设性的社区行为，这回减轻你作为一名维护者的压力。

想获得更多信息，请查阅我们的 行为规则指南.

除了沟通如何期望参与者行为之外，行为准则还倾向于描述这些期望适用于谁，何时应用，以及如果违规发生时该做什么。

许多开源协议一般也会为行为规则制定标准，所以你可以不用再编写。这份贡献者盟约 是一份被超过40,000个开源项目所使用的行为规则，包括 Kubernetes, Rails和Swift。无论你使用哪个文本，在必要的时候你都应该执行你的行为规则。

将文本直接粘贴到你仓库中的CODE_OF_CONDUCT文件中。将文件放在项目的根目录中方便查找，同时在README中添加相应的链接。

命名和品牌化你的项目

品牌不仅是一个华丽的logo或者易记的项目名。它还关于你如何谈论你的项目，以及你想把信息传递给谁。

选择正确的名字

选择一个容易记住，有创意，能表达项目用意的名字。例如：

- Sentry 监控应用程序的崩溃报告
- Thin 是一个简单快速的Ruby web服务器。

如果你的项目是基于一个已存在的项目创建，那么使用他们的名字作为你项目名的前缀会帮助你阐述你项目的用途。 （例如 node-fetch将 `window.fetch` 添加到了 Node.js)。

考虑阐明所有。押韵虽然有趣，但是记住玩笑不可能转变成其它的文化，或者他人与你有不同的经历。你的一些潜在用户可能是公司员工，你不能让他们在工作中很难解释你的项目！

避免命名冲突

查看是否有同名的开源项目，尤其是你分享的是同样的语言或者生态系统。如果你的名字与一个已存在的知名的项目有冲突，你会让你的粉丝感到困惑。

如果你想要一个网站，Twitter账号或者其他特性来展示你的项目，先确保你能得到你想要的名字。理想情况下，为了美好的未来现在保留这些名字，即使你现在不想用他们。

确保你的项目名没有侵权。如果有侵权，可能会有公司要求你的项目下架，或者对你采取法律措施。这样得不偿失。

你可以查阅WIPO全球品牌数据库避免商标冲突。如果你是在公司工作，法律团队会帮你做这件事。

最后，去谷歌搜索你的项目名。大家会很容易地找到你的项目吗？在搜索结果礼是否有你不想让大家看到的东西？

你的写作（和代码）如何影响你的品牌

在项目的整个生命周期中，你需要做很多文字工作：READMEs，教程，社区文档，回复issues，甚至肯能要处理很多来信和邮件。

是否是官方文档或者一封普通的邮件，你的书写风格都是你项目品牌的一部分。考虑你可能会拥有粉丝，以及这是你想传达的声音。

我尝试处理每一个细节，包括：处理邮件，展示示例，友好待人，认真处理大家的issues以及试图帮助到大家。经过一段时间后，大家可能不再是只问问题，还会帮助我解决其他人的疑问以及给我喜悦，他们模仿我的风格。

— @janl on CouchDB, “Sustainable Open Source”

使用热情，通俗易懂的语言（如“他们”，即使是指一个人）能够让新来的贡献者感觉项目非常欢迎他们。使用简单的语言，因为你的读者可能英语不是很好。

除了书写风格外，你的编码风格也是你项目品牌的一部分。 Angular 和 jQuery是两个项目代码风格严谨的示例和指南。

当你的项目才开始时，没有必要为项目编写一份风格指南。你可能会发现你喜欢将不同的编码风格融入到项目。但是你应该想到你的书写和编码风格会吸引或者拒绝不同类型的人。项目的早期是你建立你希望看见的先例的机会。

你的预发布清单

准备好开源你的项目了吗？有一份帮助检查清单。检查所有内容？你准备开始吧！ 点击 “publish” 以及拍下自己的后背。

文档

- 需要为项目指定一个开源协议
- 项目要有基础文档 (README, CONTRIBUTING, CODE_OF_CONDUCT)
- 易记的项目名，指出项目是做什么的，不能和已存在的项目冲突或者商标侵权
- 最新的issue队列，组织和标记清除的issues

代码

- 项目使用一致的代码风格和明确的功能/方法/可用的名字
- 注释清晰的代码，记录意图和边缘案例
- 在修改历史，issues或者 pull requests 中没有敏感的信息 (例如 密码或者其他不能公开的信息)

人

如果你是代表个人:

- 你已经告诉了你的法律部门，以及/或者理解了你公司 (如果你是某一家公司的员工) 的开源政策和IP

如果你有一家公司或者组织:

- 你已经告诉了你的法律部门
- 你有一个宣布和促进项目的营销计划
- 一些人被允许管理社区互动 (回复issues，检查和合并pull requests)
- 至少有两人管理访问项目

你做到了！

恭喜你开源了你的首个项目。不论结果如何，对开源社区都是一份礼物。随着每次commit,comment和pull request，你正在为自己或者他人创造学习和成长的机会。

找到你的用户

推广你的项目

当你建立一个项目之后，没有说你一定要去推广他，即使是这个项目并不是很流行，仍然可以找到很多理由让你在这个项目上花时间。但是如果你希望别人能发现并且使用你的项目，那么这个时候您就需要把你辛苦工作的成果告知世人！

找出你的卖点

在你开始推广你的项目之前，你应该能够解释你的项目是做什么的，为什么大家需要他?

是什么让你的项目变得不同或者有趣，在自己心中问这些问题会让你更容易说服别人。

牢记一件事情，别人之所以使用你的项目，甚至是为你的项目做贡献，是因为你的项目解决了他们的问题。所以你要找出他们需要什么，然后把他当成你项目的卖点或者说价值所在。

举个例子，`@robb`用代码实例来清晰的阐述为什么他的项目`Cartography`是有用的。

如果你想深入了解如何挖掘项目的“卖点”，看一下Mozilla的`“Personas and Pathways”`，练习如何建立用户的形象。

帮助人们发现然后关注你的项目。

你最好有一个唯一的“主页”链接用来推广，引导人们关注你的项目。你不需要找一个炫酷的模板或者域名，但是你的项目确实需要一个入口。

— Peter Cooper & Robert Nyman, `“How to Spread the Word About Your Code”`

通过引导他们到一个唯一的地址来帮助人们发现和记住你的项目。

要有一个推广的主阵地。一个Twitter账号，Github链接，或者IRC频道是引导人们查看你们项目的一个简单的方式。这些方式也给你日益增长的社区一个讨论的好地方。

如果你目前还不想给你的项目搞这么多乱七八糟的东西，而且还要在有机会的时候推广你的Twitter账户和Github账户。举个例子，如果你某一个讨论会或者活动上发言要保证在你的简历或者幻灯片上包含这些信息。只有这样人们才会知道怎么找到你或者关注你的工作。

我之前犯过的一个错误就是没有给项目开一个Twitter账户。Twitter是一个让人们知晓项目进展的好渠道，也可以让人们持续的接触到你的项目。

— @nathanmarz, “History of Apache Storm and Lessons Learned”

考虑给你的项目做一个网站一个网站可以让你的项目更加友好，而且更加容易浏览，更重要的是附上清晰的文档和教程。这也是象征着你的项目还是活跃的，这会让你的用户使用你项目的时候感觉更放心。可以用一些例子告诉人们如何使用的项目。

@adrianholovaty, Django的协作者说，我们给Django做的网站可以说是“在早期开发Django的时候做的最好的一件事情了”。

如果你的项目是托管在GitHub上的，你可以用GitHub Pages简单的创建一个网站。Yeoman,

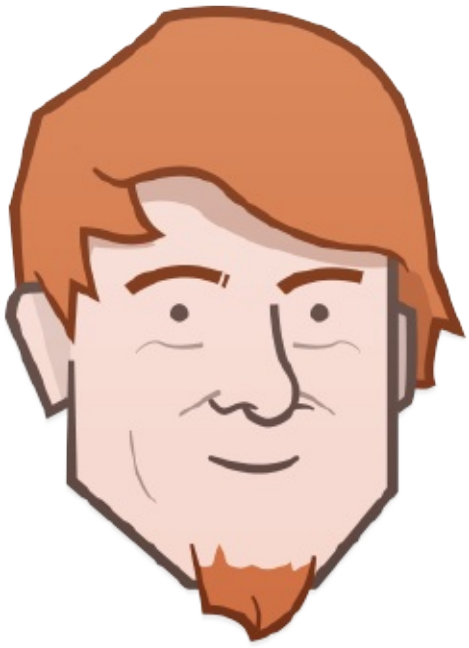
Vagrant, and Middleman 是一些优秀的内容详尽的网站的例子

现在你的项目有了“卖点”，和让人们很容易发现你项目的渠道，接下来我们谈谈如何和你的用户交流吧！

到你项目的受众在的地方去（线上）

网上拓展是分享和快速宣传项目的一个好方法。借助一些网上的渠道，你有可能找到一大批受众。

利用好已有的线上社区和平台去找你的受众。如果你的开源项目是一个软件项目，你可能会在Stack Overflow, reddit, Hacker News, 或者Quora。找到你觉得人们会最有可能从你的项目中受益或者对你项目感兴趣的渠道。



每个程序都会有那么一些方法只有一部分人才会用到，所以不要想着去打扰每一个人，把你的力气用在可能会从你项目受益的社区就好。

— @pazdera, “Marketing for open source projects”

来看看下面的一些方法吧，也许推广你的项目的时候用得着。

- 快找找有没有相关的开源项目和社区。有时候，你不需要直接的推广你的项目。如果你的项目对使用Python的数据科学家来说是无可挑剔的，那么就去找Python数据科学的社区。等他们知道你的项目之后，很自然的就会谈论然后分享你的工作成果。
- 如果你项目尝试解决某些问题，那么找到会遇到这些问题的人。想象你的项目受众会在哪些论坛，然后搜索这些论坛，回答他们的问题，然后找一个合适的实际，向他们建议使用你的项目来作为一种解决方案。
- 寻求反馈。给一个可能会用到你项目的人介绍你自己和你做的工作。对哪些人会从你的项目受益要很明确。尝试完善一下下面这句话：“我觉得我的项目能够帮助A，那些尝试做B的人”。听取和回复别人的反馈，而不是简单的推广。

一般来说，在你索取什么回报之前先把精力放在帮助别人上。因为在网上推广一个项目对任何人都是一个不难的事情，所以有很多人和坐着一样的事。告诉人们你是谁，而不是你想要什么，这样才能从众多推广者中脱颖而出。

如果没有人对你的推广感兴趣，不要灰心！大部分的项目的开展都是一个要花费数月和数年的反复过程。如果你第一次没收到反应，尝试换一种策略，或者找办法给别人的项目做做贡献。这都是些需要时间和奉献精神的事情。

到你项目受众在的地方去（线下）

线下活动是一个推广项目流行的方式。这是一个接触某个忠实听众和建立深层次的联系的好方式，特别是如果你对到场的开发者感兴趣的话。

如果你还是个公中演讲的新手，从寻找一个和你项目使用的语言或者生态系统相关的当地的聚会开始吧。

我去Pycon的时候非常紧张。我要发表一个演讲，在那儿我就认识几个人，还在那儿呆了整个周。但是其实我不应该焦虑的。Pycon真是太他妈吊了！每个人都是超级友好外向，以至于我没有找到时间和人们讲话。

— @jhamrick, “How I learned to Stop Worrying and Love PyCon”

如果你从来没在公共场合讲过话，感觉紧张那就太正常啦！记住你的听众就在哪儿，因为他们都是真正的想听你介绍你的项目。

当你在写你的演讲稿的时候，把重点放在你的听众会感兴趣而且能获取价值的事情上。保证你的语言要友好和和蔼可亲。笑一笑，深呼吸，幽默一点儿。

当你开始写你的演讲稿的时候，不管你的主题是什么，如果你能把你的演讲当成是给别人讲故事的话，效果会更更好。

— Lena Reinhard, “How to Prepare and Write a Tech Conference Talk”

等你准备好了，考虑一下在某个会议上发言的时候推广你的项目研讨会可以帮助你接触更多人，有时

候是来自全世界各地的人。

我非常认真的给JSConf的人写了一封信，然后求他们给我一点时间让我JSConf上展示我的项目。同时我又非常担心，这个项目我做了六个月，要是大家不认可怎么办。那时候我就一直在想，我的天，我他妈在这里是干吗？

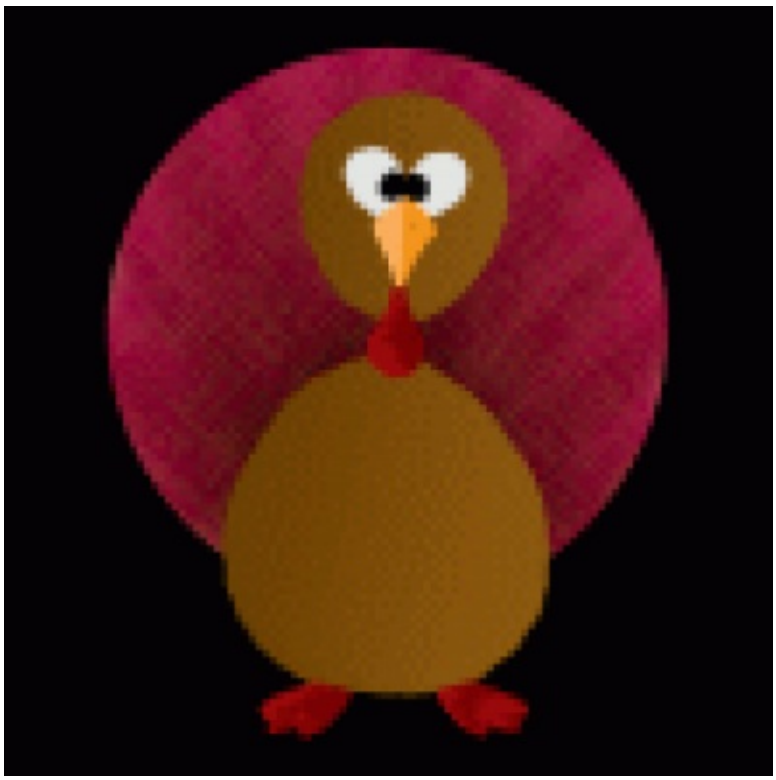
— @ry, “History of Node.js” (video)

建立声望

除了上面提到的策略之外，邀请人们分享和支持你的项目的最好办法就是分享和支持他们的项目。

帮助新手，分享资源，给别人的项目认真的做贡献会帮助你建立起良好的声誉。然后他们就很有可能知道你的项目而且更有可能关注和分享你在做的事情。

有时候，这些关系还会进一步发展成更广阔的生态系统中的官方合作伙伴（意思即使你有可能成为那些有名社区的成员）



urllib3之所以是现在最流行的Python第三方库的唯一原因就是大家都需要他。(待修改)

— @shazow, “How to make your open source project thrive”

种一棵树最好的时候是十年前，其次是现在。所以啥时候开始建立你的声望都不晚。即使是你早就已经建立了自己的项目，还是要继续找办法帮助别人。

建立用户群没有一蹴而就的方法。获取别人的新人和尊重需要时间，同样，建立声望的过程也永远不会停止。

PhantomJS公开第一个版本的时候实在2011年初。我也就是用一些常规的方法来推广：发Twitter，写博客告诉别人可以用它来做什么，在各种各样的聚会上我都提到过它。当2014年他已经广为人知的时候。我才开始做关于它的演讲。

— @ariya, “Maintainer Stories”

一如既往的坚持！

有时候，让人么注意你的开源项目会话费很多事件。但是关系！现在很多流行的项目都是花了很多年才有今天的活跃度。把重点放在建立声望上而不是企图一夜成名。耐心一点，一如既往的和那些可能会从中受益的人们分享你的项目。

建立成功的项目

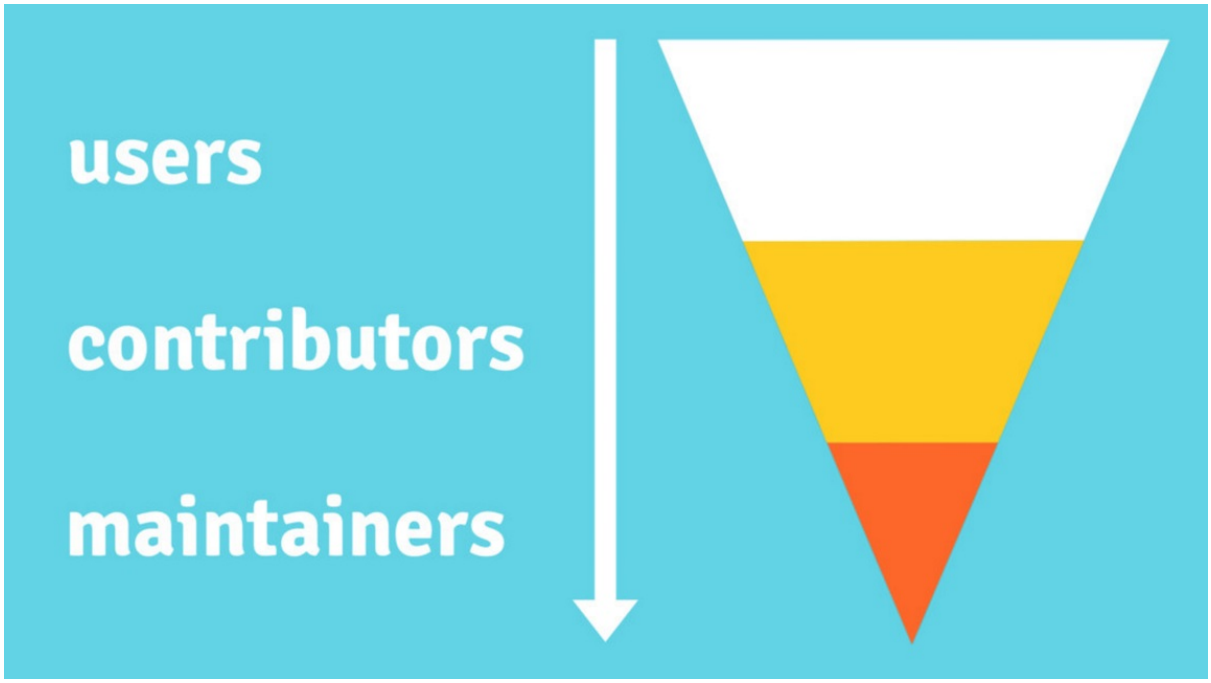
建立成功的项目

你们已经启动了你们的项目，你们正在传播它，同时有人正在查看它。真棒！现在，你如何让他们坚持下去。

一个受欢迎的社区是对你们项目的未来和声誉的投资。如果你的项目才开始收到第一次贡献，那么你们需要尽早的给贡献者们一次积极的经历，以至于能让他们继续参与贡献。

让大家感到受欢迎

可以通过被@MikeMcQuaid称之为贡献者漏斗的方法思考你们项目的社区。



当你们建立了自己的社区，你需要考虑如何让那些处在漏斗上方的人（潜在用户）转移到漏斗下方（活跃的维护者）。你们的目标是减少贡献者们在每个阶段遇到的摩擦。当人们能够轻易的取得胜利时，他们会乐意去做更多事。

从你的文档开始：

- 让大家很容易使地用你的项目。 一份友好的 README以及清晰的代码示例将让大家很简单地开始你们的项目。
- 清楚的解释如何做贡献，使用你的CONTRIBUTING file以及持续更新issues。

好的文档能够邀请他人参与你们项目的互动。最终，一些人会开一个issue或者pull request。将这些互动视为机会，将他们转移到漏斗的下方。

- 当一些人选择了你们的项目，请对他们表示感谢！仅仅只是一次消极的经历就能让一些人不想再回来。
- 及时回应。如果你们一个月都没有回答他们的问题，他们可能早已忘记了你们的项目。
- 对你以后接受的所有贡献者持开放态度。 很多贡献者是从一份bug报告或者小的修复开始的。这里有很多为项目做贡献的方式。让大家选择他们喜欢的方式。
- 如果你不赞成一个贡献， 首先你需要对他们的想法表示感谢，同时 解释为什么它不适合项目，如果有必要的话你可以给出相关的文档链接。

为开源做贡献对一些人来说很简单，但对另外一些人可能就不是这样了。有很多人因为没有做正确的事而害怕叫喊，或者只是不适合。（。。。）通过允许贡献者参与一些对技术要求比底的工作（文档，web content markdown，etc），可以极大的减少你对这些情况的关注。

— @mikeal, “Growing a contributor base in modern open source”

多数开源贡献者是“临时贡献者”，因为他们只是偶尔参与项目贡献。一位临时贡献者可能没有充足的时间全程跟踪你的项目，所以你们的工作是能让他们很容易地参与贡献。

鼓励其他的贡献者也是对你们项目的一种投资。当你们授权大量的粉丝做他们感兴趣的工作时，你们的压力就少了很多。

记录一切

你是否参加过一个（技术）活动，你不认识在场的人，但是似乎每个人站在一个小组里像老朋友一样聊天？（。。。）现在想象下你想为一个开源项目做贡献，但是你不知道为什么或者这个是如何发生的。

— @janl, “Sustainable Open Source”

当你开始一个新项目，你会觉得保持工作的私有性是正常的。但是开源项目发开始于你在公共平台记录自己的工作进程。

当你门把事情记录下来，会有更多的人能够按照你们的方式参与每一步。你可能会得到意想不到的帮助。

书写东西不仅仅只是技术文档。任何时刻，你们有写一些东西或者私自讨论项目的冲动，请询问自己是否能将之公开。

保持项目透明的项目路线：你们期待什么类型的贡献者，如何审查贡献，或者你们为什么做某些决定。

如果你们注意到有多个用户遇到过同样的问题，那么你们应该将答案记录在README中。

对于经常遇到的问题，你们可以考虑发布你们的笔记或者相关的issue。在这种情况下得到的反馈将会令你们惊讶。

记录一切也适用于你们的工作。如果你正在进行大量的更新工作，请将其放入pull request并标记

为正在进行（WIP）。这样，可以让其他人感觉参与过早期工作。

积极回应

一旦你推广项目，人们将会给你们反馈。他们可能会问项目是如何工作的，或者需要你们帮助开始项目。

当有人列出一条issue，提交一个pull request，或者询问项目的有关问题时，你们应该尽量回答他们。当你们快速地做出回应时，人们将感觉到他们参与了对话，以及他们将会更热情地参与。

如果你无法及时审查请求，请尽早确认，这样会有助于提高参与度。这里是@tdreyno在Middleman上如何回应一个pull request:

一份Mozilla研究发现 如果贡献者在48小时内收到代码审查，他们会有很大的可能返回以及重复贡献。

与你们项目有关的话题也可能发生在互联网的其它地方，例如Stack Overflow，Twitter，或者Reddit。你门可以在像这样的一些网站设置通知，这样当有人提及你们项目时可以收到提醒。

为你们的社区提供一个聚会的场所

有两个理由可以解释为什么要给社区提供一个聚会的场所。

第一个理由是为了他们。帮助人们相互认识。有着共同兴趣的人会想要一个可以聊天的地方。同时当信息是公开的而且是适宜的时候，任何人可以阅读过去的档案以至于能够快速的追赶以及参与。

第二个理由是为了你们。如果你们没有提供一个公共的场所来谈论你们的项目，他们可能会直接与你们联系。刚开始时，回复私有来信可能对你们来说很轻松。但是经过一段时间后，尤其是如果你们的项目变得流行的时候，你们就会感到累了。不要私下和人们谈论你们的项目，而是直接指明他们去指定的公共渠道。

公共交流和指明人们开一条issue一样简单，而不是直接发给你们发邮件或者在你们的博客发表评论。你们也可以为了方便人们谈论你们的项目设置一个邮件列表，或者创建一个Twitter账号，Slack，护着IRC渠道。或者尝试上述的所有方式。

Kubernetes kops sets aside office hours every other week to help community members:
Kubernetes kops 每隔一周抽出办公时间帮助社区成员:

> *Kops*每隔一周都会留出时间为社区提供帮助和指导。*Kops*维护者已经同意留出时间专门与新手一起工作，帮助*PRs*，以及讨论新特性。

公开交流需要特别注意的异常有：1）安全的issues和2）敏感的行为准则。你们应该为大家提供一

个私下报告这些issue的方式。如果你们不想使用自己的个人邮箱，那么就创建一个准用邮箱。

让社区成才

社区非常有能量。这种能量可能是祝福也可能是诅咒，这取决于你们如何执掌它。随着你们项目社区的成长，有办法帮助它成为一股有建设性的力量，而不是具有破坏性的。

不要容忍糟糕的角色

一些流行的项目将不可避免地回吸引来一些伤害它们的人。他们可能从一些没必要的争论开始，对一些细小的功能进行谬论，或者伤害他人。

对于那种类型的人你们必须采取零容忍的政策。如果发现较晚，那些消极的人将会社区中的其他人不舒服。他们可能会离开。

事实是，拥有一个支持性社区是关键。如果没有来自我的同事，互联网上一些友好的陌生人，以及聊天渠道IRC的帮助，我不可能做好这些工作。（。。。）不要太少。不要找麻烦。

— @karissa, “How to Run a FOSS Project”

定期对你们项目琐碎方便的辩论，使他人，包括你们不能把注意力集中于重要的任务上。新人如果看见这样的情景，他们是不会加入你们项目的。

当你们发现社区中有消极行为时，需要公然指出来。特别说明的是，要用坚定的语气解释他们的行为为什么是不可接受的。如果这种问题继续发生，你们有必要要求他们离开。你们的行为准则是为这些情景准备的建设性指南。

知道贡献者在哪里

随着你们项目的成长，好的文档只会变得越加重要。临时贡献者不可能对项目非常熟悉，通过阅读你们的文档他们能很快找到他们需要的。

在你们的CONTRIBUTING文件里，需要明确告诉新来的贡献者该如何开始。你们可能为了想要达到这个目的而准备一个专门的部分。

在你们的issue列表中，bugs标签需要适合不同类型的贡献者：例如，@kentcdodds/first-timers-only-78281ea47455#.g1k01jy05">*“first timers only”*, *“good first bug”*, 或者 *“documentation”*. 这些标签能够帮助新人快速浏览issues以及开始。

最后，使用你们的文档让人们在每一步都感到欢迎。

你们永远不会与登陆项目的大多数人互动。你们可能没有收到一些贡献，因为有些人感到害怕或者不知从和开始。即使是几个字也能阻止一些人沮丧地离开你们的项目。

例如，这里是Rubinius如何开始它的贡献指南：

> 我们想感谢你们使用*Rubinius*。这个项目是一个充满爱的劳动，我们希望所有用户查找*bugs*，取得性能上的提升，以及帮助完善文档。每一个贡献都是有意义的，所以感谢你们的参与。话虽如此，但我们还是要求你们遵守一些指南，这样我们就能够找到你们的*issue*。

分享你们项目的所有权

你们的领导者们将有不同的观点，因为这是所有健康社区都该做的！然而，你们会体会到粗暴鲁莽的做法不能得到大家的认同，反而谦虚低调的做法更容易让大家接受。

— @sarahsharp, “What makes a good community?”

当大家觉得拥有项目的所有权时，他们会乐意为项目做贡献。这并不意味着你们需要转变项目的愿景或接受你们不想要的贡献。但是你们越信任他们，他们就会越忠诚。

看你们是否能尽快地找出向社区分享所有权的方法。下面有些建议：

- 反对你们自己修复简单（非关键）的**bugs**。 相反，使用它们作为招募新贡献者的机会，或者指导想要参与贡献的人。开始时可能效果不是很理想，但经过一段时间你们会得到想要的结果。例如，@michaeljoseph要求一位贡献者提交一个pull request在一个Cookiecutter issue的下面，而不是自己修复它。

- 在项目中添加一个贡献者或者作者文件用于记录每一个参与贡献的人。
- 如果你们的社区有了一定的规模，那么发送一封信或者发表一篇博客感谢贡献者们。Rust的This Week in Rust和Hoodie的Shoutouts是两个非常好的示例。
- 给每个贡献者**commit**的通道。@felixge发现这样会使大家越发乐意斟酌他们的补丁，以及他甚至发现了他在一段时间没有工作的项目的新维护者。

- 如果你们的项目在GitHub上，那么将项目从你们的个人账号转移到一个组织，以及添加至少一个备份管理员。组织能让与其他人一起工作于同一个项目变得更加容易。

事实上很多项目只有一个或者两个做大量工作的维护者。随着你们的项目以及社区越来越大会更加容易得到帮助。

当你们不能总是发现一些人去回答问题时，你们可以释放一个信号增加其他人能接触到的机会。如果你们能尽早地开始，大家就能尽快地帮助。

你们最大的兴趣是招募喜欢你们项目以及能够做你们不能做的事的贡献者。你喜欢编码，但不喜欢回答issues？那么让社区中能做这件事的人去做。

— @gr2m, “Welcoming Communities”

解决冲突

在你们项目的早期，做主要的决定是件容易的事。你们想做什么就可以做什么。

随着你们的项目越加流行，会有更多的人对你们的决定有兴趣。即使你们的社区没有大量的贡献者，如果你们的项目有很多用户，你们会发现大家的重点在决策上或者增加他们的issues。

在大多数情况下，如果你们培养了一个友好，尊重的社区并公开记录你的过程，你们的社区应该能够

找到解决方案。但有时候你们遇到一个issue，有点难以解决。

建立友好的氛围

当你们的社区正在讨论一个很难的issue时，气氛会很激烈。人们可能会变得愤怒或者沮丧，以及发泄在其他人或者你们身上。

作为一名维护者你们的工作是不要让这种情况出现。即使你们对话题有很强烈的观点，也要尽量站在一个主持者或者推动者的位置，而不是参与争吵以及推动自己的观点。如果有人不友好或者垄断话题，那么立即采取行动以保持有礼貌以及丰富的讨论。

作为一名维护者，尊重你们的贡献者非常重要。他们经常处理一些你们描述亲切的事情。

— @kennethreitz, “Be Cordial or Be on Your Way”

一些人希望你们指导他们。编写一个好的示例。你们仍然可以表达失望，不高兴或者忧虑，但得心平气和。

保持你们的酷并不容易，但是展示领导能力能促进健康的社区。互联网感谢你们。

将你们的README视为宪法

你们的README不仅仅是一组指令。它也是一个谈论你们目标，产品愿景和路线的地方。
如果人们过分专注于辩论特定功能的优点，它可能有助于重新审视您的README，并谈论你们的项目的更高的愿景。关注你们的README也会使对话变得个人化，所以你们可以进行建设性的讨论。

专注过程，而不是结果

一些项目用投票的方式做重要决定。虽然看上去是明智的，投票强调的是得到一个“答案”，而不是倾听以及解决每个人的顾虑。

投票会变成政治，社区成员在做感兴趣的事或者表决一个明确的方法时会感到压力。不是每个人都参与了投票，可能在你们的社区中保持沉默的人占了多数，或者用户不知道投票这件事正在发生。

有时候，投票是必要的手段。尽你们所能强调“寻求共识”而不是共识。

在寻求共识的过程中，社区成员讨论主要问题，直到他们感到他们已经得到充分的意见。当仅遗留下一些无关紧要的问题时，社区需要向前迈进。“寻求共识”不能确保社区能得到一个完美的答案。而是侧重聆听和讨论。

Atom Issues不存在投票系统的部分原因是因为Atom团队在所有情况下都不会遵循投票系统。有时我们必须选择我们认为是对的事，即使它不流行。（。。。）我能通过社区的反馈知道我能够提供什么以及做什么样的工作。

— @lee-dohm on Atom's decisionmaking process

即使你们不确定是否采用寻求共识的方式，作为维护者，让大家知道你们正在关注他们。让其他人知道，以及承诺解决他们的问题，这在很大程度上减少了敏感情况的发送。然后，按你说的去做。

不要为了获得决议而急于做出决定。在做一个决议之前请确保每个人已经知道以及所有的信息以及公开。

将对话的重点聚焦于行动

讨论很重要，但是富有成效和没有效果的对话时有区别的。

鼓励讨论，只要它正积极地朝着解决问题的方向进行着。如果对话已经无法再进行下去，只有很少的人在参与或者大家正在讨论无关紧要的问题，这时候就该结束对话了。

允许这些对话进行下去不仅对解决问题没有帮助，而且不利于社区的健康发展。它释放了这样一个信号，表示允许或甚至鼓励这种类型的对话，它可能阻止人们提高或者解决未来的问题。

当你们或者其他人每提出一个观点时，请自问："这如何使我们更接近一个决议？"

如果对话开始有解散的征兆，问团队："我们下一步该做什么?"才能重新对话。

如果一个对话没有清晰的方向，没有明确的措施可以采取，或者合适的措施已经被使用，那么关掉issue并解释为什么关掉它。

指导一件事朝着正确的方向发展是一门艺术。它对阻止人们浪费时间或者要求他们发表有建设性的看法没有作用。（。。。）反而，你们必须为接下来的进展给出条件：给大家一个路线，跟随一个可以得到你们想要的结果的途径，这样就不像是些无用的口头行为。

— @kfogel, *Producing OSS*

挑战你们的智慧

上下文很重要。考虑谁参与讨论，以及他们如何代表社区的其他人。

社区中的每个人都为这个问题而烦恼，或者参与讨论了吗？或者只是一部分人感到困惑吗？不要仅关心活跃的声音，也请不要忘记考虑社区中保持沉默的人。

如果这个问题不代表社区的更广泛的需求，你们可能要承认只是少数人的担心。如果这是一个反复出现的issue，没有一个清晰的解决方案，那么指向他们以前讨论的话题。

找出社区中的决策者

通过一个态度端正和目标清晰的对话，很多困难都是可以解决的。即使在富有成效的对话中，对于如何进行的意见也可能存在差异。在这些情况下，确定一个人或一组人，可以作为决策者。

决策者可以是项目的主要维护者，或者是大家投票选出的一个小团体。理想情况下，在你们使用

GOVERNANCE文件之前，你们已经确定了决胜者和与之相关的事宜。

使用决策者应该是你们最后才能采取的手段。分离issues是一个你们社区成长和学习的机会。利用这些机会以及协同合作，尽量找出解决方案。

社区是开源的❤

健康，蓬勃的社区每周都会为开源付出大量辛勤的劳动。许多贡献者指出其他人在开源工作或不在开源工作的原因。通过学习如何建设性地利用这个权力，你们会帮助他人有一个难忘的开源体验。

项目维护者的最佳实践

作为一个项目的维护者意味着什么

如果你维护者一个很多人都在用的项目，你可能就会意识到你写代码的时间变少了而回答issue的时间变多了。

在项目开始之初，你会不断尝试着实现自己的新想法，在知道自己 想要什么的基础上决定项目的走向。一旦你的项目开始流行之后，你会发现你的大部分时间都花在和用户以及贡献者打交道上。

维护一个项目需要的不仅仅是写代码的能力。有些时候会有一个你意想不到的的事情要应付，但是这对一个项目的成长也很重要（相对于代码来说），我们收集了一些小技巧来让你的生活变得更轻松，从写文档到管理社区。

记录项目的进展

对于一个项目的维护者来说写文档是最重要的事情之一。

文档不仅说清楚了你的想法是什么，而且还帮助别人在问问题之前理解你需要什么和接下在希望做什么。

即使你不想长篇大论，对要点略说一二也比啥都不写要好。

写下你的项目的发展方向。

从写下你的项目目标开始。把他们加到README， 或者创建一个单独的VISION文件。如果还有什么其他的老古董能帮助人们了解这方面的信息，比如项目管理路线图，也把他们公开吧。

有一个明确的，用文档写清楚的方向能保证你不跑偏，也不会因为其他的贡献者增加一个奇怪的需求。

比如，@lord 发现有项目有一个明确的发展方向会帮助他决定哪个PR值得花时间。作为一个维护者的新手，他甚至还后悔当他接到第一个关于slate)PR的时候没有坚持它项目本身的原则。

我一直都在摸索。我没有努力寻求一个完整的解决方案。与其采用那种半吊子办法，我真希望曾经对某些Issue的提出者说：我暂时没有时间干这个，但是我会把他放到我的待办事项中。

— @lord, “Tips for new open source maintainers”

和大家交流你自己对项目的期望

制定规则是很伤脑筋的。有时候你可能觉得你像是在限制别人的行为或者说把好玩的东西都搞没了。

制定了规则，然后严格执行，当然啦，好的规则会让维护者更轻松。他们会把你从做自己不愿意做的事情中解脱出来。

大多数知道你项目的人对你或者你的处境都是一无所知。他们可能会觉得你做这个项目是有钱拿的，特别是你的项目是他们经常用的而且某种程度上有点依赖的时候。其实你只是在有时候会在项目上花很多时间，但是现在你在忙着安置新工作或者照顾刚出生的儿子。

这些都是再正常不过的事情！所以雀斑别人也知道这点。

如果你维护某个项目是临时的或者完全自愿的，能花多少时间就花多少时间。而不是你觉得项目需要你花多少时间或者别人想让你花多少时间。

这里是一些值得你写进项目里的东西：

- 怎样的贡献才会被复查和接受(需要测试吗？提*Issue*有模板吗？)
- 你希望有什么类型的贡献？（你是不是只希望在某些部分的代码上需要别人的帮助？）
- 在合适的时候跟进项目（比如说 如果你在七天之内没有收到*maintainer*的回复，那就先干别的去吧）
- 你会在这个项目上话多少时间（比如说 “我们每星期只会在这个项目上花*5*个小时“）

`Jekyll`, `CocoaPods`, and `Homebrew` 这是一些给维护者和贡献者制定了基本规则的项目例子.

保证交流公开

不管是什么时候，保证你的交流实在公共的场所（就是大家都看的地方）。如果有人尝试私聊你讨论一个新的需求或者功能，礼貌的引导它到公共的交流场所，比如邮件列表或者`iuuse tracker`。

如果你和别的维护者面基了，或者在私下做了一个很重要的决定，把这些信息告诉大家，即使只是把你的笔记发上去。

这样的话，每个人新加入到你们社区的人和已经呆了几年的人能够了解到的信息是一样的。

学会说不

文档你也写好了。理论上讲，每个人应该都会阅读你的文档，但是实际上，你总是要提醒有些小伙子他问的东西已经有答案了。

把所有的事情都写下来，当然，对你执行你制定的规则的时候客观分析实际情况也有帮助。

拒绝别人没啥意思，但是“你的贡献不符合这个项目的标准”比“我不喜欢你的贡献”要好听。

作为一个维护者，在很多情况下，你都要拒绝别人：不符合项目规则的PR， 某个人脱离了讨论的重点，给别人做无关紧要的工作。

保持友好沟通

你要学会拒绝的最重要的一个地方就是Issue和PR请求。作为一个项目的维护者， 你会不可避免的收到你不想接受的建议。

可能某个贡献改变了项目范围或者和你的期望不合。可能想法很好，实现的很烂。

不管是什么原因，在处理这些不符合项目标准的贡献的时候都要婉转。

如果你收到了你不想接受的贡献，你的第一反应可能是忽略或者假装没看到。但是这么做会严重伤害到别人而且可能会让你社区里的其他贡献者失去动力。

管理大型开源项目的关键就是保证issue活跃。尽量避免让issue停滞不前。如果你是一个IOS开发者，你会知道提交雷达是多么让人沮丧（我也不知道这是什么意思。。）你可能过了两年之后有人让你兼容一下现在的IOS版本。

— @KrauseFx, “Scaling open source communities”

别因为自己感到内疚或者想做一个好人就把你不想接受的贡献继续保留。随着时间的流逝，这些你没有回答的issue和PR会让你觉得很不爽。

更好的方式是马上关掉你不想接受的贡献。如果你的项目已经保守积压的issue的折磨，@steveklabnik可以给你点儿建议，how to triage issues efficiently.

第二点，忽略别人的贡献给社区传递了一个负面的信号。让人感觉做一个贡献是很吓人的事情，特别是对于新手来说。即使你不接受他们的贡献，告诉他们为什么然后致谢。这会让人觉得更舒服。

如果你不想接受某个贡献：

- 感谢他们 的贡献
- 解释为什么他们的贡献不符合项目的需求范围，然后提供清楚的建议以供改善，如果你可以的话。和蔼一点，但同时也要讲原则。
- 引用相关的文档， 如果你有的话。如果你发现你反复收到你不想接受的贡献，把他们加到文档以免你重复叙述。

你不需要用超过1-2两句话来回复。比如，当一个celery的用户报告了一个window相关的错误，@berkerpeksag 是这么回复的

如果你感觉拒绝别人很不好意思，也很正常，其实很多人都是这样。就像 @jessfraz 说到的：

> 我和很多来自诸如*Mesos, Kubernetes, Chromium*等不同开源项目的维护者交流过，他们都异口同声的觉得做一个维护者最难的就是拒绝你不想打补丁。

对于不想接受别人的贡献这件事不要感到愧疚。如 @shykes所说开源的第一原则就是 “拒绝是暂时的，接受是永远的”当然啦，认同别人的热情还是一个好事，拒绝他的贡献和拒绝他这个人是两码事。

最后，如果一个贡献不是够好，你没任何义务接受它。对那些想对你的项目做贡献的人保持和蔼和积极的态度，但是只能接受那些你确定会让你的项目变得更好的提交。你说拒绝的次数越多，对你来说拒绝别人就越容易。谨记！

保持主动

要想减少你不想接受的贡献的数量，首先，在你项目的贡献指南中解释如何提交贡献以及怎样的贡献会被接受。

如果你收到太多低质量的贡献，让那个贡献者先多做做功课，比如：

- 填写一个issue或者PR的模板/清单
- 在提交PR之前先开一个issue

如果他们不遵从你的规则，马上关掉issue病并引用你的文档。

当然啦这么搞一开始是不太舒服，但是你主动一点确实对双方都好。它减少了某个人花了太多事件到一个你不想要的PR上的可能性。而且让你管理起来更容易。

理论上，在CONTRIBUTING.md文件里面告诉别人在他们开始干活之前如何更清楚的知道的干完之后会不会被接受。

— @mikemcquaid, “Kindly Closing Pull Requests”

有时候，当你说不的时候，你潜在的贡献者会感到对你的沮丧或者不爽。如果他们开始找你撕逼了，take steps to diffuse the situation或者干脆把他们从你的社区开除，如果他们不打算和你保持建设性的合作关系的话。

接受指导

可能在你的社区里有人不断提交一些不符合项目需求的贡献。对你们双方来说不停的拒绝他的提交都很尴尬。

如果你发现有人对你的项目很上心，但是就是需要调教，那就耐心一点。给他解释明白每次它的提交为什么不符合项目需求。尝试着让他先做一些简单一点的事，比如那些标有*“good first bug”*标签的issue，以此让他慢慢习惯。如果你有时间的话，考虑教他怎么完成第一次贡献，或者在社区找一个人教他。

利用社区的力量

你不需要凡事亲力亲为。这就是社区存在的原因！即使你没有一个活跃的贡献者社区，但是如果你有很多用户的话，拉他们来干活儿。

分担工作量

如果你正在寻找其他人来参与， 从身边的人开始。

当你看到新的贡献者不停的提交贡献，通过分配给他们更多任务来表示认可。如果别人愿意的话，记录下别人是怎么成长为领导者的过程。

鼓励别人来一起管理项目能很大程度上减轻你的工作量，就像 @lmccart在他的项目上做的那样，p5.js

我曾经说过，“对，每个人都可以参与进来，你不需要有很多编程的经验。”当有申请来参加我们的活动的时候，我就在想，这是真的吗，我说了啥？有将近40个人来了，我虽然不可能和每个人都单独交谈，但是大家一起来了，这说明我说的没错。只要有人知道怎么做了，他们就能教他们的邻居。

— @lmccart, “What Does “Open Source” Even Mean? p5.js Edition”

如果你需要暂时或者永久的离开的项目，大可找人来代替你。

如果别人认同项目的发展方向，给他们提交的权限或者正式把项目所有权转移给他。如果有人fork了你的项目而且在保持活跃的维护，考虑在你的原始的仓库放上这个fork版本的链接。如果大家都希望你的项目继续的话这不失为一种好办法。

@progruim 发现 由于它给他的项目Dokku写一个关于项目发展方向的文档，即使在它离开这个项目后他的那些目标仍然会被实现。

> 我写了一个*wiki*来描述我想要啥和为什么。不知道为啥，项目的维护者就开始推动项目朝那个方向发展，这对我来说还是有点惊讶的。他们会一点不差的按照我的意愿去做这个项目吗？不总是这样，但是总是会把项目推动到离我的理想状态更近的位置。

让别人尝试他们自己想要的解决方案

如果有贡献者关于项目有不同的意见，你可以礼貌的鼓励它在他自己fork版本上继续工作。

fork一个项目不什么坏事情。能复制并且修改别人的代码是开源项目最大的好处之一。鼓励你的社区成员在他自己fork的仓库上继续工作是在不和你的项目冲突的基础上，给实现他们的想法最好的出口。

我迎合80%的用户需求。但是如果你是那20%中的一个，那么fork我的项目吧。我不会介意的！我的公开的项目是用来解决那些最常见的问题的。我尝试着让别人fork我的项目然后在上面拓展得更加简单。

— @geerlingguy, “Why I Close PRs”

这对于那些强烈的需要某个你没时间实现的解决方案的用户来说也是一样的。提供API或者自定义的钩子帮助他们更好的实现自己的需求而不需要改动源码。@orta发现鼓励在CocoaPods上使用插件导致了很多有趣的想法的诞生。

> 一旦一个项目变大之后，维护者对怎么增加新代码变得保守是不可避免的事情。你可能很会拒绝别人的需求，但是很多人提的都是合法的需求。所以，你不得不把你的一个工具变成平台。

使用机器人

就像很多工作别人可以帮你做一样，也有很多工作不需要人来做。机器人是你的朋友，用他们让你的维护者生活变得更容易。

引进测试和别的检查来改善你的代码质量

让你项目自动化的最重要的方法之一就是引进测试。

测试能够帮助贡献者自信他们没有弄坏什么。测试也让你复查代码和接受别人的贡献的过程更加容易。你反应的越多，社区参与的就越多。

设置自动化的测试让所有新来的贡献者都可用，而且保证你的测试可以很容易在贡献者的本地运行。保证所有的代码贡献者在提交之前都运行你的测试。你还得为所有的提交设置一个基本的标准。

如果你添加了测试，确保在CONTRIBUTING文件里面解释他们是怎么工作的。

我相信测试对所有的代码都是需要的。如果代码被完整的覆盖了测试，以后就不需要改了。我们只需要在代码崩溃或者需要某个功能的添加代码。不管你在修改什么，测试对于检查那些你可能不小心制造的问题都是必须的。

— @edunham, “Rust’s Community Automation”

用工具来自动化日常的维护工作

对于维护一个流行的项目来说，一个好消息是别的维护者也可能遇到过类似的问题而且已经找到一个解决方案。

这里有各种各样的工具帮你自动化一部分的维护工作。一些例子i:

- semantic-release 自动化版本发布
- mention-bot @可能的贡献者来帮你复查代码
- Danger 帮你自动复查代码

对于bug报告和其他常见形式的贡献，Github有Issue Templates and Pull Request Templates, 你可以用来降低沟通成本。你也可以设置邮件过滤来管你你的邮件提醒。

如果你想逼格更高，代码风格指南和linter能让你项目收到的贡献更规范，而且更同意复查和接受。

当然啦，如果你的标准太复杂了，反倒会增加了贡献的难度。所以保证你只添加一个让每个人工作起来更容易的规则。

如果你不确定用什么工具，看一看别的流行项目是怎么做的，特别是和你在一个生态系统的。比如，其他的Node模块的贡献流程是怎么样的？用想死的工具和方法会让你项目的贡献流程对贡献者更熟悉。

不干了也没关系

开源项目曾经让你开心，但可能现在开始让你不开心了。

可能当你想到你的项目的时候感觉到亚历山大。而同时，issue和PR又纷至沓来。

疲倦在开源工作工作中是一个常见的问题，特别是在维护者中间。作为一个维护者，你做的开心对项目的生存来说是一个没有商量余地的条件。

虽然你不需要跟谁请假，但是也不腰拖到自己疲倦不堪的时候才去度假。@brettcannon，一个python的核心开发者，决定在14年的义务劳动之后休一个月的假

就像其他工作一样，有规律的休息会让你对工作保持舒适愉快的心情。

我是WP-CLI的维护者，我发现我需要首先让自己开心，在开源项目和其他事情之间设定清楚的界限。

我发现最好的平衡点就是每周在日常的工作安排中花2-5小时在这上面。这会让我从感觉太累到保持持续的激情。因为我给我需要解决的issue表上了优先级，从而我能够在我认为重要的事情上有所进展。

— @danielbachhuber, “My condolences, you’re now the maintainer of a popular open source project”

有时候，当你感觉大家都离不开你的时候请假去休息是一件很难的事情。甚至人们会让你因为离开而感到愧疚。

在你离开项目的时候尽可能的在用户和社区中间寻求支持，如果你找到支持你的人，还是休息吧。在你不工作的时候还是要保持和别人交流，这样人们不会因为你的失联感到奇怪。

休息并不只是独家。如果你周末不想做开源项目的工作了，或者在本该工作的时候不想干活了，和别人说，这样他们才知道这个时候不该打扰你。

首先照顾好自己！

维护一个大型项目需要和在项目开始成长时不同的技能。作为一个维护者，你会将自己的领导力和个人能力提示一个层次，而这是很少人能体会到的。但是与此同时，管理项目，设定清晰的界限，制作你感到舒服的事情会让你保持开心，活力，高产的状态。

学会管理成长中的项目

学会管理成长中的项目

你们的项目正在成长，也有人参与进来，你们承诺保持这样的状态。在这个阶段，你们可能想知道如何将常规项目贡献者纳入你们的工作流程，无论是否给他们提交权限或者解决社区辩论。如果你们有疑问，我们给出了答案。

在开源项目中使用正式角色的示例有哪些？

许多项目遵循类似的贡献者角色和认可结构。

这些角色实际上意味着什么，完全取决于你。这里有一些你可能认识的角色类型：

- **`Maintainer`**（维护者）
- **`Contributor`**（贡献者）
- **`Committer`**（提交者）

对于一些项目来说，**”maintainers”**是在项目中有提交权限的人。在其他项目中，他们是作为维护者被名字被列在README中的人。

维护者不一定要是为项目编码的人。他可能是为你们项目的宣传做了很多工作的人，或者是为项目编写文档以方便其他人使用的人。无论他们每天做了什么，维护者是对项目的方向有责任感以及致力于改善项目的人。

一名”contributor”*可以是任何评论过issue或者`pull request`的人，给项目带来价值（无论是对issues进行分类，编码，或者组织活动）的人，或者任何合并过`pull request`的人（可能是贡献者最狭隘的定义）。

[对于Node.js来说，]项目社区中的每个人都参与评论issue或者提交代码。这意味着他们从用户过度到了贡献者。

— @mikeal, “Healthy Open Source”

术语”提交者”可能是用于区分提交权限，它是一个有着明确责任的角色类型，来自其它形式的贡献。

虽然你们可以以任何你们想要的方式定义项目角色，但
请考虑使用更广泛的定义来鼓励更多来自其他形式的贡献。你们可以利用管理者的身份认识为项目做出特别贡献的人，不论他们的技能是什么。

你们可能认为我是Django的“发明者”…但事实是当她诞生一年后我才参与这份工作。（…）人们怀疑我成功了，因为我的编程技能…但我最多只是个普通的程序员。

— @jacobian, “PyCon 2015 Keynote” (video)

我如何正式化这些领导者角色？

正式化你们的领导者角色能够帮助人们有归属感并告诉社区中的其他成员他们可以向谁寻求帮助。

对于小项目来说，指派领导者可以和将他们的名字添加到你们的README或者CONTRIBUTORS的文本文件中一样简单。

对与比较大的项目来说，如果你们有网站，可以创建一个团队网页或者将你们项目的领导者们排列在上面。例如PostgreSQL有一个详细的团队页面并有每个贡献者的简单信息。

如果你们的项目有个非常活跃的社区，你们可能形成一个维护者的“核心团队”，甚至拥有来自不同issue领域（例如：安全，issue分类或者社区准则）的人组成的小组委员会。让大家自己组织并志愿选择他们喜欢的角色，而不是分配给他们。

[我们]补充了核心团队和几个“子团队”。每个子团队专注于一个特定领域（如：语言设计或者库）。（。。。）为了整个项目的协作和强大，每个团队的愿景要与整个项目愿景保持一致，每个子团队由一名来自核心团队的成员领导。

— “Rust Governance RFC”

领导者团队可能想要创建一个指定的通道（像IRC）或者定期会面来讨论项目（像Gitter或者Google Hangout）。你们甚至可以组织这样的公开会议以便其他人参与。例如，Cucumber-ruby的每周办公时间。

一旦你们建立了领导者角色，请记得以文档的形式记录告诉大家如何联系他们！给大家如何成为你们项目的一名维护者或者仅仅是加入一个子委员会建立一个清晰的流程，并将之写进你们的GOVERNANCE.md中。

有些工具像Vossibility可以帮助你公开跟踪谁给项目了贡献（或者没有）。记录这些信息可以避免社区认为维护者是一个团体，可以私下作出决定。

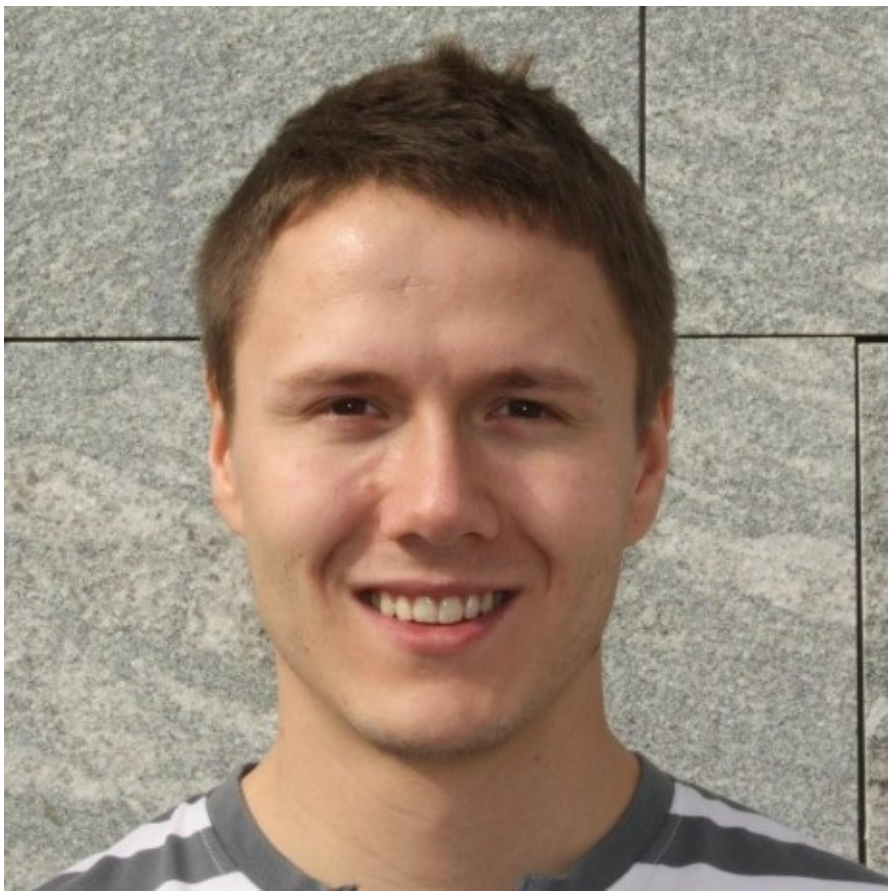
最后，如果你们的项目在GitHub上，请考虑将你们的项目从个人账号转移到一个组织并添加至少一个备份管理员。GitHub组织

我什么时候应该给一些人提交权限？

一些人认为你们应该给所有参与贡献的人提交权限。这么做能够让更多的人对项目有归属感。

一方面，特别是对于比较大，很复杂的项目，你们可能只想把提交权给那些已经表示了忠心的人。这种事没什么对错，你们开心就好！

如果你们的项目在GitHub上，你们可以利用受保护的branches管理谁可以在什么情况下像某个特定的branch进行push。



无论什么时候有人向你们发送一个pull request，请给他们你们项目的提交权限。虽然这听上去非常愚蠢，但是使用这样的策略能够让你们充分利用GitHub的优势。（。。。）一旦大家有了提交权限，他们就 不用再担心他们的补丁没有合并了…造成他们浪费了大量的时间。

— @felixge, “The Pull Request Hack”

开源项目的一些常见治理结构有哪些？

这里有三个与开源项目有关的常见治理结构。

- **BDFL**:SDFL表示”Benevolent Dictator for Life（仁慈的独裁者的生活）”。在这个模式下，只有一个人（通常是项目的作者）对项目的主要决议有最终得决定权。Python就是一个经典的示例。小项目一般默认是BDFL，因为他们只有一两个维护者。如果是公司的项目可能也会使用这中策略。

- **Meritocracy（精英）：**(注意：这个术语”**meritocracy**（精英）”对一些社区来说带有消极的意味，同时用有一个复杂的社会以及政治历史。)在精英模式下，活跃的贡献者（展示”价值”的人）被赋予了一个正式的决策者角色。决议通常是通过纯投票的形式抉择。这个权威的概念是由Apache Foundation首创；所有Apache的项目都是精英。贡献仅由他们个人提供，而不是

公司。

- **Liberal contribution**（自由贡献）:在自由贡献模式下，做最多工作的人被认为是最有影响力的，但是是以现在的当前工作为基准而不是过去的贡献。项目的主要决议都是通过寻求共识的方式得到的，而不是以纯投票的形式，并努力考虑社区中更多的观点。使用自由贡献模式非常流行的示例包括：Node.js和Rust。

你们应该选择哪种？这取决于你们！每种模式都有有点以及需要权衡之处。虽然咋一看他们有着很大的不同，但是他们的共同点要比看上去的多。如果你们有兴趣采用其中的一种模式，可以浏览这些模板：

- BDFL model template
- Meritocracy model template
- Node.js's liberal contribution policy

当我的项目启动时我需要编写管理文档吗？

虽然没有合适的时间写下你们项目的管理文档，
但一旦你们看到你门的社区动态表现，它就容易定义了。开源管理的最好（最难的）部分是它是由社区塑造的！

一些早期文档不可避免的是用于你们项目的管理。所以，开始写下你们可以写的。例如，在你们项目启动的时候你们可以清晰地说明期待什么样的行为，或者你们的贡献者如何处理工作。

如果你们是参与公司开源项目启动的成员，在项目发布之前，你们有必要进行内部讨论，了解你们的公司如何保持并决定项目的进展。你们也可以公开解释贵公司将如何（或不会）参与该项目的任何事情。

我们分配小团队来管理GitHub上的项目，他们实际上在Facebook工作。例如，React是由一位React工程师管理。

— @caabernathy, “An inside look at open source at Facebook”

如果公司的员工开始提交贡献会发生什么？

成功大开源项目会被很多人和公司使用，以及甚至有些公司的收入会与这些项目有关。例如，公司可能使用开源项目中大代码作为他们商业服务的一部分。

随着项目被广泛地使用，会需要更多具有专业知识的人，你们可能就是他们中的一个！同时，有时大家在为项目工作时会得到报酬。

重要的是平常心对待商业活动，并且将之视作其他资源发展的动力。当然，不应该区别对待有报酬的开发者和其他无薪酬的；每个贡献都应该根据其技术特点进行评估。然而，大家应该开心地参加商业轰动；同时当争论对项目有利时，大方地陈述他们的用例。

“商业”和“开源”是兼容的。“商业”仅仅是意味着有金钱的参与，软件被用于商业，这有利于项目的发展和推广。（虽然非开源产权中使用了开源软件，但整个产品依然是“专利”软件。开源可以用于商业

或者非商业目的。）

和任何人一样，具有商业动机的开发者也是通过他们贡献的质量和数量提高影响力的。很明显，得到报酬的开发者可能会比没有报酬的做的更多，但这是被允许的；金钱只是影响一些人做少事情的很多因素中的一个。让你们的项目讨论侧重于贡献，而不是关注使人们能够做出贡献的外部因素。

我需要一个法律顾问来支持我的项目吗？

你们不需要法律顾问来支持你们的开源项目，除非涉及到金钱。

例如，如果你们想创建商业业务，你们将要建立C Crop或者LLC（如果你们位于美国）。如果你们只是做与你们的开源项目相关的合同工作，你可以作为独资经营者接受金钱，或设立一个LLC（如果你在美国）。

如果你们的开源项目想要接受捐赠，你们可以设置一个捐赠按钮（例如使用PayPal或者Stripe,还有中国的支付宝和微信支付），除非您是符合条件的非营利机构（如果您在美国），否则这笔钱不会免税。

很多项目为了省去建立非盈利机构的麻烦而去找一家非营利机构赞助他们。非营利机构代替你们接受捐赠，但你们需要给他一定比例的捐款。自由软件保护，Apache基金会，Eclipse基金会，Linux基金会和Open Collective都是为开源项目提供赞助的组织。

我们的目的是提供可被商业持续使用的基础设施，因此创造一个每个人（包括贡献者，支持者，赞助者）都能受益的环境。

— @piamancini, “Moving beyond the charity framework”

如果你们的项目与某种特定的语言或者生态系统联系紧密，那么你们可以和与之相关的基金会合作。例如，Python软件基金会帮助支持用于管理Python包的PyPI，Node.js基金会帮助支持一个Node基础框架项目Express.js。

我为什么需要行为守则

我为什么需要行为守则？

行为守则是一份确立项目参与者行为规范的文件。采用和执行行为守则可以帮助你们的社区营造积极的氛围。

行为守则不仅帮助保护你们的参与者，同时还有你们自己。如果你们维护一个项目，随着时间的推移，可能会发现其他参与者懒散的态度会让你们疲惫或对工作不满意。

一份行为守则可以帮助你们促进健康，有建设性的社区行为。积极主动减少你们或其他人在你们的项目中变得疲劳的可能性，并帮助你们在有人做出你们不同意的事情时采取行动。

建立行为守则

尽可能早地建立行为守则，当你们第一次创建项目的时候。

此外，说出你们的要求。行为守则的描述遵循如下几点：

- 行为守则在哪里有效 *(只在issues以及pull requests，或者社区活动？)*
- 行为守则适用于谁 *(社区成员以及维护者，那赞助商呢？)*
- 如果有人违反了行为守则会怎样？
- 大家如何举报违规

无论你们在哪里，请使用已有的行为守则。贡献者盟约是一个被超过40，000个开源项目（包括Kubernetes, Rails和Swift）所使用的行为守则。

Django行为守则和Citizen行为守则都是非常好的行为守则。

请将CODE_OF_CONDUCT文件放在你们项目的根目录，并在README中附上其链接，这样对你们的社区是可见的。

决定你们如何执行行为守则

一份行为守则没有（或者不能）执行会比没有行为守则更糟糕。它释放这样一个信息：行为守则或者尊重在你们的社区并不重要。

– Ada Initiative

你们应该解释如何执行行为守则在违规发生之前。有几点理由说明为什么这么做：

- 必要的时候，它表示你们处事认真谨慎。
- 你们的社区会因为投诉确实可以得到回复而更加放心。
- 如果他们发现自己因为违规而被调查时，你们能确保社区的审查流程是公平透明的。

你们可以给大家一个私有的渠道（如email地址）以便大家报告违规行为以及解释谁收到了这一的报告。它可以是维护者，一组维护者或行为守则工作组。

请不要忘记了有人可能想要报告某些人违规接受了这些报告。在这样的情况下，也给他们举报那些人的机会。例如，@ctb和@mr-c 在他们的项目上解释， khmer：

> 对于滥用现象，扰乱或者其他不可接受的行为都可以向***khmer-project@idyll.org***（仅由*C. Titus Brown*和*Michael R. Crusoe*处理）发送邮件。要报告涉及其中任何一个的问题，请电邮***Judi Brown Clarke, Ph.D.*** *BEACON*行动进化研究中心的多元化主任，*NSF*科学技术中心。

为了获得灵感，可以查阅Django的执行手册（你们是否需要如此详细的手册，这取决于你们的项目）。

执行你们的行为守则

有时，尽管你们尽了最大的努力，仍然会有人违反守则。当这样的情况发生时，有几种方法来解决消极或有害的行为。

搜集有关违规的信息

认真对待社区中每个成员的想法。如果你们收到有人违规的报告，请认真对待并调查此事，即使它不符合你们自己的经验。这样做可以向你们的社区表面，你们珍视他们的观点和信任他们的判断。

有的社区成员可能是让大家一直不舒服的惯犯，或者他们只是说了或做了一次。这都需要依据实际情况进行处理。

在你们做出回应之前，请认真思考发生了什么事。通过阅读他们过去的评论和对话可以更好地理解他

们为什么要那样做。尽量收集其他人对他们行为的看法。

不要陷入争论。在你们处理完手头上的事情之前，不要侧重于处理别人的行为。专注于你们需要什么。

— Stephanie Zvan, “So You’ve Got Yourself a Policy. Now What?”

采取适当的行动

当搜集和处理足够的信息后，你们需要决定做什么。当你们在考虑下一步的时候，请牢记你们的目的是营造一个安全，尊重和协作的社区氛围。不仅要考虑如何处理有问题的情况，还要考虑们的反应将如何影响你们社区的其他行为和期望。

当有人报告违规时，处理它是你们的工作，而不是他们的。有时，报告者透露他们的信息会给他们的职业生涯，声誉和人生安全带来很大的风险。迫使报告者面对骚扰者会将他们置于妥协的位置。除非报告者有特别的要求，你们应该直接和有问题的人沟通。

这里有些方法帮助你们回应违规行为：

- 向相关人员发出公开警告以及解释他们的行为产生了怎样的负面影响，最好在发生问题的地方。在可能的情况下，公开沟通会向社区的其他人传达你们认真对待行为守则。要友善，但坚定的沟通。
- 私下接触相关人员向他们解释他们的行为对其他人产生了怎样的负面影响。如果相关情况涉及到个人敏感信息，你们可能会使用私有通信方式。如果你们和一些人私下沟通，对于首先报告这个情况的CC来说是个好主意，因为他们知道你们采取了行动。在征求他们的意见之前，请向报告人征求同意。

有时，一个解决方案不能达到目的。有关的人可能在面对或者不改变他们的行为时变得气势汹汹或敌对。在这种情况下，你会想到考虑采用强制措施。例如：

- 暂停有关人员在项目中的工作，通过暂时禁止参与项目的任何方面执行
- 永久禁止有关人员加入项目

对于禁止成员的做法，你们应该非常谨慎，只有在没有其他解决方案的情况下才能使用。

维护者的责任和义务

行为守则不是可以任意执行的法律。你们是行为守则的执行者，同时你们的责任是遵守行为守则确立的规矩。

作为维护者，你们可以为社区指定准则，同时你们可以根据行为守则执行这些准则。这意味着你们需要认真处理违规行为。报告者对他们的投诉进行了彻底和认真地审查。如果你们确定他们报告的行为没有违规，你们需要他们进行沟通并解释你们为什么不进行处理。他们会怎样做，取决于他们：容忍他们认为有问题的行为，或者停止参与社区。

如果报告的行为没有技术上的违规，这可能表面你们的社区依然存在问题，同时你们应该调查潜在的问题以及采取相应的行动。这可能包括修改你们的行为守则，以澄清可接受的行为和/或与行为被举报的人交谈，并告诉他们，虽然他们没有违反行为守则，但是他们在期望和确定的边缘另其他参与者感到不舒服。

最后，作为维护者，你们给可接受的行为建立和执行标准。你们有能力塑造项目社区的价值观，以及参与者希望你们能 公平公正地执行这些价值观。

鼓励你们希望看见的行为 ？

当你们的社区变得似乎敌对或者不受欢迎时，即使是一个大家能容忍的个人行为，也会让你们失去很多贡献者，你们可能再也遇不到其中的一些人。虽然执行或者采用行为守则很难，但是营造一个受欢迎的环境将帮助你们社区成长。

了解开源的法律含义

了解开源的法律含义

向世界分享你们具有创造性的工作，这是一个多么令人激动和有价值的经历。这也意味着你们必须担心一堆你们不清楚的法律问题。幸运的是，你们不必从头开始。我们已经涵盖了你们的法律需求。（在你们行动前，请确定阅读了我们的免责声明。）

为什么大家非常担心有关开源的法律问题？

很高兴你们提问！当你们行创造性工作（例如写作，图形或代码）时，默认情况下该作品属于专有版权（copyright）。也就是说，法律承认你们是你们作品的作者，他人在没有经得你们同意的情况下不能使用你们的工作。

一般来说，这意味着没有人可以在没有你们授权的情况下使用，复制，分发或者修改你们的工作。

然而，开源有着不一样的情况。因为作者希望他人使用，修改以及分享他们的工作。但因为法律默认依然是专有版权（copyright），所以你们需要一个明确说明这些权限的协议。

如果你们不应用开源许可协议，那么对你们项目做出贡献的人也都将成为其工作的专属版权（copyright）所有者。这意味着没有人（也包括你们）可以使用，复制，分发后者修改他们的贡献，

最后，你们的项目可能具有你们不知道的许可证要求的依赖关系。你们的项目社区，或者你们的雇主政策也可能要求你们使用特定的开源许可协议。

公开的GitHub项目是开源的吗？

当你们在GitHub上创建一个新项目 时，你们可以选择将仓库设置成**private**或者**public**。

Making your GitHub project public is not the same as licensing your project. Public projects are covered by GitHub's Terms of Service, which allows others to view and fork your project, but your work otherwise comes with no permissions.
让你们的**GitHub**项目公开与许可你们的项目是不同的。公开项目是由GitHub的服务条款保护，它允许他人浏览以及fork你们的项目，但是没有权限参与你们的工作。

如果你们想让他人使用，复制，修改你们的项目，或者参与贡献你们的项目，那么你们的项目就需要包含一个开源许可协议。例如，即使你们的项目是公开的，但没有你们的授权，一些人是不能合法在他们的代码中使用你们GitHub项目中的任何部分。

请告诉我该如何保护项目

你们很幸运，开源许可协议已经标准化了同时使用简单。你们可以直接复制粘贴一个已经存在的许可协议到你们的项目里。

MIT，Apache 2.0和GPLv3都是非常流行的开源许可协议，但是也有其它选择。你们可以在choosealicense.com上找到这些许可协议的全部文本，同时说明了如何使用他们。

当你们在GitHub上创建了一个新项目，你们会被要求添加一个许可协议。

一个标准化的许可协议可以作为没有法律培训的人员的代理，以准确地知道他们可以和不能用软件做什么。除非绝对要求，否则应避免使用定制，修改或非标准术语，这将成为他人使用代码的障碍。

— @benbalter, “Everything a government attorney needs to know about open source software licensing”

哪个开源许可协议适合我的项目？

如果你们是小白，那么使用MIT License，不容易出错。它很短，很容易理解，并允许任何人做任何事情，只要他们保留许可证的副本，包括你们的版权声明。如果你们需要，您们能够根据不同的许可协议发布项目。

然而，为项目选择合适的开源许可协议这取决于你们。

你们的项目非常可能有（或将有）依赖。例如，如果你们开源了一个Node.js的项目，你们将可能使用来自npm（Node Package Manager）的库。你们依赖的这些库都有它们自己的开源许可协议。如果他们的许可协议“允许”（对使用，修改和分享给予公共权限，而对有关项目的许可协议没有要求），这样你们就可以使用任何你们想要的许可协议。共同允许许可协议包括MIT，Apache 2.0，ISC和BSD。

另一方面，如果你们的依赖中有一个的许可协议是“强硬的copyleft”（他们也给同样的允许，但条件是有关项目得使用同样的许可协议），那么你们的项目将使用与之相同的许可协议。copyleft许可协议包括GPLv2，GPLv3和AGPLv3。

你们也会想到考虑希望你们的社区使用以及贡献你们的项目：

- 你们是否想让你们的项目成为其它项目的依赖？在你们的相关社区最好尽可能使用最流行的许可协议。例如，MIT是npm libraries使用的最流行的许可协议。
- 你们的项目是否想吸引大企业？大型企业可能需要所有贡献者的明确专利许可。在这种情况下，Apache 2.0适合你们。
- 你们的项目是否想吸引不愿自己的贡献用于其它同类型软件的贡献者？GPLv3和AGPLv3适合你们。

你们的公司可能为自己的项目准备了特定的许可协议。例如，它可能需要许可许可证，以便公司可以在公司的闭源产品中使用你们的项目。或者你们的公司要求强大的copyleft许可协议同时要求贡献者赞成，这样项目只属于你们公司，没有人能在有关的软件中使用你们的项目。或者你们的公司可能有与标准，社会责任或透明度相关的某些需求，其中任何一个都可能需要特定的许可策略。与你们公司的法律部门谈谈。

当你们在GitHub上创建了一个新项目，它给你们提供了选择许可协议的机会。包括上面提到的可以使你们的GitHub项目开源的许可协议。如果你们想要了解其他选择，可以通过查

阅choosealicense.com找到适合你们项目（即使它不是软件）的许可协议。

如果我想更换项目的许可协议，该怎么办？

大多数项目绝不需要更换许可协议。但是情况偶尔有变。

例如，随着你们项目的壮大，它添加了新的依赖或用户，或者你们的公司改变了策略，或者其他的要求要更换不同的许可协议。如果你们在开始项目的时候没有添加许可协议，那么再添加一个许可协议和更换许可协议是一样的效果。当你们要添加或者更换项目的许可协议时，需要考虑以下三件事：

这件事很复杂。确定许可协议的兼容性和合规行，以及谁拥有版权，这会非常快速地变得复杂和混乱。为新的发布和贡献选择一个新的且合适的许可协议与重新许可已存在的贡献是不同的。一旦你们有任何想改变许可协议的想法，请首先让法律团队知道。即使你们已经或者能获得可以更换许可协议的权限，请考虑者会给项目的其他用户和贡献者带来怎样的影响。将更换许可协议视为“管理事件”，只有通过与项目的利益相关者进行明确的沟通和咨询，才更有可能顺利进行。请谨记，从一开始就为你们的选择和使用合适的许可协议。

你们的项目已经有了许可协议。如果项目的现有许可证与您要更改的许可证兼容，则可以开始使用新许可证。这是因为如果A许可协议与B许可协议兼容，你们将遵守A的条款，同时遵守B的条款（但不一定反之亦然）。因此，如果你们目前正在使用许可型的许可协议（例如MIT），则可以更改为具有更多条件的许可协议，只要你们保留MIT许可协议的副本和任何相关的版权声明（即继续遵守MIT许可协议的最低条件）。如果你们现在的许可协议不是许可型的（例如，copyleft或者你们还没有许可协议）以及你们不是版权的唯一所有者，那么你们不能将许可协议改为MIT。基本上，只要是使用的许可型的许可协议，版权所有者能事先更换许可协议。

你们的项目已经有版权所有者。如果你们是你们项目的唯一贡献者，然后你们或者你们的公司是项目版权的唯一所有者。你们可以添加或更换任何你们或者你们公司心仪的许可协议。不然你们需要取得其他版权所有者的同意。他们是谁？他们是已经参与你们项目提交的人。但有些情况是项目版权掌握在这些人的雇主手中。在某些情况下，人们只是做了微小的贡献，但没有硬性规定，在一些行代码下的贡献不受版权保护。对与这样的情况该怎么办？对于一个相对较小以及年轻的项目来说，取得所有贡献者对更换许可协议的同意是可行的。。但对于大项目和老项目来说，你们必须寻求很多贡献者以及他们继承者的共识。Mozilla花费了多年重新授权Firefox，Thunderbird和相关软件。

或者，你们可以让贡献者事先同意（通过额外的贡献者协议 - 见下文）在某些条件下更改某些许可协议，这些更改将超过现有的开源许可协议。这会改变许可协议改的复杂性。如果在执行许可协议更改时，你们仍然想要和项目利益相关者进行沟通，你们需要从你们律师那得到更多帮助。

我的项目需要额外的贡献者协议吗？

]可能不要。对于大多数的开源项目，一个开源许可协议可作用与所有贡献者和用户。

贡献者协议会给维护者带来额外的管理工作。这些协议增加了多少工作得取决去项目和实施。简单的

协议可能要求贡献者确认和点击，在项目的开源许可协议下他们有权利贡献。更复杂的协议可能需要法律的审查和贡献者的雇主的签字。

此外，贡献者协议有时被认为对项目社区不友好。他们也增加了人们参与社区的障碍。

我们已经删除了Node.js的CLA。这样做降低了Node.js贡献者的参与门槛，从而吸引更多的贡献者。

— @bcantrill, “Broadening Node.js Contributions”

一些情况下你们可能想要为你们的项目考虑一个额外的贡献协议：

- 你们的律师想要所有的贡献者明确接受贡献者条款，或者因为他们觉得只有开源许可协议还远远不够。如果这是唯一的问题，那么有肯定项目开源许可协议的贡献者协议就足够了。jQuery个人贡献者许可协议就是一个很好的轻量级的个人贡献者协议。对于一些项目来说，Developer Certificate of Origin是一个很好的先择。
- 你们的项目使用的开放源许可协议不包括明确的专利授权（如MIT），你们需要所有贡献者的专利授权，这些可能适合用于你们公司的专利组合或者项目的其他贡献者和用户。Apache 个人贡献者许可协议是一种常用的附加贡献者协议，其具有与Apache许可2.0中的专利许可相同的专利许可。

- 如果你们的项目使用的是copyleft许可协议，但你们也需要分发项目的专有版本。那你们需要每个贡献者分配版权给你们或者授予你们许可权。MongoDB贡献者协议就是这中类型的。
- 你们认为你们的项目在其有效期内需要更换许可协议，以及事先得到贡献者的同意。

如果您们实需要在您的项目中使用额外的贡献者协议，请考虑使用诸如CLA助手之类的集成，以最大限度地减少贡献者的分心。

我的公司的法律团队需要知道什么？

作为一名公司的雇员，如果你们在发布一个开源项目，你们首先需要让法律团队知道。
即使只是一个个人项目，请考虑让他们知道。你们可能和公司有一个“员工知识产权协议”，这给了公司一些对你们项目的控制权，特别是当项目和公司的业务有着很多的联系或者你们使用公司的资源发展项目。你们的公司应该很容易给们许可，也许已经通过一个员工友好的知识产权协议或公司政策。如果不是这样，你么可以谈判（例如，解释你们的项目为公司的专业学习和发展目标服务），或者你们在找到一个更好的公司前停止你们项目的工作。

如果你们的开源项目是为了你们的公司，者绝对需要让他们知道。根据公司的业务需求和专业知识，你们的法律团队可能已经制定了有关开放源代码许可协议（以及可能还有其他贡献者协议）的政策，以确保您的项目符合其依赖关系的许可协议。如果不是这样，你们和他们很幸运！你们的法律团队应该渴望与你们合作，把这个东西搞清楚。你们需要思考这些事：

- 第三方资源：你们的项目有其他人创建的依赖或者使用他人的代码？如果这些事开源项目，你们需要遵守第三方资源的开源许可协议。首先，选择与第三方资源的开放源许可协议一起使用的许可协议（见上文）。如果你们的项目修改或者发布第三方开源资源，那么你们法律团队还想知道你们符合第三方开源许可协议的其他条件，例如保留版权声明。如果你们使用了其他没有开源许可协议的代码，那么你们可能会要求第三方资源的维护者添加一个开源许可协议，要是你们得不到许可，你们只能停止使用他们的代码。

- 商业机密：请考虑项目中是否有公司不想对外公开的东西。如果是这样的话，你们只能开源项目的一部分，得保护好公司的商业机密。

- 专利：你们公司是否申请了与你们项目有关的专利？如果开源源代码，这会对公司的专利进行公开披露。可悲的是，你们可能被要求等待（或者公司会重新思考应用程序）。如果你们期望从拥有大量专利组合的公司的员工那里得到贡献，们的法律团队可能希望你们使用来自贡献者的明确专利授权的许可协议（例如Apache 2.0或GPLv3）或其他贡献者协议（见上文）。

- 商标：认真检查你们的项目名没有与已经存在的商标冲突。如果你们将自己公司的商标用于项目，请检查它有没有造成冲突。FOSSmarks是在自由和开源项目的背景下理解商标的实用指南。

- 隐私：你们的项目是否收集了用户数据并存储到公司的服务器？你们的法律团队可以帮助你们遵守公司政策和外部法规。

如果你们发布了公司的第一开源项目，为了能通过，以上这些绰绰有余（不要担心，大多数项目不会

引起重大关注）。
长期来说，们的法律团队可以做更多的事情，以帮助公司从开源中获得更多，并保持安全：

- 员工贡献策略：考虑制定一个公司策略，指明你们的员工如何为开源项目贡献。明确的政策将减少你们员工的迷惑，并帮助他们为公司的最佳利益向开源项目做贡献，无论是作为他们工作的一部分还是在自由时间。Rackspace的Model IP和开源贡献策略就是很好的示例。

放弃与补丁相关的只是产权以构建员工知识库和信誉。它表明，公司关心员工的发展，以及让员工有种被赋权和自主的感觉。所有这些好处还导致更高的士气和更好地保留员工。

— @vanl, “A Model IP and Open Source Contribution Policy”

- 发布什么：（几乎）所有？如果你们的法律团队了解并投资于你们公司的开源策略，他们将是你们最好的帮助，而不是阻碍你们的努力。
- 合规性：即使你们公司没有发布任何开源项目，他们也会使用别人的开源软件。意识和过程可以避免麻烦，产品延迟发布和诉讼。

组织必须具有适合[“允许”和“copyleft”]类别的许可协议和合规性策略。首先，记录适用于你们所使用的开源软件的许可条款（包括子组件和依赖项）。

— Heather Meeker, “Open Source Software: Compliance Basics And Best Practices”

- 专利：你们的公司可能希望加入开放发明网络，一个共享的专利防御池，以保护成员使用主要开源项目，或探索其他替代专利许可。
- 管理：特别是当如果将项目转移到公司以外的法律实体是有意义的。