

《开源软件设计与开发》课程总结

方孝君 51195100032

1. 开源理解

这个部分我想谈一谈在提交 PR 过程中，对参与开源项目贡献流程的理解，更具体地说，就是使用 Git 向开源项目提交 PR 的流程以及对 Git 这个工具新的理解。

① 幼年期：

最开始接触 Git 应该是在本科阶段，那时候我还只是将其当做一个简单的版本控制工具，只会 `git commit`，`git pull`，`git push` 等操作，说白了就是在玩单机。当然不可否认，Git 仅仅作为版本控制的工具，其体验也是非常优秀的。

② 成熟期：

本科毕业后我没有直接读研，而是在互联网公司工作了一年（当然后来被社会毒打了一顿之后又滚回校园来了）。在工作的过程中，我渐渐意识到，Git 的灵魂并不在于版本控制，而在于多人协作。

当很多人共同开发一个项目时，Git 的强大才被凸显出来。比如：Git 提供分支 `branch` 的概念，这样 `master` 主分支就可以作为一个被保护的线上部署分支，控制不让所有开发者都有权限向该分支 `push` 代码，而是选择自己新建一个私有的 `dev` 分支，然后在这个私有分支上自主开发，然后再向 `master` 分支提出 `merge` 请求，这样的做法显然更加安全，在多人协作时也更加有条理。再比如：当两个人都对同一个文件做修改时，有可能会发生冲突，而 Git 也提供了一套完整的解决冲突的方案。

除了协作方面，Git 也使得项目的容错机制更加强大，而这种容错机制很大程度上是依赖于其日志的，`git log` 记录了每个开发者每次提交代码的时间戳、`email` 账户、`commit message`、修改的文件等信息，这些日志有什么作用呢？——比如当在某个文件发现 `bug` 时，可以查看该文件的 `git history`，然后你就可以通过该文件的修改历史来进一步排查 `bug` 原因了；又比如在某次提交后导致了系统某个功能不可用，那可以依靠 `git log` 来进行版本回滚，保证线上版本的高可用性，然后再在线下修改 `bug`。

③ 完全体：

在本学期参与《开源软件设计与开发》这门课程，以及参与具体开源项目提交 PR 的过程中，我了解到参与 `github` 开源项目的规范流程。

以前开发一个工程时，常常是一起工作的同事对该代码仓库都有 `push` 权限，那大家就把代码 `git clone` 到本地，各自开发后向远程仓库 `push` 就行，与其他开发者有冲突就解决冲突。但 `github` 开源项目不一样，只有项目的 `committer` 才能

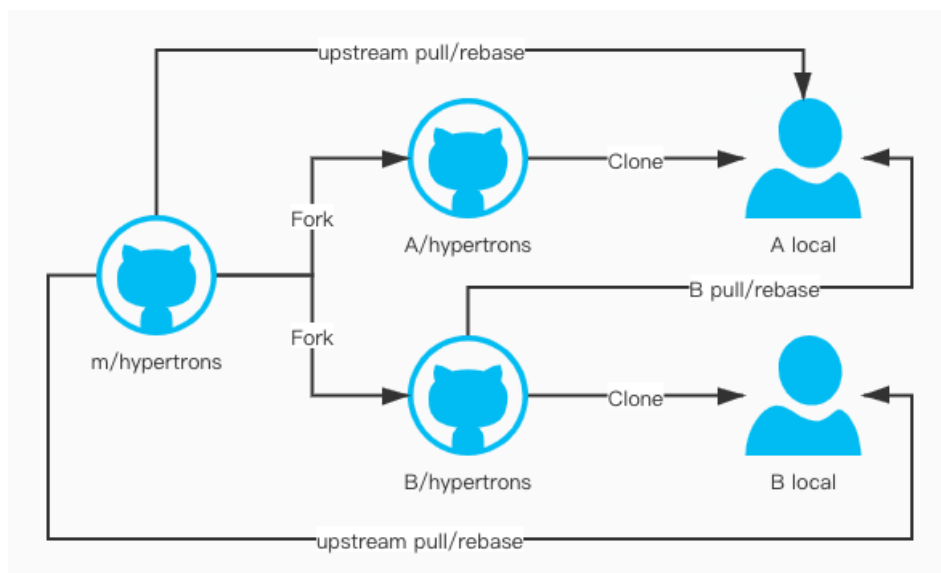
直接向项目 `push` 代码，而想要成为一个项目的 `committer` 并不容易，你不仅需要贡献大量的代码或者文档，还要在社区中比较活跃，经常参与讨论或回答别人的问题。成为 `committer` 之后，你不仅能向这个仓库 `push` 代码，还可以 `review` 别人的 `pull request`。

那么当你还不是 `committer` 时，如何向一个开源项目做贡献呢？——答案就是利用 `pull request`。具体的做法是：先将该仓库 `fork` 到自己的 `github` 账户下的 `repository` 中，相当于是在当前时刻拉取了一个该仓库的镜像到你本地仓库，这时候你就可以在自己的仓库下进行开发；当开发完成后，向自己 `fork` 的仓库 `push`；


然后，你需要 `git fetch`，`git rebase`，这是整个流程的关键所在，相当于拉取一个最新的上游仓库到本地，然后与本地 `fork` 的仓库做比较，有冲突就解决冲突；冲突解决后，你相当于领先了上游仓库一个提交，这时候你就可以去 `github` 上游仓库里提交 `pull request` 了。等待项目的 `committer` 给你 `review` 代码，他们觉得 `ok` 你的代码就会被合入，完成一次贡献，但通常情况是他们觉得你写的代码还有改进的地方，有可能是代码规范，也可能是有更优雅的写法，这时候你需要再修改一下代码重新提交 `PR`。

④ 究极体？

在 12 月 24 号的开源课程上，在同济大学的师兄给我们介绍了一种 `git` 更高级的用法：



当多人协作开发一个开源项目时，上游仓库、多个开发者 `fork` 的仓库、多个开发 `clone` 到本地的仓库，它们是完全分布式的，也即理论上它们是可以没有任何耦合的。而当两个人开发的模块有重合时怎么办呢？

 [WIP] refactor: modify the Lua component load mechanism ✓ kind/enhancement
#184 opened 3 days ago by WuShaoling

 [WIP] feature: lua hot fix ✗ kind/feature
#168 opened 7 days ago by liwen-tj • Changes requested


有时候开发者会给自己的 PR 前面打上一个[WIP](work in process)的标签，表示该 PR 还在开发过程中，不要着急 review 和 merge 代码，当 A 看到 B 也有一个与自己相同模块的 WIP PR 时，A 就能意识到他与 B 开发的内容有耦合。此时，A 可以将 B 的代码仓库 fork 过来，与自己的仓库做比较和解决冲突、合并，以减少不必要的重复工作。


当然 git 可能还有很多高级的用法，它确实包含了很多分布式协作的哲学思想，我还需要进一步学习，这还远不是究极体。


2. 开源贡献

我在课程开始的时候选择的项目是华为的 serviceComb，之前姜宁导师有在 slack 中与参与该项目的同学做一些讨论，他要求我们通过围观 issue 来学习相关技术和融入开源社区（issue 地址：<https://github.com/apache/servicecomb-pack/issues/571>），我在围观后第一个做出了反馈：

Also sent to the channel · Pinned by Willem Jiang

 **Willem Jiang** 2 months ago
大家可以在此回复围观心得。

 **方孝君** 2 months ago
技术层面的还没有去了解，就说一下别的方面的围观心得：1. 在提出issue的时候，对问题的描述方式是很重要的。这个issue的提出者描述了问题出现的情景、提供了本地测试的日志、讲了自己对问题原因的猜测、补充了自己所用工具的版本。这些都值得学习；2. 参与到开源项目的方式可能有很多。比如提出有价值的issue（可能并不需要自己去修复）、在issue下回复自己的想法（这个issue下就有人说自己碰到过类似的问题，以及是用什么方法解决的）、主动去提交PR来解决issue等等；3. 在相对成熟的开源项目里提交PR时，对代码质量的要求是很高的。在#572这个PR中，姜老师和coolbeevip都对xiangyuQi的代码提了很多意见和建议，直到coolbeevip提出要先判断HystrixPlugins.getInstance().getConcurrencyStrategy是否已经是ServiceCombConcurrencyStrategy实例的问题，贡献者认识到除了代码规范的问题，还有正确性的问题，他就删除了这个branch，修改代码后，重新提交了#574这个PR，但仍然要被严格地review，即使是代码格式化的问题都不能放过。所以，如果要贡献代码，确实是对自己代码质量有很大的考验和锻炼。4. 开源社区里的人一般都会很乐意去帮助别人。这个issue的提出者并没有自己去写代码修复，但他提供了一个可能可以解决问题的思路（也就是那个链接）；有人会主动来提交PR解决对应的issue；有人会来提自己的建议，会有人来仔细地review你的代码😊

 **Willem Jiang** 2 months ago
总结的不错。围观的过程就是一个学习的过程。提升自己软件工程能力的方法有很多，通过围观PR能够避免我们走之前的弯路，并且有很多可以上手的素材。@方孝君 从这个issue你还能总结出什么和技术相关的内容吗？



Willem Jiang 2 months ago

这里面有一个技术上下文是Thread Local传递是有局限性的，在线程发生切换的时候，我们需要将ThreadLocal的变量拷贝到新的线程中，才能保证之前涉及的业务逻辑能够继续执行。



Willem Jiang 2 months ago

大家在后续提PR的时候也要想想如何描述清楚问题，让更多人能够通过阅读PR就能理解其中的上下文。



方孝君 2 months ago

这两天我看了一下serviceComb-saga的文档（其中有Omega、Alpha的概念），了解了一下Hystrix、Feign的相关概念和基本原理。我了解到Hystrix是Netflix开源的一款容错框架，它提供了线程池隔离和信号量隔离两种资源隔离策略（资源隔离是为了防止由于依赖的单个服务出现问题，导致整个服务不可用）。-----issue中描述的问题是基于一以下两个场景的：1. Hystrix默认的资源隔离方式是Thread，也即线程池隔离；2. Omega需要靠ThreadLocal变量来传递服务调用的上下文信息。-----在这两个场景下就会衍生出issue中描述的这个问题：一旦Feign开启Hystrix支持，由于资源隔离策略Thread的规则，我们是没法拿到ThreadLocal中的值的，这样服务调用的上下文信息就会丢失（或者说维持原来的值不变）-----所以问题的关键就在于，我们如何在线程上下文切换的场景下传递ThreadLocal变量。贡献者最后是参考了spring security解决securityContext 线程变量在使用hystrix时，传递线程变量的方案，自定义并发策略，针对OmegaContext里的线程变量进行了传递，并提供了可扩展的接口。

在 serviceComb 这个项目中我了解到了一些参与开源项目的规范，也学习到了一些微服务框架的技术知识。

后来因为我是王老师的学生，所以后来对 kfcoding 社区的 GitCourse 项目做了一些贡献。PR 地址：<https://github.com/kfcoding/gitcourse/pull/24>。我为 GitCourse 集成了自动化测试框架 Jest 和前端测试组件库 Enzyme，并写好了测试文件配置，还写了三个测试用例。

提这个 PR 的原因是：GitCourse 作为一个前端项目，还没有集成其自动化测试框架，也没有写过任何一个测试用例。

所以基于这是一个使用 React 框架的前端项目，我在调研了适合做 React 测试的几个测试框架和工具库，最终得出三个方案做备选：Jest+Enzyme、Mocha 全家桶、react-dom/test-utils。最终综合了 GitCourse 项目的具体情况、框架易用性与鲁棒性，最终选择了 Jest+Enzyme 这个方案。

3. 课程反馈

①

不得不说这个课程一开始列出的安排还是挺吸引人的：

Syllabus (tentative)	
1. 开放源代码介绍	9. 融入和参与上游开源项目所需要的软技能
2. CVS、SVN、Git，版本控制系统的演化	10. 社区的艺术
3. 开源协作的核心：Diff/Patch/Merge/PR	11. 开源与商业
4. 源代码阅读的艺术（一）：工具篇	12. 开放式开发（一）：以Linux Kernel为例阐明
5. 源代码阅读的艺术（二）：意义篇	13. 开放式开发（二）：以Apache HTTPD为例阐明
6. 源代码阅读的艺术（三）：实战篇	14. 开放式开发（三）：以FreeBSD为例阐明
7. 源代码阅读的艺术（四）：软技能篇	15. 开放式开发（四）：以Kubernetes为例阐明
8. 开放源代码需要了解的法律常识	16. 开放式开发（五）：以TensorFlow为例阐明

但好像后来几个以开源项目为例都没有讲，也没有什么阅读源代码的内容，直接点说就是干货比较少，而介绍性质的、软技能性质的占的比重太多。

所以建议对课程的内容在学期初期就有一个比较具体的规划，然后比较严格地按照课程安排来上课。

②

课程的内容很重要，讲课的人也很重要。有的人可能确实很牛逼，但是并不是一个好的演讲者，所以在定好课程内容的同时还要定好讲课的人，当然也可以先定人员再根据人员擅长的领域来定内容。

③

在参与 ServiceComb 和 GitCourse 的过程中，我能明显感觉到 ServiceComb 作为一个成熟的微服务框架项目，对其做贡献是相对困难的，因为其中涉及的技术栈、业务逻辑对于我们这些学生来说基本都是陌生的。所以一般有以下几点建议来改善这种学生难以参与到开源项目中去的情况：

在选择开源项目的环节之前，课堂上更具体地介绍如何参与一个技术栈完全陌生的开源项目，最好有实际例子；

开源项目的导师在宣传宣传自己项目的时候，多介绍一些该项目涉及的技术，更直接点就是直接介绍项目代码的模块划分与业务逻辑，当然一个成熟的开源项目必定有优秀的文档；

开源项目的导师可以开几个比较 easy 的 issue。

4. 参考文献

无，纯手打的。