二

48 再谈开源项目：如何选择、使用以及二次开发?

我在专栏特别放送第 3 期谈了如何高效地学习开源项目，主要聊了我在学习开源项目的一些看法和步骤。今天我们再聊开源项目，谈谈如何选择、使用以及二次开发。

软件开发领域有一个流行的原则：DRY，Don’t repeat yourself。翻译过来更通俗易懂：**不要重复造轮子**。开源项目的主要目的是共享，其实就是为了让大家不要重复造轮子，尤其是在互联网这样一个快速发展的领域，速度就是生命，引入开源项目可以节省大量的人力和时间，大大加快业务的发展速度，何乐而不为呢?

然而现实往往没有那么美好，开源项目虽然节省了大量的人力和时间，但带来的问题也不少，相信绝大部分技术人员都踩过开源软件的坑，小的影响可能是宕机半小时，大的问题可能是丢失几十万条数据，甚至灾难性的事故是全部数据都丢失。

除此以外，虽然 DRY 原则摆在那里，但实际上开源项目反而是最不遵守 DRY 原则的，重复的轮子好多，你有 MySQL，我有 PostgreSQL；你有 MongoDB，我有 Cassandra；你有 Memcached，我有 Redis；你有 Gson，我有 Jackson；你有 Angular，我有 React……总之放眼望去，其实相似的轮子很多！相似轮子太多，如何选择就成了让人头疼的问题了。

怎么办？完全不用开源项目几乎是不可能的，架构师需要更加聪明地选择和使用开源项目。形象点说：**不要重复发明轮子，但要找到合适的轮子**！但别忘了，如果你开的是保时捷，可别找个拖拉机的轮子。

选：如何选择一个开源项目

1. 聚焦是否满足业务

架构师在选择开源项目时，一个头疼的问题就是相似的开源项目较多，而且后面的总是要宣称比前面的更加优秀。有的架构师在选择时有点无所适从，总是会担心选择了 A 项目而错过了 B 项目。这个问题的解决方式是**聚焦于是否满足业务，而不需要过于关注开源项目是否优秀**。

Tokyo Tyrant 的教训

在开发一个社交类业务时，我们使用了 TT（Tokyo Tyrant）开源项目，觉得既能够做缓存取代 Memcached，又有持久化存储功能，还可以取代 MySQL，觉得很强大，于是就在业务里面大量使用了。但后来的使用过程让人很郁闷，主要表现为：

不能完全取代 MySQL，因此有两份存储，设计时每次都要讨论和决策究竟什么数据放 MySQL，什么数据放 TT。

功能上看起来很高大上，但相应的 bug 也不少，而且有的 bug 是致命的。例如所有数据不可读，后来是自己研究源码写了一个工具才恢复了部分数据。

功能确实强大，但需要花费较长时间熟悉各种细节，不熟悉随便用很容易踩坑。

后来我们反思和总结，其实当时的业务 Memcached + MySQL 完全能够满足，而且大家都熟悉，其实完全不需要引入 TT。

简单来说：如果你的业务要求 1000 TPS，那么一个 20000 TPS 和 50000 TPS 的项目是没有区别的。有的架构师可能会担心 TPS 不断上涨怎么办？其实不用过于担心，架构是可以不断演进的，等到真的需要这么高的时候再来架构重构，这里的设计决策遵循架构设计原则中的“合适原则”和”演化原则”。

2. 聚焦是否成熟

很多新的开源项目往往都会声称自己比以前的项目更加优秀：性能更高、功能更强、引入更多新概念……看起来都很诱人，但实际上都有意无意地隐藏了一个负面的问题：更加不成熟！不管多优秀的程序员写出来的项目都会有 bug，千万不要以为作者历害就没有 bug，Windows、Linux、MySQL 的开发者都是顶级的开发者，系统一样有很多 bug。

不成熟的开源项目应用到生产环境，风险极大：轻则宕机，重则宕机后重启都恢复不了，更严重的是数据丢失都找不回来。还是以我上面提到的 TT 为例：我们真的遇到异常断电后，文件被损坏，重启也恢复不了的故障。还好当时每天做了备份，于是只能用 1 天前的数据进行恢复，但当天的数据全部丢失了。后来我们花费了大量的时间和人力去看源码，自己写工具恢复了部分数据，还好这些数据不是金融相关的数据，丢失一部分问题也不大，否则就有大麻烦了。

所以在选择开源项目时，**尽量选择成熟的开源项目**，降低风险。

你可以从这几个方面考察开源项目是否成熟：

版本号：除非特殊情况，否则不要选 0.X 版本的，至少选 1.X 版本的，版本号越高越好。

使用的公司数量：一般开源项目都会把采用了自己项目的公司列在主页上，公司越大越

好，数量越多越好。

社区活跃度：看看社区是否活跃，发帖数、回复数、问题处理速度等。

3. 聚焦运维能力

大部分架构师在选择开源项目时，基本上都是聚焦于技术指标，例如性能、可用性、功能这些评估点，而几乎不会去关注运维方面的能力。但如果要将项目应用到线上生产环境，则**运维能力是必不可少的一环**，否则一旦出问题，运维、研发、测试都只能干瞪眼，求菩萨保佑了！

你可以从这几个方面去考察运维能力：

开源项目日志是否齐全：有的开源项目日志只有寥寥启动停止几行，出了问题根本无法排查。

开源项目是否有命令行、管理控制台等维护工具，能够看到系统运行时的情况。

开源项目是否有故障检测和恢复的能力，例如告警、切换等。

如果是开源库，例如 Netty 这种网络库，本身是不具备运维能力的，那么就需要在使用库的时候将一些关键信息通过日志记录下来，例如在 Netty 的 Handler 里面打印一些关键日志。

用：如何使用开源项目

1. 深入研究，仔细测试

很多人用开源项目，其实是完完全全的“拿来主义”，看了几个 Demo，把程序跑起来就开始部署到线上应用了。这就好像看了一下开车指南，知道了方向盘是转向、油门是加速、刹车是减速，然后就开车上路了，其实是非常危险的。

Elasticsearch 的案例

我们有团队使用了 Elasticsearch，基本上是拿来就用，倒排索引是什么都不太清楚，配置都是用默认值，跑起来就上线了，结果就遇到节点 ping 时间太长，剔除异常节点太慢，导致整站访问挂掉。

MySQL 的案例

很多团队最初使用 MySQL 时，也没有怎么研究过，经常有业务部门抱怨 MySQL 太慢

了。但经过定位，发现最关键的几个参数（例如，innodb_buffer_pool_size、sync_binlog、innodb_log_file_size 等）都没有配置或者配置错误，性能当然会慢。

通读开源项目的设计文档或者白皮书，了解其设计原理。

核对每个配置项的作用和影响，识别出关键配置项。

进行多种场景的性能测试。

进行压力测试，连续跑几天，观察 CPU、内存、磁盘 I/O 等指标波动。

进行故障测试：kill、断电、拔网线、重启 100 次以上、切换等。

2. 小心应用，灰度发布

假如我们做了上面的“深入研究、仔细测试”，发现没什么问题，是否就可以放心大胆地应用到线上了呢？别高兴太早，即使你的研究再深入，测试再仔细，还是要小心为妙，因为再怎么深入地研究，再怎么仔细地测试，都只能降低风险，但不可能完全覆盖所有线上场景。

Tokyo Tyrant 的教训

还是以 TT 为例，其实我们在应用之前专门安排一个高手看源码、做测试，做了大约 1 个月，但最后上线还是遇到各种问题。线上生产环境的复杂度，真的不是测试能够覆盖的，必须小心谨慎。

所以，不管研究多深入、测试多仔细、自信心多爆棚，时刻对线上环境和风险要有敬畏之心，小心驶得万年船。我们的经验就是先在非核心的业务上用，然后有经验后慢慢扩展。

3. 做好应急，以防万一

即使我们前面的工作做得非常完善和充分，也不能认为万事大吉，尤其是刚开始使用一个开源项目，运气不好可能遇到一个之前全世界的使用者从来没遇到的 bug，导致业务都无法恢复，尤其是存储方面，一旦出现问题无法恢复，可能就是致命的打击。

MongoDB 丢失数据

某个业务使用了 MongoDB，结果宕机后部分数据丢失，无法恢复，也没有其他备份，人工恢复都没办法，只能接一个用户投诉处理一个，导致 DBA 和运维从此以后都反对我们用 MongoDB，即使是尝试性的。

虽然因为一次故障就完全反对尝试是有点反应过度了，但确实故障也给我们提了一个

醒：对于重要的业务或者数据，使用开源项目时，最好有另外一个比较成熟的方案做备份，尤其是数据存储。例如，如果要用 MongoDB 或者 Redis，可以用 MySQL 做备份存储。这样做虽然复杂度和成本高一些，但关键时刻能够救命！

改：如何基于开源项目做二次开发

1. 保持纯洁，加以包装

当我们发现开源项目有的地方不满足我们的需求时，自然会有一种去改改的冲动，但是怎么改是个大学问。一种方式是投入几个人从内到外全部改一遍，将其改造成完全符合我们业务需求。但这样做有几个比较严重的问题：

投入太大，一般来说，Redis 这种级别的开源项目，真要自己改，至少要投入 2 个人，搞 1 个月以上。

失去了跟随原项目演进的能力：改的太多，即使原有开源项目继续演进，也无法合并了，因为差异太大。

所以我的建议是不要改动原系统，而是要开发辅助系统：监控、报警、负载均衡、管理等。以 Redis 为例，如果我们想增加集群功能，则不要去改动 Redis 本身的实现，而是增加一个 proxy 层来实现。Twitter 的 Twemproxy 就是这样做的，而 Redis 到了 3.0 后本身提供了集群功能，原有的方案简单切换到 Redis 3.0 即可（详细可参考这里）。

如果实在想改到原有系统，怎么办呢？我们的建议是直接给开源项目提需求或者 bug，但弊端就是响应比较缓慢，这个就要看业务紧急程度了，如果实在太急那就只能自己改了；如果不是太急，建议做好备份或者应急手段即可。

2. 发明你要的轮子

这一点估计让你大跌眼镜，怎么讲了半天，最后又回到了“重复发明你要的轮子”呢？

其实选与不选开源项目，核心还是一个成本和收益的问题，并不是说选择开源项目就一定是最优的项目，最主要的问题是：**没有完全适合你的轮子**！

软件领域和硬件领域最大的不同就是软件领域没有绝对的工业标准，大家都很尽兴，想怎么玩就怎么玩。不像硬件领域，你造一个尺寸与众不同的轮子，其他车都用不上，你的轮子工艺再高，质量再好也是白费；软件领域可以造很多相似的轮子，基本上能到处用。例如，把缓存从 Memcached 换成 Redis，不会有太大的问题。

除此以外，开源项目为了能够大规模应用，考虑的是通用的处理方案，而不同的业务其

实差异较大，通用方案并不一定完美适合具体的某个业务。比如说 Memcached，通过一致性 Hash 提供集群功能，但是我们的一些业务，缓存如果有一台宕机，整个业务可能就被拖慢了，这就要求我们提供缓存备份的功能。但 Memcached 又没有，而 Redis 当时又没有集群功能，于是我们投入 2~4 个人花了大约 2 个月时间基于 LevelDB 的原理，自己做了一套缓存框架支持存储、备份、集群的功能，后来又在这个框架的基础上增加了跨机房同步的功能，很大程度上提升了业务的可用性水平。如果完全采用开源项目，等开源项目来实现，是不可能这么快速的，甚至开源项目完全就不支持我们的需求。

所以，如果你有钱有人有时间，投入人力去重复发明完美符合自己业务特点的轮子也是很好的选择！毕竟，很多财大气粗的公司（BAT 等）都是这样做的，否则我们也就没有那么多好用的开源项目了。

小结

今天我从如何选、如何用和如何改三个方面，为你讲了如何才能用好开源项目，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，目前的云计算厂商很多都提供了和开源项目类似的系统（例如阿里云的云数据库 HBase），你倾向于购买云厂商提供的系统，还是只是将开源系统部署在云服务器上？理由是什么？

上一页 下一页