# 阿里毕玄：提升代码能力的4段经历

简介： 对于程序员而言，我始终认为代码是展现能力的关键。一个优秀程序员写的代码，和一个普通程序员写的代码是很容易看出差别的，代码是展示程序员硬实力的名片。如何提升写代码的能力，始终是一个关键的话题，不过很遗憾这篇文章其实也不是讲具体的步骤、银弹方法、武功秘籍什么的，这篇文章讲讲我自己印象中，对我写代码能力提升比较大的四段经历，也许可供参考。

第一段：第一次感受每天亿级系统的挑战

2008年，HSF的第二个版本，在当时淘宝最重要的交易中心

上线，上线当天造成淘宝网站访问巨慢，交易类的页面几乎打不开，最后靠下线HSF才恢复。

下线后开始查问题，HSF的第二个版本基于的是JBoss Remoting，JBoss Remoting在当时的版本里远程同步调用的超时时间是写死在代码里的60s，而调用的服务确实会有一些超过10几秒的现象出现，导致了Web应用处理Web请求的线程池被这些慢请求给逐渐占据，请求堆积，最终呈现出了页面打开非常慢的现象。

查清原因后，决定基于当时的Mina重写整个HSF的通信。重写的这两个月时间对我自己写代码的能力有很大的提升，无论是对网络IO方面处理的深入学习，还是在高并发系统上的深入学习。现在想想学习的方式也就是翻各类网络IO的科普资料，然后是读Mina的源码、Java网络IO的源码。并发这块的学习主要还是靠那本经典的《Java并发编程实战》，以及读Java J.U.C里的代码。这段时间的学习相比以往翻《Think in Java》之类的最大区别是，学习后付诸实践，随着HSF这个新的重写的版本的上线，基本算是逐渐真正掌握了这些部分的代码能力。

除了代码能力的提升外，得到了另外一个最大的教训就是，对于一个亿级且长时间运行的系统，很多看起来的小概率的问题都一定会成为严重的问题。这也是写高并发系统难的原因，要求必须对自己写的代码，以及自己代码调用到的各种API里的实现都非常的清楚，这样才能真正确保最终代码的鲁棒性。

## 第二段：民间"消防队"的故事

第二段对我自己写代码能力提升特别大的经历是在民间"消防队"的那段日子。淘宝在2009年故障特别多，但处理故障还没有一个标准的体系和组织，导致很多时候会出现故障出了都没什么人处理，或者处理效率不高。于是当时有个运维团队的同学拉了一些人组建了一个群，群的名字叫淘宝消防队，用来处理淘宝出现的各种故障，我很凑巧的也加入了这个群，这个群里还有另外一个整个阿里公认的超级技术大神：多隆。

一开始看到各种故障的时候，压根就不知道怎么下手。处理故障需要的通常不仅仅是写代码的能力，还需要对一个系统的全貌要有一定的掌握。例如前几年一篇特别火的文章，点击搜索背后发生了什么，其实就是要对一个系统的处理流程特别的熟悉，这在处理故障的时候是非常重要的。在了解了故障大概在哪个环节后，很重要的就是对这个环节代码运行机制的细节的掌控了，这个时候通常来说运用各种工具是非常重要的，可以有效地帮助你知道具体发生了什么，例如像系统层面的top -H之类的，Java层面的BTrace等等，都可以让你根据运行情况去定位问题。

这段时间我觉得我的提升就是靠大量的练手。故障确实有点多，一开始就靠看别人怎么处理，主要是从多隆那里学，然后是尝试自己解决一些故障，解决的越来越多后慢慢熟练度就上去了。除了解决故障能力的提升外，由于看了很多由代码层面造成的故障，对自己在写代码时如何更好的保证鲁棒性来避免故障，也是非常有帮助的。例如，我看过很多滥用

线程池造成创建了大量线程，最终导致线程创建不出来的case，就会明白自己在用线程池的场景里一定要非常清楚地控制最大的数量，包括堆积的策略等。又例如，我看过N多的因为自增长容量的数据结构导致的OOM的case，就会明白在写代码的时候不能认为一定不会发生数据结构增长到超级大，所以不做任何保护的case。这段时间我明白到的就是，写一段能运转、实现需求的代码不难，但要写一段在各种情况下都能长期稳定运行的代码是真心不容易，我觉得这是一个职业的写商业系统的程序员和只写程序玩玩的程序员的最大差别。

第三段：重写通信框架

2010年，我从中间件团队离开，去做HBase。那个时候的HBase里面的通信还是用一个非常简单的写法实现的。我想着要么就把以前HSF里用的移植到HBase里用，这个时候刚好多隆在用C给各类C的应用写一个通用的通信框架libeasy，于是就有了一次测试，我记得第一次测试的结果，就看到了原来HSF里面的通信框架的高并发能力和libeasy比相差无比巨大。我便和多隆探讨他是怎么实现的，我看看能不能学习下，在Java这边的版本里也改改，所以有了这段重写通信框架的经历。

本来以为之前在写HSF的那几年应该算是对通信框架这块的代码相关的能力掌握的不错了，在和多隆一起重写的这段过程中，才发现差距还是很大的。多隆教会了我很多细节的问题，基于NIO的通信框架的核心是用非常少的IO线程来处理IO事件（太多也没用，因为有些部分就只能串行），所以如

何高效的使用好这几个IO线程是非常关键的，要尽量减少这几个IO线程处理一些不相关的动作，另外一点就是尽量减少IO线程和业务处理线程的切换，例如后来常见的批量把一个流里的多个请求一次性丢给业务处理线程。

这段经历对自己在代码逻辑整体的细节层面更加深入地掌握是非常有帮助的，这对于写要求很高的系统是非常重要的，毕竟对于一个超大规模的系统而言，1%的提升还是可观的。

第四段：学习JVM

之前因为处理故障比较多，有段时间我开始给公司同事们分享如何处理故障，后来发现有些问题自己也讲不清楚，或者也不知道怎么处理，必须深入学习JVM才行，但其实一开始我完全摸不着门路，JVM代码打开都不知道从哪看起。

很幸运，碰到了一个同样爱好JVM又比我强很多的同学，就是撒迦，圈内通常叫R大。我和撒迦好几个周末约着在公司一起看JVM代码，有撒迦的指点，我终于算是入门了，知道大概怎么去看了，而且两个人一起看代码，互相分享和探讨，效率是非常高的。

有了这段经历，再加上继续处理着一些故障，基本上逐渐对JVM的代码实现有了更多的理解。在后来做故障分享、问题解决什么的时候终于能更好地做到知其然知所以然。同样，这对处理故障的能力、写代码的能力也是非常有帮助的，例如会更加明白以前认为的所谓的面向GC友好的代码是几个意思。也有了更深的感受，就是其实Java的代码呢，通常不会

写的太烂，因为JVM在运行期会做很多的尽可能的优化，拉到一个平均线，但要写得很好，难度是非常大的，因为需要懂JVM，懂JVM下面的OS。

总结

其实也总结不出什么，因为每个人所处的环境不一样，有不同的适合各自提升的方法。我看自己的经历总结下来，我觉得：

如果环境不具备，就给自己一个命题挑战。例如要学高并发的通信，可以尝试自己写一个和其他的做对比，做性能等的PK，这个通常提升还是会很大的。要学GC，可以尝试给自己几个题目，来控制GC的行为等，如果环境具备的话，确实会更加有利。多和优秀的程序员一起学习。我自己从多隆、撒迦身上学习到了很多很多。从很多优秀的开源代码，像Netty、OpenJDK里面也学习到了很多很多，所以多参与一些优秀的开源项目也是一个很好的提升方法，看优秀的书（例如并发里的那本《Java并发编程实战》，JVM里的《Oracle JRockit: The Definitive Guide》，《深入理解Java虚拟机》等），也一样是一种向优秀程序员学习的好方法。多多尝试解决问题/故障。这绝对是提升代码综合能力非常好的一个方法，自己工作里机会少的话，网上有大把的平台，像Stack Overflow之类的，都是很好的练习场。最后的最后，我还是想说，代码能力作为程序员的硬名片，始终是最有效的区分程序员能力的东西，"talk is cheap, show me the code"，这句话我觉得是永远成立的。

作者：开发者小助手_L