Q&A加餐丨关于代码质量，你关心的那些事儿

2019-01-08 范学雷 来自北京

《代码精进之路》

专栏上线后，有一些同学对于代码质量有关的问题还不是很清楚，有很多疑问，所以我特意做了一期 Q&A，来回答一下这些问题。

1．有没有什么技巧可以不费力地查看源代码？

这是一个好问题。但遗憾的是，我们费力的程度，主要取决于代码的作者，而不是我们自己。我想了好久，也没有找到不费力气查看源代码的技巧。

通常我自己用的办法，有时候就像剥洋葱，从外朝里看；有时候也像挖井，找到地表的一小块地儿，朝下一直挖，直到我理解了代码的逻辑关系。

如果你刚开始接触，我建议你先不要看代码，先去看 README，再去看用户指南。先把软件干什么、怎么用搞清楚。然后再去看开发者指南，搞清楚模块之间的关系、功能，理解代码中的示例。最后，再去看代码。

看代码的时候，找一个顺手的 IDE。IDE 丰富的检索功能，可以帮助我们找到一个方法，在什么地方定义的，有哪些地方使用了。

如果你还不知道要看哪一个源代码，先找一个例子开始。不管这个例子是开发指南里的，还是测试代码里的。先找出一个例子，把它读懂，然后阅读例子中调用的源代码。

比如，你要是看到示例代码调用了 Collections.unmodifiableList() 方法，如果想了解它，就查看它的规范文档或者源代码。从例子开始剥每一个你关心的方法，一层一层地深入下去。

OpenJDK 的代码评审，很多时候代码量很大。代码评审的时候，很多文档还没有出来。我一般是分层看的。先看用户接口设计的这部分代码，这一部分的代码量一般比较少。看完用户接口的设计，才能明白作者的思路和目标。这样，自己就有了一个思路，对代码的方向有了一个大致的了解。然后再看接口实现的代码，看看实现和接口是不是吻合的。这个过程中，我一般会记录下类和方法之间的依赖关系，也会顺着依赖关系来理解代码的逻辑关系。

好的代码，有清晰的分割和层次，逻辑清晰，代码的行文一般也是简单直观，读起来比较容易。不好的代码，阅读起来就费力得多了。

2. 代码质量和工作效率的矛盾如何取舍?

这个问题有一个隐含的假设，就是代码质量和工作效率不可兼得。这本身是个常见的误区。这个误区也给了我们一个看似成立的借口：要么牺牲代码质量，要么牺牲工作效率。

代码质量和工作效率，是不是矛盾的呢？这取决于我们观察的时间、地点以及维度，甚至我们是怎么定义效率的。

如果给我们一个小时的时间，看看谁写的代码多。不顾及代码质量的也许会胜出（只是也许，我们后面再说为什么只是也许）；认真设计、认真对待每一行代码的也许会败北（也只是也许）。

短期内代码写得多与否，我们可以把这个比喻成“走得慢，还是走得快”的问题。

如果给我们半年的时间，那些质量差的代码，编写效率也许可以和质量好的代码保持在同一水准，特别是软件还没有见到用户的时候。

如果给我们一年的时间，软件已经见到了用户，那么质量差的代码的编写效率，应该大幅度落后于优质代码了。甚至生产这些代码的团队，都被市场无情淘汰了。

看谁的代码能够长期赢得竞争，我们可以把这个比喻成“到得慢，还是到得快”问题。

为什么会这样呢？ 一小时内，什么都不管，什么都不顾，怎么能不多产呢！

可是，不管不顾，并不意味真的可以高枕无忧。需求满足不了就会返工，程序出了问题也会返工，测试通不过还会返工……每一次的返工，都要你重新阅读代码，梳理逻辑，修改代码。

有很多时候，你会发现，这代码真是垃圾，没法改了，只有推倒重来。

这个时候再回过头看看这种代码编写的背景，你能说这是一种高效率的行为吗？

这就相当于，一个马拉松比赛，前 1000 米你在前头，后来你就要往回跑。1000 米这个槛，有人跑一次就够了，你要是跑七八次，还谈什么效率呢。这种绝望的事情看似荒唐，其实每天都会发生。

为什么会这样呢？ 因为在软件开发的过程中，遗留的问题需要弥补，这就类似于往回跑。所以，走得快，不一定到得快。

你不妨记录一下三个月以来，你的工作时间，看看有多少时间是花在了修修补补上，有多少时间是花在了新的用户需求上。这样，对这个问题可能有不一样的感受。

另外，**是不是关注代码质量，就一定走得慢呢？**

其实也不是这样的。比如说，如果一个定义清晰，承载功能单一的接口，我们就容易理解，编码思路也清晰，写代码就又快又好。可是，简单直观的接口怎么来？我们需要花费大量的时间，去设计接口，才能获得这样的效果。

为什么有的人一天可以写几千行代码，有的人一天就只能写几十行代码呢？这背后最重要的一个区别就是心里有没有谱，思路是不是清晰。几千行的代码质量就比几十行的差吗？ 也不一定。

你有没有遇到这样的例子，一个同学软件已经实现一半了，写了十万行代码。另一个熊孩子还在吭哧吭哧地设计接口，各种画图。当这个同学写了十五万行代码的时候，完成一大半工作的时候，那个熊孩子已经五万行代码搞定了所有的事情。你想想，效率到底该怎么定义呢？

那个熊孩子是不是没有能力写二十万行代码呢？不是的，只要他愿意，他也许可以写得更快。只是，既然目标实现了，为什么不去聊聊天，喝喝咖啡呢？搞这么多代码干啥！你想想，效率能用代码数量定义吗？

就单个的程序员而言，代码质量其实是一个意识和技能的问题。当我们有了相关的意识和技能以后，编写高质量的代码甚至还会节省时间。如果我们没有代码质量的意识，就很难积累相关的技能，编写代码就是一件苦差事，修修补补累死人。

有很多公司不愿意做代码评审，效率也是其中一个重要的考量。大家太注重一小时内的效率，而不太关切一年内的效率。如果我们将目光放得更长远，就会发现很多不一样的东西。

比如说代码评审，就可以减少错误，减少往回跑的频率，从而节省时间。代码评审也可以帮助年轻的程序员快速地成长，降低团队出错的机率，提高团队的效率。

有一些公司，定了编写规范，定了安全规范，定了很多规范，就是执行不下去，为什么呢？没有人愿意记住那么多生硬的规范，这个时候，代码评审就是一个很好的方法，有很多眼睛看着代码，有反馈，有讨论，有争议，有建议，团队能够更快地形成共识，找出问题，形成习惯，共同进步。看似慢，其实快。

英文里，有一句经典的话 “Run slowly, and you will get there faster”。汉语的说法更简洁，“**因为慢，所以快**”。

一般情况下，通常意义上的软件开发，如果我们从产品的角度看，我认为高质量的代码，会提升工作的效率，而不是降低工作效率。

当然，也有特殊情况。比如我们对质量有着偏执般的追求，这时候，效率就不是我们的首选项。也有情况需要我们在五秒以内眨眼之间就给出代码，这时候，质量也不是我们的首选项。

代码的质量该怎么取舍呢？这取决于具体的环境，和你的真实目标。

3. 你加入了 Java SE 团队，经历了从 JDK 1.5.0 到 JDK 12 的整个迭代过程，这个阶段中，Java 开发的流程都经历了哪些迭代?

在十多年间，Java 开发的流程有很多大大小小的调整。影响最大的，我觉得主要有三个。

第一个变化是更加开放了。Java 开源以后，不仅仅是把代码开放出来，开发流程也开放了出来。OpenJDK 有详细的开发人员手册，告诉大家怎么参与到 OpenJDK 社区中来。

OpenJDK 开放了 Java 改进的流程，这就是 JEP（JDK Enhancement-Proposal & Roadmap Process）。每一个 Java 的改进，从雏形开始，一直到改进完成，都要经过 OpenJDK 社区的讨论、评审。什么都要经过 OpenJDK 讨论，这效率不是变慢了吗？其实，这种开放反而加快了 Java 的演进。

创新性的想法第一时间就送到用户面前，接受用户的审视。

一个项目是好还是坏？做还是不做？该怎么做？这都在用户可以“挑剔”的范围内。Java 的演进，也从少数的专家委员会模式，变更为小步快走的大集市模式。

OpenJDK 也开放了 Java 代码评审的流程。现在，几乎所有的变更，都是通过 OpenJDK 进行的。为什么要变更？变更的是什么？变更带来的影响有哪些，都要描述得清清楚楚。而且，任何人都可参与评审，都可以发表意见。如果有兼容性的影响，用户会在第一时间发现，而不是等到系统出了问题再来修补。透明化带来的好处就是，有更多的眼睛盯着 Java 的变更，代码的质量会更好，潜在的问题也会更少。

第二个变化是研发节奏更快了。Java 的版本演进，从传统的好几年一个版本，变更为半年一个版本。两三年一个版本的演进模式，使得 Java 的任何改进，都要两三年以后见。即使这些改进已经成熟了，也得在代码库里躺着，到不了用户的场景里。没有用户反馈，产品的质量也就没有经过真实的检验了，没有改进的真实建议。这其实是一种浪费，效率会变低。

第三个变化是自动化程度提高了。现在，OpenJDK 提交的每一个变更，都会自动运行很多的测试。如果这个变更引起了测试错误，测试系统会给参与者发邮件，指明出错的测试，以及潜在的怀疑对象。变更提交前，我们也可以发出指令，运行这些测试。这些自动化的测试，可以提高代码的质量，减轻工程师的压力，提高工作的效率。

4. 您是 JDK 11 TLS 1.3 项目的 leader，在这个项目中，你对代码安全又是怎么理解的呢?

代码的安全，我一直以为是一个见识问题。一个安全问题，你见识到了，认识到了，你就会在代码里解决掉。没有认识到安全问题，可能想破脑袋，也不知道问题出在哪。

比如说，TLS 1.3 废弃掉了密码学的很多经典算法，包括 RSA 密钥交换、链式加密算法等。如果去你去查看经典的密码学教材，你会发现这些算法都被看做牢不可破的算法，全世界的每一粒沙子都变成 CPU，也破解不了它们。

可是，站在 2019 年再来看这些算法，各有各的问题，有的破解也就是几分钟的事情。那我们还应该使用这些算法吗？当然要想办法升级。可现实是，你根本不知道这些算法已经有问题了。当然，也想不到去升级使用这些算法的应用程序。这就是我们说的见识。知道了，你才能想到去解决。

见识是一个坏东西，需要我们看得多、见得多，才能够拥有。甚至，需要付出代价，比如遭受到黑客攻击。

见识也是一个好东西，见得越多，看得越多，你构筑起来的竞争优势就越明显。随着阅历的增长，见识也会增强，竞争力就提高了。

如果一个东西，每个人三秒就可以掌握，那当然是好的。但同时，它就算不上你的优势了。即使有优势，也只是三秒钟的差距。

另一个常见的问题，就是认为安全的代码牺牲效率。

编写安全的代码，会不会牺牲工作的效率呢？一般情况下，这对效率的影响几乎可以忽略不计。 比如说，一个公开接口，我们不应该信任用户输入的数据，应该检查用户输入数据的边

界和有效性。做这样的检查，增加不了多少代码量，反而会让我们的思路变得清晰。再编写具体的业务逻辑的时候，编码的效率就变高了，甚至还会减少代码量。

就拿 TLS 1.3 来说，当废弃掉一些经典的算法时，一幅全新的画面就出现在我们面前。TLS 协议的设计更加简单，更有效，效率也会翻倍地提升。

代码质量、工作效率、软件性能、代码安全，这些东西可以作为基准，但是不适用拿来对比。如果非要单纯地从概念上对比，看看有没有冲突，没有一点儿现实意义。安全的代码会牺牲软件性能吗？ 不一定。重视代码质量，就会牺牲工作效率吗？也不一定。

今天挑了几个同学的问题来回答。其实关注代码质量这种事情，就像爬山一样，每个人都会，但不是所有人都能爬到最后。会当凌绝顶，一览众山小。当自己在山峰上爬得越来越高的时候，再回过头，你会发现自己和身边的人已经不一样了。

如果你觉得这篇文章对你有所启发，欢迎你点击“请朋友读”，把它分享给你的朋友或者同事。

精选留言 (16)

置顶

2019-01-08

第二个问题个人认为目标还是——经济

关于第二个问题，个人认为可能不能算是鱼和熊掌，可能更偏向于鱼和鱼竿。

一面是很多鱼，而另一面是一根鱼竿。二选一。
另外效率这个词，个人认为不恰当。效率是保障质量的前提下更高的产出。个人把这个理念替换为速度。

速度质量二选一。理想化肯定是质量。但是实际上还要参考实际情况。

类似于我们公司的团队，碰到这样的项目，明天就要求能看到效果。都来不及做详细设计，直

接把以前的项目拿来改一改调试一下就拿去演示了。然后后续再进行优化调试完善。

这种硬环境下，不是不保障。而是无暇保障。
另一种情况，是有些功能模块，重要性不是很高，甚至根本不会有人用。只是个凑数投标的模块，但是这个模块测试以及优化的话可能很复杂。这时候，去花费大量时间精力在上面明显是不合适的。就假设，这个模块定价1000，但是为了这个模块的测试优化就花了800的成本，公司是不愿意为你的高质量代码买账的。

再举个反例。某模块追求速度开发。两天做完，节约了一天代码优化以及测试的步骤。但是未来的某天，发现这里有个坑。因为代码很久没有看过。导致这个坑难以找到问题所在，最后导致功能模块几乎重写。这里就是不应该因为速度舍弃质量的例子。

综上所述，个人认为，取舍的终极目标。还是老师之前说到的——经济

但是达到这个，并不容易。不仅仅是编写高质量代码的习惯与能力。还有不可或缺的经验，以至于可以灵活取舍。

但是——我们至少要有高质量编码的能力与基础。这样，在你有经验的时候才可以做到灵活的取舍。

作者回复: 能力和习惯是基础啊。鱼和鱼竿的类比，我第一次看到，很有意思👍

13

李英权
2019-01-08

最近接手一个质量不高的JAVA工程，读代码过程中 摸索出一个经验——用eclipse的bookmark功能为代码创建索引，现实世界的代码库 大多数像是没有索引的图书馆，运气好的你遇到有文档的项目 也不过是残缺的过期的和错误的引索。
所以需要你去重建准确的索引，eclipse的bookmark用好了 可以达到这个目的。

作者回复: 这是个好办法，小伙伴找找其他的IDE有没有类似的功能。

共 2 条评论>

13

余山头
2019-01-13

作为团队负责人，以前推行过upsource作为代码审查工具，这工具和idea结合起来，非常好

用。结果怎么样呢，发现工具是次要的，项目各种赶，各种应标废标，各种演示项目。能谈下的项目经过商务的一再拖延最后留给研发的时间少的可怜，结果就是导致低效的加班赶进度，代码都不忍直视，怕小心脏受不了。理想很丰满，现实很骨感，对于主要业务是短期外包的公司来说，只希望用廉价劳动力快速交差。

作者回复: 理解这种状况！ 这种加班其实是恶性循环，时间越紧，代码质量越难以保证；代码质量越差，能复用的东西积累越难，加班就越多。

6

xavier
2019-01-08

老师讲得很好，我觉得很多人跟我一样，就因为见识不够，不知道如何去编写高质量的代码。老师可以提供一些实践性的项目或者资源，供大家学习、操作。

作者回复: 专栏的文章就像一个引子，练手题的就是引玉的砖，留言区有很多好东西。我们用好留言区，多参与，多评论。

你的建议很好，我想一想有没有可以参与的项目。OpenJDK当然是现成的一个项目。学了算法，你可以看看是不是可以优化Java的类库。学了接口设计，你可以看看Java类库哪些接口设计的好，哪些设计的不好。 你目前工作的项目，😁也是一个可以改进的项目。

5

liu
2019-01-08

广博精深，做好取舍。大处着眼，细处着手；质量从过程抓起，从细节抓起，做好质量把控；同时注重反馈总结。方向对了，过程把握好了，结果不会太坏。

作者回复: 👍

3

陈威洋
2021-04-21

非常感谢评论区的哥哥姐姐分享的经验~

1

Sisyphus235

2019-05-21

代码的开发最重要的影响因素之一是经济，有时候为了效率，很多部分我都会写 TODO，因为交付时间总是很急迫，来不及认真的设计架构和实现。有过几次因为这样开发，不长时间就要停下来重构的经历后，我现在往往会在开发新功能的时候，把涉及部分的 TODO 一起做了，局部重构，以解决开发速度和代码质量的问题。

共 1 条评论 >

1

hyeebeen

2019-03-23

是否可以通过调用链路来帮助我们从整体去认识类与类、方法与方法之间的关系，然后再去“剥洋葱”?

作者回复: 也是个好办法！debug的调用堆栈分析就是这么做的。

1

若尘

2019-01-09

从编写，测试，交付，使用的大视角拆解效率，结论是质与量两个变量，质跟效率相关性更高，又一次证明了慢慢做会更快

作者回复: “大视角”👍

1

虢國技醬

2019-01-08

打卡

作者回复: 加油

1

allean

2019-01-08

老师您好，质量高的代码是否意味着使用恰当的设计模式?

作者回复: 也不一定，设计模式是经验总结，只是好代码的一小部分。不过，学设计模式有助于理解接口设计。

1

ifelse

2022-08-04 来自浙江

学习了

ifelse

2022-08-04 来自浙江

如果一个东西，每个人三秒就可以掌握，那当然是好的。但同时，它就算不上你的优势了。即使有优势，也只是三秒钟的差距。--记下来

aoe

2021-12-17

谢谢分享

Geeker

2020-03-11

感谢老师!

老吴

2019-02-20

养肥了 可以开始看了

作者回复: 哈哈，打开方式多样啊。