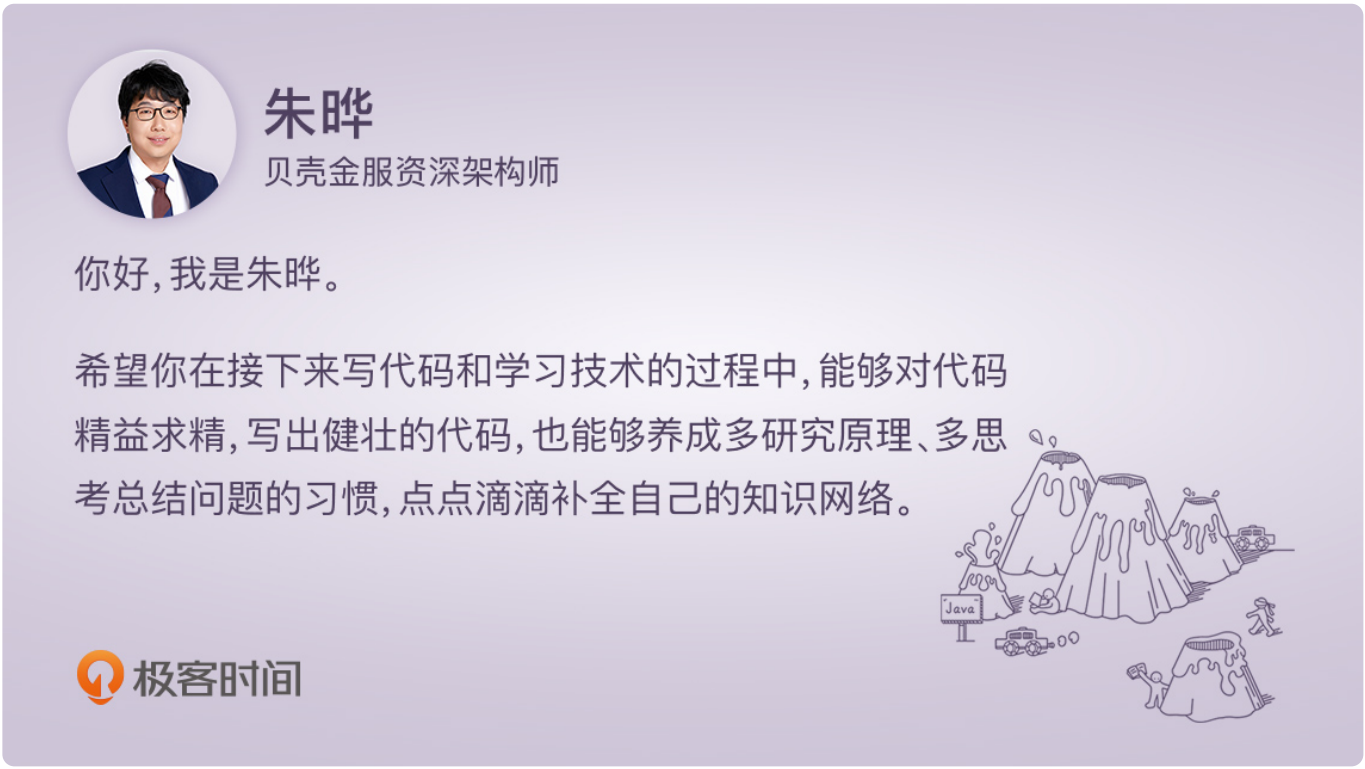
结束语 | 写代码时，如何才能尽量避免踩坑？

2020-05-28 朱晔

Java业务开发常见错误100例 进入课程 >



讲述：王少泽
时长 11:09 大小 10.22M

你好，我是朱晔。

这个课程要告一段落了，在这里我要特别感谢你一直以来的认可与陪伴。于我而言，虽然这半年多以来我几乎所有的业余时间都用了在这个课程的创作，以及回答你的问题上，很累很辛苦，但是看到你的认真学习和对课程内容的好评，看到你不仅收获了知识还燃起了钻研源码的热情，我也非常高兴，深觉一切的辛苦付出都是甜蜜的。

相信一路走来，你不仅理解了业务代码开发中常见的 130 多个坑点的解决方式，也知其根本原因，以及如何使用一些常用工具来分析问题。这样在以后遇到各种坑的时候，你就更加能有方法、有信心来解决问题。

如何尽量避免踩坑？

不过，学习、分析这些坑点并不是我们的最终目的，在写业务代码时如何尽量避免踩坑才是。所以，接下来，我要重点和你聊聊避免踩坑的一些方法。

所谓坑，往往就是我们意识不到的陷进。虽然这个课程覆盖了 130 多个业务开发时可能会出错的点，但我相信在整个 Java 开发领域还有成千上万个可能会踩的坑。同时，随着 Java 语言以及各种新框架、新技术的产生，我们还会不断遇到各种坑，很难有一种方式确保永远不会遇到新问题。

而我们能做的，就是尽可能少踩坑，或者减少踩坑给我们带来的影响。鉴于此，我还有 10 条建议要分享给你。

第一，遇到自己不熟悉的新类，在不了解之前不要随意使用。

比如，我在并发工具这一讲中提到的 CopyOnWriteArrayList。如果你仅仅认为 CopyOnWriteArrayList 是 ArrayList 的线程安全版本，在不知晓原理之前把它用于大量写操作的场景，那么很可能会遇到性能问题。

JDK 或各种框架随着时间的推移会不断推出各种特殊类，用于极致化各种细化场景下的程序性能。在使用这些类之前，我们需要认清楚这些类的由来，以及要解决的问题，在确认自己的场景符合的情况下再去使用。

而且，越普适的工具类通常用起来越简单，越高级的类用起来越复杂，也更容易踩坑。比如，代码加锁这一讲中提到的，锁工具类 StampedLock 就比 ReentrantLock 或者 synchronized 的用法复杂得多，很容易踩坑。

第二，尽量使用更高层次的框架。

通常情况下，偏底层的框架趋向于提供更多细节的配置，尽可能让使用者根据自己的需求来进行不同的配置，而较少考虑最佳实践的问题；而高层次的框架，则会更多地考虑怎么方便开发者开箱即用。

比如，在HTTP 请求这一讲中，我们谈到 Apache HttpClient 的并发数限制问题。如果你使用 Spring Cloud Feign 搭配 HttpClient，就不会遇到单域名默认 2 个并发连接的问

题。因为，Spring Cloud Feign 已经把这个参数设置为了 50，足够应对一般场景了。

第三，关注各种框架和组件的安全补丁和版本更新。

比如，我们使用的 Tomcat 服务器、序列化框架等，就是黑客关注的安全突破口。我们需要及时关注这些组件和框架的稳定大版本和补丁，并及时更新升级，以避免组件和框架本身的性能问题或安全问题带来的大坑。

第四，尽量少自己造轮子，使用流行的框架。

流行框架最大的好处是成熟，在经过大量用户的使用打磨后，你能想到、能遇到的所有问题几乎别人都遇到了，框架中也有了解决方案。很多时候我们会以“轻量级”为由来造轮子，但其实很多复杂的框架，一开始也是轻量的。只不过是，这些框架经过各种迭代解决了各种问题，做了很多可扩展性预留之后，才变得越来越复杂，而并不一定是框架本身的设计臃肿。

如果我们自己去开发框架的话，很可能会踩一些别人已经踩过的坑。比如，直接使用 JDK NIO 来开发网络程序或网络框架的话，我们可能会遇到 epoll 的 selector 空轮询 Bug，最终导致 CPU 100%。而 Netty 规避了这些问题，因此使用 Netty 开发 NIO 网络程序，不但简单而且可以少踩很多坑。

第五，开发的时候遇到错误，除了搜索解决方案外，更重要的是理解原理。

比如，在OOM这一讲，我提到的配置超大 server.max-http-header-size 参数导致的 OOM 问题，可能就是来自网络的解决方案。网络上别人给出的解决方案，可能只是适合“自己”，不一定适合所有人。并且，各种框架迭代很频繁，今天有效的解决方案，明天可能就无效了；今天有效的参数配置，新版本可能就不再建议使用甚至失效了。

因此，只有知其所以然，才能从根本上避免踩坑。

第六，网络上的资料有很多，但不一定可靠，最可靠的还是官方文档。

比如，搜索 Java 8 的一些介绍，你可以看到有些资料提到了在 Java 8 中 Files.lines 方法进行文件读取更高效，但是 Demo 代码并没使用 try-with-resources 来释放资源。在文

件 IO这一讲中，我和你讲解了这么做会导致文件句柄无法释放。

其实，网上的各种资料，本来就是大家自己学习分享的经验和心得，不一定都是对的。另外，这些资料给出的都是 Demo，演示的是某个类在某方面的功能，不一定会面面俱到地考虑到资源释放、并发等问题。

因此，对于系统学习某个组件或框架，我最推荐的还是 JDK 或者三方库的官方文档。这些文档基本不会出现错误的示例，一般也会提到使用的最佳实践，以及最需要注意的点。

第七，做好单元测试和性能测试。

如果你开发的是一个偏底层的服务或框架，有非常多的受众和分支流程，那么单元测试（或者是自动化测试）就是必须的。

人工测试一般针对主流程和改动点，只有单元测试才可以确保任何一次改动不会影响现有服务的每一个细节点。此外，许多坑都涉及线程安全、资源使用，这些问题只有在高并发的情况下才会产生。没有经过性能测试的代码，只能认为是完成了功能，还不能确保健壮性、可扩展性和可靠性。

第八，做好设计评审和代码审查工作。

人都会犯错，而且任何一个人的知识都有盲区。因此，项目的设计如果能提前有专家组进行评审，每一段代码都能有至少三个人进行代码审核，就可以极大地减少犯错的可能性。

比如，对于熟悉 IO 的开发者来说，他肯定知道文件的读写需要基于缓冲区。如果他看到另一个同事提交的代码，是以单字节的方式来读写文件，就可以提前发现代码的性能问题。

又比如，一些比较老的资料仍然提倡使用MD5 摘要来保存密码。但是，现在 MD5 已经不安全了。如果项目设计已经由公司内安全经验丰富的架构师和安全专家评审过，就可以提前避免安全疏漏。

第九，借助工具帮我们避坑。

其实，我们犯很多低级错误时，并不是自己不知道，而是因为疏忽。就好像是，即使我们知道可能存在这 100 个坑，但如果让我们一条一条地确认所有代码是否有这些坑，我们也很难办到。但是，如果我们可以把规则明确的坑使用工具来检测，就可以避免大量的低级错误。

比如，使用 YYYY 进行日期格式化的坑、使用 == 进行判等的坑、List.subList原 List 和子 List 相互影响的坑等，都可以通过阿里 P3C 代码规约扫描插件发现。我也建议你为 IDE 安装这个插件。

此外，我还建议在 CI 流程中集成Sonarqube代码静态扫描平台，对需要构建发布的代码进行全面的代码质量扫描。

第十，做好完善的监控报警。

诸如内存泄露、文件句柄不释放、线程泄露等消耗型问题，往往都是量变积累成为质变，最后才会造成进程崩溃。如果一开始我们就可以对应用程序的内存使用、文件句柄使用、IO 使用量、网络带宽、TCP 连接、线程数等各种指标进行监控，并且基于合理阈值设置报警，那么可能就能在事故的婴儿阶段及时发现问题、解决问题。

此外，在遇到报警的时候，我们不能凭经验想当然地认为这些问题都是已知的，对报警置之不理。我们要牢记，所有报警都需要处理和记录。

以上，就是我要分享给你的 10 条建议了。用好这 10 条建议，可以帮助我们很大程度提前发现 Java 开发中的一些坑、避免一些压力引起的生产事故，或是减少踩坑的影响。

最后，正所谓师傅领进门，修行靠个人，希望你在接下来学习技术和写代码的过程中，能够养成多研究原理、多思考总结问题的习惯，点点滴滴补全自己的知识网络。对代码精益求精，写出健壮的代码，线上问题少了，不但自己的心情好了，也能得到更多认可，并有更多时间来学习提升。这样，我们的个人成长就会比较快，形成正向循环。

另外，如果你有时间，我想请你帮我填个课程问卷，和我反馈你对这个课程的想法和建议。今天虽然是结课，但我还会继续关注你的留言，也希望你能继续学习这个课程的内容，并会通过留言区和你互动。

你还可以继续把这个课程分享给身边的朋友和同事，我们继续交流、讨论在写 Java 业务代码时可能会犯的错儿。

上一篇　加餐6 | 这15年来，我是如何在工作中学习技术和英语的?

下一篇　结课测试 | 关于Java业务开发的100个常见错误，你都明白其中缘由了吗?

精选留言 (27)

写留言

insight

2020-05-28

非常感谢老师的课，这是我第一门在极客时间听完的课，让我学到了很多东西，解决了很多编程中遇到的问题，流处理的教学也让我打开了新大门，爱上了这种工具，期待老师的新课程!

作者回复: 大家的留言我就不一一回复了，如果觉得有帮助欢迎大家推荐给身边的朋友，也欢迎提出意见建议，或者是说说你想学的内容，你的困惑，说不定还有第二季，还是这个风格还是这个味道，继续一起学习和进步。有成长学习的问题也可以在这里留言，技术相关问题可以继续在相关主题的文章下留言。

6

Darren
2020-05-28

谢谢老师，学到了很多，也收获了很多，期待能出下一个专栏

作者回复: 课代表来了

1 4

winner_0715
2020-05-28

意犹未尽，老师有别的课程计划吗

展开

4

2020-05-29

我都没发现，居然都半年过去了。

这应该是我第一个，从第一篇追到结语的专栏了。

无疑，这个专栏学习的过程，也是自己成长的一个重要过程之一。...

展开

作者回复: 是的，专栏能带给大家的知识点是有限的，正确的思考思维方式对以后自己解决问题就很有用了，加油

3

吴世东
2020-05-28

朱老师的专栏讲到的都是我们实际工作中经常遇到的问题，专栏的每篇文章都认真学习过。计划再二刷一遍，相信会有不一样的思考。

展开

2

张先生、
2020-05-28

感谢老师的辛苦付出，这个课程真的让我学到了很多，都是和平时开发息息相关的，很实

用

2

子一
2020-05-29

感谢老师的付出，毫不避讳的说对于一个30个以上专栏的用户来说基本上属于我心中的top3，课的实战性很强，期待老师的下一门课程，提点建议老师能付考虑一下来个测试实战的专栏，测试用例是一个老大难的问题啊

展开

作者回复: 感谢认可

1

梦倚栏杆
2020-05-29

这一篇看的非常有压力：发现提到的某些坑竟然已经忘记了，我明明也下载了老师的代码，跟着跑了。

老师这门课程真的非常给力，有一些确实是想当然的再用，也没发现问题就以为没有问题。老师能再加一篇技术设计需要考虑的点吗？ 我最近遇到一个问题：明明是一个很小...

展开

作者回复: 其实你说的问题不是绝对意义上的坑。对于一个高并发的分布式系统，看似运作稳定，其实哪怕有任何一个小改动，都可能影响到平衡引起重大问题。所以一般这样的系统都需要定期全链路压测，每次重大发布都需要独立模块压测，通过压测才能确定系统的整体吞吐是否可以满足要求。我们也遇到过增加一个insert的SQL这样的小需求增加了2ms的时间导致队列堵塞（服务处理能力不够）的问题，所以不能看需求是不是小，关键还是在于知道系统的容量余量。

1

kyl
2020-05-28

非常棒的课程 期待朱老师下一个课程

展开

1

待时而发
2020-05-28

感谢老师，这么快就结束了。通过这个我也进去了老师的博客 挖了很多老师的博客来看 挺有意思的 把老师博客看完收获也很大 哈哈

展开

1

海战

2020-05-28

经验丰富，讲解清晰受用

展开

1

mgs2002

2020-05-28

有点意犹未尽，很多坑都是日常开发中遇到的

展开

1

Jeff.Smile

2020-05-28

今天我还在想读源码的好:第一，熟悉常使用框架的运行核心原理。第二，欣赏高手写的代码。不过我觉得这还不够，归根结底，都是为了让自己的设计和实现能力得到提升，不然好处再多，少了这个都是自娱自乐!

1

黄琼

2020-05-28

一路走来的一门课，基本上都跟上没掉队。最大的感受是，用什么组件或者特性要知其所以然。

很多问题或者坑的发生，是不能正确评估这段代码下去后，会发生什么样的动作。

...

展开

1

ACGCoder

2020-05-28

跟着老师的课程，最重要的是思维串起来了，真的是受益匪浅，非常感谢老师!

1

Geek_3b1096
2020-05-28

依依不舍期待老师下一讲

展开

1

Demon.Lee
2020-05-28

多研究原理、多思考总结问题的习惯，点点滴滴补全自己的知识网络！谢谢老师！
这么快就结束了？不过瘾啊，真的不过瘾，二刷走起来！

1

终结者999号
2020-05-28

感谢老师的讲解，真的是发人深省，很多地方都讲的很好

1

QQ怪
2020-05-28

这个课程含金量真的很高，受益匪浅，受教了，感谢老师

1

..
2020-05-28

不知不觉就已经结束了，还有一种意犹未尽的感觉……

展开

1