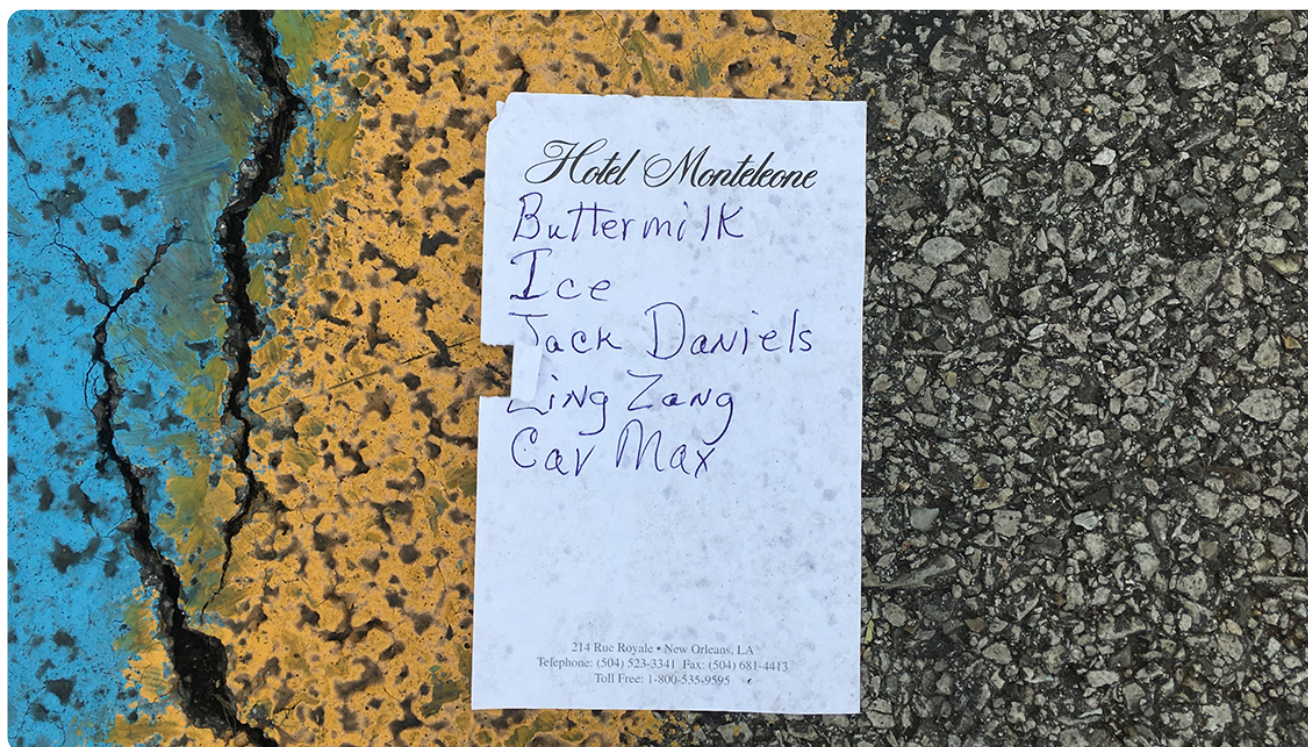


极客时间 | 29 | 编写经济代码的检查清单

time.geekbang.org 已更新2019年3月11日

29 | 编写经济代码的检查清单

范学雷 2019-03-11



00:00

10:14

讲述：刘飞 大小：18.77M

通过前面十几讲的学习，我们已经把代码“经济”篇的内容学习完了。今天，我们一起把前面讨论到的观点总结一下，并探索一下编写经济代码时的最佳实践检查清单。

为什么需要经济的代码？

我在[经济篇这一模块开始的时候](#)讲过这个问题，这里再来简单回忆一遍。

1. 提升用户体验

一致性的性能体验，是软件产品赢得竞争的关键指标。复杂的，反应迟钝的软件，很难赢得用户的尊敬。

2. 降低研发成本

通过降低软件的复杂度，提高软件的复用，提前考虑性能问题，可以降低软件研发成本，缩短软件开发周期。

3. 降低运营成本

经济的代码可以降低软件的复杂度，提高计算资源的使用效率，降低运营成本。

4. 防范可用性攻击

复杂的代码和性能低下的代码，更容易成为黑客攻击的目标。如果一个服务器，需要耗费很多资源才能处理一个请求，那么数量很少的模拟请求攻击，就可以导致服务器瘫痪。

怎么编写经济的代码？

既然我们都知道编写经济代码的重要性，那么如何让自己的代码经济又高效呢？

在前面的文章中，我给你从避免过度设计、选择简单直观、超越线程同步、减少内存使用、避免性能陷阱、规模扩张能力等角度探讨了一些方法，下面我提炼了几个点，我们再来重新温习一遍。

1. 避免过度设计

我们从[需求和设计两个角度](#)探讨了代码的经济问题。

避免需求膨胀的方式主要有两个，第一个是识别核心需求，我们要从用户的角度出发，知道什么是核心需求，什么是衍生需求，什么是无效需求。就像建火车站一样，能够满足乘客出行需求的就是好的设计方案，其他方面再细心认真，起到的也只是锦上添花的效果。那么有一些功能现在好像用不上，但又必须做，该怎么办呢？这就用到了第二个方法：迭代演进，有所主次。

避免过度设计和避免需求膨胀一样，我们需要时刻问自己，什么是现在就必须做的？什么是必须做的？

搞清楚这两个问题，有助于我们始终关注核心需求和核心问题，为代码的质量和编码的效率打好基础。

避免需求膨胀和过度设计，是编写经济代码时需要关注的基础性问题。

2. 选择简单直观

我们用了两篇文章，讨论了[让代码简单直观的原则和实践](#)。

设计一个简单直观的接口，首先，我们要从问题开始。把问题逐步拆解成一个个已经完全穷尽的小问题，这就是我讲到的“相互独立，完全穷尽”原则。在拆解的过程中，软件的接口与接口之间的关系会自然而然地产生。

此外我们还要注意，一个接口只应该做一件事情，如果这个情况太理想化，就要想办法减少接口的依赖关系。

一定记住这个经过实践检验的理念：选择最简单，最直观的解决方案。

3. 超越线程同步

现实中，线程同步需要排队，有损效率。我们用了两篇文章，主要讲了[该怎么超越线程的同步](#)。

只要满足这三个条件中的一个，我们就不需要线程同步了：使用单线程；不关心共享资源的变化；没有改变共享资源的行为。

我们要重新认识 Java 的“final”这个限定词。使用了限定词“final”的类变量，只能被赋值一次，而且只能在实例化之前被赋值。这样的变量，就是不可变的量。如果一个类的所有的变量，都是不可变的，那么这个类也是不可变的。不可变的量是无法改变的资源，不需要线程同步。

如果线程同步不可避免，就要想办法减少线程同步时间。

另外，我们还讨论了如何使用同步的代码，调动异步的事件。异步编程，可以大幅度降低线程同步的使用，更有效地使用计算机资源。

4. 减少内存使用

内存管理对任何一门编程语言来讲都是一个难题。我们用了两篇文章，讨论了[提高内存使用效率](#)的一些方法。

减少内存的使用主要有两个方法，第一个方法是减少实例的数量，第二个办法是减小实例的尺寸。

如何减少实例的数量呢？我们可以使用数据静态化的处理方式（比如枚举类型）、用单实例模式、延迟分配技术等。

在减小实例尺寸这一模块，我们要尽量减少独占的空间，尽量使用共享的实例。不可变（immutable）的资源禁止修改（immutable）的资源，是两半理想的共享资源。

(immutable) 的资源 and 禁止修改 (unmodifiable) 的资源，是两类理想的共享资源。

5. 规避性能陷阱

我们要学会规避一些[常见的性能陷阱](#)，比如字符串的操作、内存泄露、未正确关闭的资源和遗漏的 hashCode 等。

另外，我们还顺便使用了一个基准测试工具 JMH，并通过它分析了一些性能陷阱。我们要有意地使用一些性能测试工具，通过测试数据来认识、积累性能问题的最佳实践。

6. 规模扩张能力

经济的代码需要跟得上产品的规模扩张。我们要理解规模垂直扩张和规模水平扩张这两种方式，特别是[支持规模水平扩张](#)。

状态数据是影响规模水平扩张能力的最重要的因素。分离无状态数据、提供无状态服务，减少有状态服务的规模，是提升规模水平扩张能力的最佳实践。

经济代码的检查清单

了解了编写经济代码的方法论之后，我们再来看下检查清单。这个检查清单是经济篇这一模块的凝练，也是我看代码的时候，通常会使用的检查点。你也可以参考一下。

如果有检查点没有通过，那么你在阅读代码的时候，就要集中注意力，深入分析；在设计和编写代码的时候，要花时间衡量、妥协、改进；在评审代码的时候，要问清楚为什么这么做，能不能有所改进，并且给出合理的建议。

需求评审

需求是真实的客户需求吗？

要解决的问题真实存在吗？

需求具有普遍的意义吗？

这个需求到底有多重要？

需求能不能分解、简化？

需求的最小要求是什么？

这个需求能不能在下一个版本再实现？

设计评审

能使用现存的接口吗？

设计是不是简单、直观？

一个接口是不是只表示一件事情？

接口之间的依赖关系是不是明确？
接口的调用方式是不是方便、皮实？
接口的实现可以做到不可变吗？
接口是多线程安全的吗？
可以使用异步编程吗？
接口需不需要频繁地拷贝数据？
无状态数据和有状态数据需不需要分离？
有状态数据的处理是否支持规模水平扩张？

代码评审

有没有可以重用的代码？
新的代码是不是可以重用？
有没有使用不必要的实例？
原始数据类的使用是否恰当？
集合的操作是不是多线程安全？
集合是不是可以禁止修改？
实例的尺寸还有改进的空间吗？
需要使用延迟分配方案吗？
线程同步是不是必须的？
线程同步的阻塞时间可以更短吗？
多状态同步会不会引起死锁？
是不是可以避免频繁的对象创建、销毁？
是不是可以减少内存的分配、拷贝和释放频率？
静态的集合是否会造成内存泄漏？
长时间的缓存能不能及时清理？
系统的资源能不能安全地释放？
依赖哈希值的集合，储存的对象有没有实现 hashCode() 和 equals() 方法？
hashCode() 的实现，会不会产生撞车的哈希值？
代码的清理，有没有变更代码的逻辑？

小结

编写经济的代码，是我们在编程入门之后，需要积累的一项重要技能。正是因为要考虑性能、安全等因素，编写代码才成了一个具有挑战性的工作。

如果我们有以下这两个好习惯，那么编写经济的代码的能力就会越来越强大。

第一个习惯是，要尽早地考虑性能问题。如果你最早接触的是需求制定，就从需求开始考虑；如果你最早接触的是软件架构，就从架构层面开始考虑；如果你最早接触的是软件设计，就从软件设计开始考虑；如果你最早接触到的是代码，代码也有很多性能问题可以考虑。总之，要主动、尽早地考虑效率问题。

第二个习惯是，性能的实践经验需要日积月累。性能的实践经验和技术丰富繁杂，大到产品蓝图，小到每一行代码，中间还有软件的架构、选型、部署等诸多环节，都有很多的最佳实践可以积累。而且这些最佳实践，也会随着时间的推移发生变化，比如说会出现更好的技术方案，曾经的技术满足不了新需求等。所以，我们也要随时更新我们的储备，摒弃过时的经验。

希望你根据自己的实际情况，不断修改、完善、丰富上面的清单，让这份清单更契合你自己的工作领域。

一起来动手

不同的场景，检查清单也不一定相同。我上面的清单，就没有考虑数据库和 Web 服务架构。如果让你列一个你实际工作中需要的，编写经济代码的检查清单，会是什么样子的？你可以在我上面的清单上加减检查点，或者新做一个列表。欢迎在留言区公布你的检查清单，我们一起来讨论、学习。

另外，推荐一本书《重新定义公司——谷歌是如何运营的》。如果你没有时间，看看随书附带的小册子也行。这本书，谈的虽然是公司运营，但是我们可以也从中学习到如何设计优秀的产品，如何编写优秀的代码的一些基本思想。

推荐的另外一本书是《Effective Java》。建议找找最新的版本（现在是第三版）。这本书里，有很多非常实用的小经验，每一个小经验都讲得深入又透彻。是一本 Java 程序员必备的好书。

如果你觉得这篇文章有所帮助，欢迎点击“请朋友读”，把它分享给你的朋友或者同事。



代码精进之路

你写的每一行代码都是你的名片

范学雷

Oracle 首席软件工程师
Java SE 安全组成员
OpenJDK 评审成员



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得转载



Ctrl + Enter 发表

0/2000字

提交留言

精选留言(1)



左。

老师,我再提一个问题<可能和本栏目无关> //

同一个类(类路径和包名都一模一样) 再两个jar包里面 ,然后一个项目引入这两个jar包. 执行这个项目用到这个类的时候 ,会选择那个类? 有什么优先级?

我自己测试的,

用maven项目,引入这两个jar包的时候,谁写在上面就调用谁 . 我把这个项目打成可执行文件, 也是按照maven中引入的顺序来的 . 我很奇怪 ,可执行jar里面并没有jar的加载顺序,为什么每次都调用指定的jar里面的类?

如果老师看见了 ,希望能解惑 .

//再stackoverflow上也提了一个单子,也是找不到答案

<https://stackoverflow.com/questions/55035461/how-jvm-load-different-jars-but-same-package-name-and-class-name>

□ 2019-03-11