

从语法、程序设计和架构、工具和框架、编码风格、编程思想
5个方面深入探讨编写高质量Java代码的技巧、禁忌和最佳实践



秦小波 著

Writing Solid Java Code: 151 Suggestions to Improve Your Java Program

编写高质量代码

改善Java程序的151个建议



机械工业出版社
China Machine Press



Writing Solid Java Code: 151 Suggestions to Improve Your Java Program

编写高质量代码

改善Java程序的151个建议

秦小波 著



机械工业出版社
China Machine Press

在通往“Java 技术殿堂”的路上，本书将为你指点迷津！内容全部由 Java 编码的最佳实践组成，从语法、程序设计和架构、工具和框架、编码风格和编程思想等五大方面，对 Java 程序员遇到的各种棘手的疑难问题给出了经验性的解决方案，为 Java 程序员如何编写高质量的 Java 代码提出了 151 条极为宝贵的建议。对于每一个问题，不仅以建议的方式从正反两面给出了被实践证明为十分优秀的解决方案和非常糟糕的解决方案，而且还分析了问题产生的根源，犹如醍醐灌顶，让人豁然开朗。

全书一共 12 章，第 1～3 章针对 Java 语法本身提出了 51 条建议，例如覆写变长方法时应该注意哪些事项、final 修饰的常量不要在运行期修改、匿名类的构造函数特殊在什么地方等；第 4～9 章重点针对 JDK API 的使用提出了 80 条建议，例如字符串的拼接方法该如何选择、枚举使用时有哪些注意事项、出现 NullPointerException 该如何处理、泛型的多重界限该如何使用、多线程编程如何预防死锁，等等；第 10～12 章针对程序性能、开源的工具和框架、编码风格和编程思想等方面提出了 20 条建议。

本书针对每个问题所设计的应用场景都非常典型，给出的建议也都与实践紧密结合。书中的每一条建议都可能在你的下一行代码、下一个应用或下一个项目中崭露头角，建议你将此书搁置在手边，随时查阅，一定能使你的学习和开发工作事半功倍。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

图书在版编目（CIP）数据

编写高质量代码：改善 Java 程序的 151 个建议 / 秦小波著. —北京：机械工业出版社，2011.11

ISBN 978-7-111-36259-3

I. 编… II. 秦… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2011）第 217329 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：杨绣国 陈佳媛

北京京北印刷有限公司印刷

2012 年 1 月第 1 版第 1 次印刷

186mm×240mm • 20 印张

标准书号：ISBN 978-7-111-36259-3

定价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991；88361066

购书热线：(010) 68326294；88379649；68995259

投稿热线：(010) 88379604

读者信箱：hzsj@hzbook.com



从决定撰写本书到完稿历时 9 个月，期间曾经遇到过种种困难和挫折，但这个过程让我明白了坚持的意义，明白了“行百里者半九十”的寓意——坚持下去，终于到了写前言的时刻。

为什么写这本书

从第一次敲出“Hello World”到现在已经有 15 年时间了，在这 15 年里，我当过程序员和架构师，也担任过项目经理和技术顾问——基本上与技术沾边的事情都做过。从第一次接触 Java 到现在，已经有 11 年 4 个月了，在这些年里，我对 Java 可谓是情有独钟，对其编程思想、开源产品、商业产品、趣闻轶事、风流人物等都有所了解和研究。对于 Java，我非常感激，从物质上来说，它给了我工作，帮助我养家糊口；从精神上来说，它带给我无数的喜悦、困惑、痛苦和无奈——一如我们的生活。

我不是技术高手，只是技术领域的一个拓荒者，我希望能把自己的知识和经验贡献出来，以飨读者。在写作的过程中，我也反复地思考：我为谁而写这本书？为什么要写？

希望本书能帮您少走弯路

- ☐ 您是否曾经为了提供一个“One Line”的解决方案而彻夜地查看源代码？现在您不用了。
- ☐ 您是否曾经为了理解某个算法而冥思苦想、阅览群书？现在您不用了。
- ☐ 您是否曾经为了提升 0.1 秒的性能而对 N 种实现方案进行严格测试和对比？现在您不用了。
- ☐ 您是否曾经为了避免多线程死锁问题而遍寻高手共同诊治？现在您不用了。

.....

在学习和使用 Java 的过程中您是否在原本可以很快掌握或解决的问题上耗费了大量的时间和精力？也许您现在不用了，本书的很多内容都是我用曾经付出的代价换来的，希望它能帮助您少走弯路！

希望本书能帮您打牢基础

那些所谓的架构师、设计师、项目经理、分析师们，已经有多长时间没有写过代码了？代码是一切的基石，我不太信任连“Hello World”都没有写过的架构师。看看我们软件界的先辈们吧，Dennis M. Ritchie 决定创造一门“看上去很好”的语言时，如果只是站在高处呐喊，这门语言是划时代的，它有多么优秀，但不去实现，又有何用呢？没有 Dennis M. Ritchie 的亲自编码实现，C 语言不可能诞生，UNIX 操作系统也不可能诞生。Linux 在聚拢成千上万的开源狂热者对它进行开发和扩展之前，如果没有 Linus 的编码实现，仅凭他高声呐喊“我要创造一个划时代的操作系统”，有用吗？一切的一切都是以编码实现为前提的，代码是我们前进的基石。

这是一个英雄辈出的年代，我们每个人都希望自己被顶礼膜拜，可是这需要资本和实力，而我们的实力体现了我们处理技术问题的能力：

- ☐ 你能写出简单、清晰、高效的代码？——Show it!
- ☐ 你能架构一个稳定、健壮、快捷的系统？——Do it!
- ☐ 你能回答一个困扰 N 多人的问题？——Answer it!
- ☐ 你能修复一个系统 Bug？——Fix it!
- ☐ 你非常熟悉某个开源产品？——Broadcast it!
- ☐ 你能提升系统性能？——Tune it!

.....

但是，“工欲善其事，必先利其器”，在“善其事”之前，先看看我们的“器”是否已经磨得足够锋利了，是否能够在前进的路上披荆斩棘。无论您将来的职业发展方向是架构师、设计师、分析师、管理者，还是其他职位，只要您还与软件打交道，您就有必要打好技术基础。本书对核心的 Java 编程技术进行了凝练，如果能全部理解并付诸实践，您的基础一定会更加牢固。

希望本书能帮您打造一支技术战斗力强的团队

在您的团队中是否出现过以下现象：

- ☐ 没有人愿意听一场关于编码奥秘的讲座，他们觉得这是浪费时间；
- ☐ 没有人愿意去思考和探究一个算法，他们觉得这实在是多余，Google 完全可以解决；
- ☐ 没有人愿意主动重构一段代码，他们觉得新任务已经堆积成山了，“没有坏，就不要去修它”；

- 没有人愿意格式化一下代码，即便只需要按一下【Ctrl+Shift+F】快捷键，他们觉得代码写完就完了，何必再去温习；
- 没有人愿意花时间去深究一下开源框架，他们觉得够用就好；
-

一支有实力的软件研发团队是建立在技术的基础之上的，团队成员之间需要经常地互相交流和切磋，尤其是基于可辨别、可理解的编码问题。不可否认，概念和思想也很重要，但我更看重基于代码的交流，因为代码不会说谎，比如 SOA，10 个人至少会有 5 个答案，但代码就不同了，同样的代码，结果只有一个，要么是错的，要么是对的，这才是一个技术团队应该有的氛围。本书中提出的这些问题绝大部分可能都是您的团队成员在日常的开发中会遇到的，我针对这些问题给出的建议不是唯一的解决方案，也许您的团队在讨论这一个个问题的时候能有更好的解决办法。希望通过对本书中的这些问题的争辩、讨论和实践能全面提升每一位团队成员的技术实力，从而增强整个团队的战斗力！

本书特色

深。本书不是一本语法书，它不会教您怎么编写 Java 代码，但是它会告诉您，为什么 `StringBuilder` 会比 `String` 类效率高，`HashMap` 的自增是如何实现的，为什么并行计算一般都是从 `Executors` 开始的……不仅仅告诉您 *How*（怎么做），而且还告诉您 *Why*（为什么要这样做）。

广。涉及面广，从编码规则到编程思想，从基本语法到系统框架，从 `JDK API` 到开源产品，全部都有涉猎，而且所有的建议都不是纸上谈兵，都与真实的场景相结合。

点。讲解一个知识点，而不是一个知识面，比如多线程，这里不提供多线程的解决方案，而是告诉您如何安全地停止一个线程，如何设置多线程关卡，什么时候该用 `lock`，什么时候该用 `synchronize`，等等。

精。简明扼要，直捣黄龙，一个建议就是对一个问题的解释和说明，以及提出相关的解决方案，不拖泥带水，只针对一个知识点进行讲解。

畅。本书延续了我一贯的写作风格，行云流水，娓娓道来，每次想好了一个主题后，都会先打一个腹稿，思考如何讲才能更流畅。本书不是一本很无趣的书，我一直想把它写得生动和优雅，但 `Code` 就是 `Code`，很多时候容不得深加工，最直接也就是最简洁的。

这是一本建议书，想想看，在您写代码的时候，有这样一本书籍在您的手边，告诉您如何才能编写出优雅而高效的代码，那将是一件多么惬意的事情啊！

本书面向的读者

- 寻找“*One Line*”（一行）解决方案的编码人员。

- 希望提升自己编码能力的程序员。
- 期望能够在开源世界仗剑而行的有志之士。
- 对编码痴情的人。

总之，只要还在 Java 圈子里混就有必要阅读本书，不管是程序员、测试人员、分析师、架构师，还是项目经理，都有必要。

如何阅读本书

首先声明，本书不是面向初级 Java 程序员的，在阅读本书之前至少要对基本的 Java 语法有初步了解，最好是参与过几个项目，写过一些代码，具备了这些条件，阅读本书才会有更大的收获，才会觉得是一种享受。

本书的各个章节和各个建议都是相对独立的，所以，您可以从任何章节的任何建议开始阅读。强烈建议您将它放在办公桌旁，遇到问题时随手翻阅。

本书附带有大量的源码（下载地址见华章网站 www.hzbook.com），建议大家在阅读本书时拷贝书中的示例代码，放到自己的收藏夹中，以备需要时使用。

勘误与支持

首先，我要为书中可能出现的错别字、多意句、歧义句、代码缺陷等错误向您真诚地道歉。虽然杨福川、杨绣国两位编辑和我都为此书付出了非常大的努力，但可能还是会有一些瑕疵，如果你在阅读本书时发现错误或有问题想一起讨论，请发邮件（cbf4life@126.com）给我，我会尽快给您回复。

本书的所有勘误，我都会发表在我的个人博客（<http://cbf4life.iteye.com/>）上。

致谢

首先，感谢杨福川和杨绣国两位编辑，在他们的编审下，本书才有了一个质的飞跃，没有他们的计划和安排，本书不可能出版。

其次，感谢家人的支持，为了写这本书，用尽了全部的休息时间，很少有时间陪伴父母和妻儿，甚至连吃一顿团圆饭都成了奢望，他们的大力支持让我信心满怀、干劲十足。儿子已经 6 岁了，明白骑在爸爸身上是对爸爸的折磨，也知道玩具是可以从网上买到的，“爸爸，给我买一个变形金刚……你在网上查呀……今天一定要买……”儿子在不知不觉中长大了。

再次，感谢交通银行“531”工程的所有领导和同事，是他们让我在这样超大规模的工程学习和成长，使自己的技术和技能有了长足的进步；感谢我的领导李海宁总经理和周云

康高级经理，他们时时迸发出的闪光智慧让我受益匪浅；感谢软件开发中心所有同仁对我的帮助和鼓励！

最后，感谢我的朋友王骢，他无偿地把钥匙给我，让我有一个安静的地方思考和写作，有这样的朋友，人生无憾！

当然，还要感谢您，感谢您对本书的关注。

再次对本书中可能出现的错误表示歉意，真诚地接受大家的“轰炸”！

秦小波

2011年8月于上海



目 录

前 言

第 1 章 Java 开发中通用的方法和准则 /1

- 建议 1: 不要在常量和变量中出现易混淆的字母 /2
- 建议 2: 莫让常量蜕变成变量 /2
- 建议 3: 三元操作符的类型务必一致 /3
- 建议 4: 避免带有变长参数的方法重载 /4
- 建议 5: 别让 null 值和空值威胁到变长方法 /6
- 建议 6: 覆写变长方法也循规蹈矩 /7
- 建议 7: 警惕自增的陷阱 /8
- 建议 8: 不要让旧语法困扰你 /10
- 建议 9: 少用静态导入 /11
- 建议 10: 不要在本类中覆盖静态导入的变量和方法 /13
- 建议 11: 养成良好习惯, 显式声明 UID /14
- 建议 12: 避免用序列化类在构造函数中为不变量赋值 /17
- 建议 13: 避免为 final 变量复杂赋值 /19
- 建议 14: 使用序列化类的私有方法巧妙解决部分属性持久化问题 /20
- 建议 15: break 万万不可忘 /23
- 建议 16: 易变业务使用脚本语言编写 /25

- 建议 17: 慎用动态编译 /27
- 建议 18: 避免 instanceof 非预期结果 /29
- 建议 19: 断言绝对不是鸡肋 /31
- 建议 20: 不要只替换一个类 /33

第 2 章 基本类型 /35

- 建议 21: 用偶判断, 不用奇判断 /36
- 建议 22: 用整数类型处理货币 /37
- 建议 23: 不要让类型默默转换 /38
- 建议 24: 边界, 边界, 还是边界 /39
- 建议 25: 不要让四舍五入亏了一方 /41
- 建议 26: 提防包装类型的 null 值 /43
- 建议 27: 谨慎包装类型的大小比较 /45
- 建议 28: 优先使用整型池 /46
- 建议 29: 优先选择基本类型 /48
- 建议 30: 不要随便设置随机种子 /49

第 3 章 类、对象及方法 /52

- 建议 31: 在接口中不要存在实现代码 /53
- 建议 32: 静态变量一定要先声明后赋值 /54
- 建议 33: 不要覆写静态方法 /55
- 建议 34: 构造函数尽量简化 /57
- 建议 35: 避免在构造函数中初始化其他类 /58
- 建议 36: 使用构造代码块精炼程序 /60
- 建议 37: 构造代码块会想你所想 /61
- 建议 38: 使用静态内部类提高封装性 /63
- 建议 39: 使用匿名类的构造函数 /65
- 建议 40: 匿名类的构造函数很特殊 /66
- 建议 41: 让多重继承成为现实 /68
- 建议 42: 让工具类不可实例化 /70
- 建议 43: 避免对象的浅拷贝 /71
- 建议 44: 推荐使用序列化实现对象的拷贝 /73
- 建议 45: 覆写 equals 方法时不要识别不出自己 /74

- 建议 46: equals 应该考虑 null 值情景 /76
- 建议 47: 在 equals 中使用 getClass 进行类型判断 /77
- 建议 48: 覆写 equals 方法必须覆写 hashCode 方法 /78
- 建议 49: 推荐覆写 toString 方法 /80
- 建议 50: 使用 package-info 类为包服务 /81
- 建议 51: 不要主动进行垃圾回收 /82

第 4 章 字符串 /83

- 建议 52: 推荐使用 String 直接量赋值 /84
- 建议 53: 注意方法中传递的参数要求 /85
- 建议 54: 正确使用 String、StringBuffer、StringBuilder/86
- 建议 55: 注意字符串的位置 /87
- 建议 56: 自由选择字符串拼接方法 /88
- 建议 57: 推荐在复杂字符串操作中使用正则表达式 /90
- 建议 58: 强烈建议使用 UTF 编码 /92
- 建议 59: 对字符串排序持一种宽容的心态 /94

第 5 章 数组和集合 /97

- 建议 60: 性能考虑, 数组是首选 /98
- 建议 61: 若有必要, 使用变长数组 /99
- 建议 62: 警惕数组的浅拷贝 /100
- 建议 63: 在明确的场景下, 为集合指定初始容量 /101
- 建议 64: 多种最值算法, 适时选择 /104
- 建议 65: 避开基本类型数组转换列表陷阱 /105
- 建议 66: asList 方法产生的 List 对象不可更改 /107
- 建议 67: 不同的列表选择不同的遍历方法 /108
- 建议 68: 频繁插入和删除时使用 LinkedList/112
- 建议 69: 列表相等只需关心元素数据 /115
- 建议 70: 子列表只是原列表的一个视图 /117
- 建议 71: 推荐使用 subList 处理局部列表 /119
- 建议 72: 生成子列表后不要再操作原列表 /120
- 建议 73: 使用 Comparator 进行排序 /122
- 建议 74: 不推荐使用 binarySearch 对列表进行检索 /125

建议 75: 集合中的元素必须做到 `compareTo` 和 `equals` 同步 /127

建议 76: 集合运算时使用更优雅的方式 /129

建议 77: 使用 `shuffle` 打乱列表 /131

建议 78: 减少 `HashMap` 中元素的数量 /132

建议 79: 集合中的哈希码不要重复 /135

建议 80: 多线程使用 `Vector` 或 `HashTable` /139

建议 81: 非稳定排序推荐使用 `List` /141

建议 82: 由点及面，一叶知秋——集合大家族 /143

第 6 章 枚举和注解 /145

建议 83: 推荐使用枚举定义常量 /146

建议 84: 使用构造函数协助描述枚举项 /149

建议 85: 小心 `switch` 带来的空值异常 /150

建议 86: 在 `switch` 的 `default` 代码块中增加 `AssertionError` 错误 /152

建议 87: 使用 `valueOf` 前必须进行校验 /152

建议 88: 用枚举实现工厂方法模式更简洁 /155

建议 89: 枚举项的数量限制在 64 个以内 /157

建议 90: 小心注解继承 /160

建议 91: 枚举和注解结合使用威力更大 /162

建议 92: 注意 `@Override` 不同版本的区别 /164

第 7 章 泛型和反射 /166

建议 93: Java 的泛型是类型擦除的 /167

建议 94: 不能初始化泛型参数和数组 /169

建议 95: 强制声明泛型的实际类型 /170

建议 96: 不同的场景使用不同的泛型通配符 /172

建议 97: 警惕泛型是不能协变和逆变的 /174

建议 98: 建议采用的顺序是 `List<T>`、`List<?>`、`List<Object>` /176

建议 99: 严格限定泛型类型采用多重界限 /177

建议 100: 数组的真实类型必须是泛型类型的子类型 /179

建议 101: 注意 `Class` 类的特殊性 /181

建议 102: 适时选择 `getDeclaredXXX` 和 `getXXX` /181

建议 103: 反射访问属性或方法时将 `Accessible` 设置为 `true` /182

- 建议 104: 使用 `forName` 动态加载类文件 /184
- 建议 105: 动态加载不适合数组 /186
- 建议 106: 动态代理可以使代理模式更加灵活 /188
- 建议 107: 使用反射增加装饰模式的普适性 /190
- 建议 108: 反射让模板方法模式更强大 /192
- 建议 109: 不需要太多关注反射效率 /194

第 8 章 异常 /197

- 建议 110: 提倡异常封装 /198
- 建议 111: 采用异常链传递异常 /200
- 建议 112: 受检异常尽可能转化为非受检异常 /202
- 建议 113: 不要在 `finally` 块中处理返回值 /204
- 建议 114: 不要在构造函数中抛出异常 /207
- 建议 115: 使用 `Throwable` 获得栈信息 /210
- 建议 116: 异常只为异常服务 /212
- 建议 117: 多使用异常, 把性能问题放一边 /213

第 9 章 多线程和并发 /215

- 建议 118: 不推荐覆写 `start` 方法 /216
- 建议 119: 启动线程前 `stop` 方法是不可靠的 /218
- 建议 120: 不使用 `stop` 方法停止线程 /220
- 建议 121: 线程优先级只使用三个等级 /224
- 建议 122: 使用线程异常处理器提升系统可靠性 /226
- 建议 123: `volatile` 不能保证数据同步 /228
- 建议 124: 异步运算考虑使用 `Callable` 接口 /232
- 建议 125: 优先选择线程池 /233
- 建议 126: 适时选择不同的线程池来实现 /237
- 建议 127: `Lock` 与 `synchronized` 是不一样的 /240
- 建议 128: 预防线程死锁 /245
- 建议 129: 适当设置阻塞队列长度 /250
- 建议 130: 使用 `CountDownLatch` 协调子线程 /252
- 建议 131: `CyclicBarrier` 让多线程齐步走 /254

第 10 章 性能和效率 /256

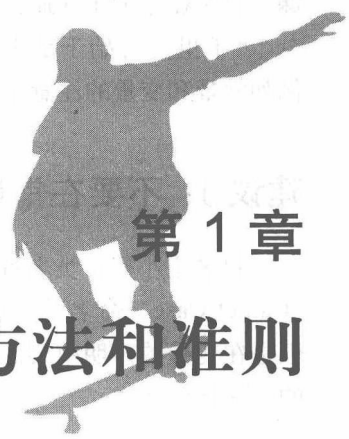
- 建议 132: 提升 Java 性能的基本方法 /257
- 建议 133: 若非必要, 不要克隆对象 /259
- 建议 134: 推荐使用“望闻问切”的方式诊断性能 /261
- 建议 135: 必须定义性能衡量标准 /263
- 建议 136: 枪打出头鸟——解决首要系统性能问题 /264
- 建议 137: 调整 JVM 参数以提升性能 /266
- 建议 138: 性能是个大“咕咚” /268

第 11 章 开源世界 /271

- 建议 139: 大胆采用开源工具 /272
- 建议 140: 推荐使用 Guava 扩展工具包 /273
- 建议 141: Apache 扩展包 /276
- 建议 142: 推荐使用 Joda 日期时间扩展包 /280
- 建议 143: 可以选择多种 Collections 扩展 /282

第 12 章 思想为源 /285

- 建议 144: 提倡良好的代码风格 /286
- 建议 145: 不要完全依靠单元测试来发现问题 /287
- 建议 146: 让注释正确、清晰、简洁 /290
- 建议 147: 让接口的职责保持单一 /294
- 建议 148: 增强类的可替换性 /295
- 建议 149: 依赖抽象而不是实现 /298
- 建议 150: 抛弃 7 条不良的编码习惯 /299
- 建议 151: 以技术员自律而不是工人 /301

A silhouette of a person in a dynamic pose, appearing to be skateboarding or performing a similar action, with one arm extended forward and the other back. The figure is positioned in the upper right quadrant of the page, partially overlapping the chapter title.

第 1 章

Java 开发中通用的方法和准则

The reasonable man adapts himself to the world;the unreasonable one persists in trying to adapt the world to himself.

明白事理的人使自己适应世界；不明事理的人想让世界适应自己。

——萧伯纳

2 ❖ 编写高质量代码：改善 Java 程序的 151 个建议

Java 的世界丰富又多彩，但同时也布满了荆棘陷阱，大家一不小心就可能跌入黑暗深渊，只有在了解了其通行规则后才能使自己在技术的海洋里遨游飞翔，恣意驰骋。

“千里之行始于足下”，本章主要讲述与 Java 语言基础有关的问题及建议的解决方案，例如常量和变量的注意事项、如何更安全地序列化、断言到底该如何使用等。

建议 1：不要在常量和变量中出现易混淆的字母

包名全小写，类名首字母全大写，常量全部大写并用下划线分隔，变量采用驼峰命名法（Camel Case）命名等，这些都是最基本的 Java 编码规范，是每个 Javaer 都应熟知的规则，但是在变量的声明中要注意不要引入容易混淆的字母。尝试阅读如下代码，思考一下打印出的 `i` 等于多少：

```
public class Client {
    public static void main(String[] args) {
        long i = 11;
        System.out.println("i 的两倍是：" + (i+i));
    }
}
```

肯定有人会说：这么简单的例子还能出错？运行结果肯定是 22！实践是检验真理的唯一标准，将其拷贝到 Eclipse 中，然后 Run 一下看看，或许你会很奇怪，结果是 2，而不是 22，难道是 Eclipse 的显示有问题，少了个“2”？

因为赋给变量 `i` 的数字就是“1”，只是后面加了长整型变量的标示字母“`L`”而已。别说是我挖坑让你跳，如果有类似程序出现在项目中，当你试图通过阅读代码来理解作者的思想时，此情此景就有可能出现。所以，为了让您的程序更容易理解，字母“`L`”（还包括大写字母“`O`”）尽量不要和数字混用，以免使阅读者的理解与程序意图产生偏差。如果字母和数字必须混合使用，字母“`L`”务必大写，字母“`O`”则增加注释。

注意 字母“`L`”作为长整型标志时务必大写。

建议 2：莫让常量蜕变成变量

常量蜕变成变量？你胡扯吧，加了 `final` 和 `static` 的常量怎么可能会变呢？不可能二次赋值的呀。真的不可能吗？看我们神奇的魔术，代码如下：

```
public class Client {
    public static void main(String[] args) {
        System.out.println("常量会变哦：" + Const.RAND_CONST);
    }
}
```

```

}
/* 接口常量 */
interface Const{
    // 这还是常量吗？
    public static final int RAND_CONST = new Random().nextInt();
}

```

RAND_CONST 是常量吗？它的值会变吗？绝对会变！这种常量的定义方式是极不可取的，常量就是常量，在编译期就必须确定其值，不应该在运行期更改，否则程序的可读性会非常差，甚至连作者自己都不能确定在运行期发生了何种神奇的事情。

甭想着使用常量会变的这个功能来实现序列号算法、随机种子生成，除非这真的是项目中的唯一方案，否则就放弃吧，常量还是当常量使用。

注意 务必让常量的值在运行期保持不变。

建议 3：三元操作符的类型务必一致

三元操作符是 if-else 的简化写法，在项目中使用它的地方很多，也非常好用，但是好用又简单的东西并不表示就可以随便用，我们来看看下面这段代码：

```

public class Client {
    public static void main(String[] args) {
        int i = 80;
        String s = String.valueOf(i<100?90:100);
        String s1 = String.valueOf(i<100?90:100.0);
        System.out.println(" 两者是否相等："+s.equals(s1));
    }
}

```

分析一下这段程序：i 是 80，那它当然小于 100，两者的返回值肯定都是 90，再转成 String 类型，其值也绝对相等，毋庸置疑的。恩，分析得有点道理，但是变量 s 中三元操作符的第二个操作数是 100，而 s1 的第二个操作数是 100.0，难道没有影响吗？不可能有影响吧，三元操作符的条件都为真了，只返回第一个值嘛，与第二个值有一毛钱的关系吗？貌似有道理。

果真如此吗？我们通过结果来验证一下，运行结果是：“两者是否相等：false”，什么？不相等，Why？

问题就出在了 100 和 100.0 这两个数字上，在变量 s 中，三元操作符中的第一个操作数（90）和第二个操作数（100）都是 int 类型，类型相同，返回的结果也就是 int 类型的 90，而变量 s1 的情况就有点不同了，第一个操作数是 90（int 类型），第二个操作数却是 100.0，而这是个浮点数，也就是说两个操作数的类型不一致，可三元操作符必须要返回一个数据，