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

實戰

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

實戰

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。

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：hzjsj@hzbook.com

前　言

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，是否能够在我们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中学习和成长，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

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

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，此情此景就有可能会出现。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，