

编写高质量代码之Java

作者：秦小波

编写高质量代码之Java

秦小波 成林 著

ISBN: 978-7-111-36259-3

ISBN: 978-7-111-39905-6

编写高质量代码：改善Java程序的151个建议纸版由机械工业出版社于2011年出版，编写高质量代码：改善JavaScript程序的188个建议纸版由机械工业出版社于2012年出版。电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

版权所有，侵权必究

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.bbbvip.com

新浪微博 @研发书局

腾讯微博 @yanfabook

[目录](#)

[编写高质量代码：改善Java程序的151个建议](#)

[前言](#)

[为什么写这本书](#)

[希望本书能帮您少走弯路](#)

[希望本书能帮您打牢基础](#)

[希望本书能帮您打造一支技术战斗力强的团队](#)

[本书特色](#)

[本书面向的读者](#)

[如何阅读本书](#)

[勘误与支持](#)

[致谢](#)

[第1章 Java开发中通用的方法和准则](#)

[建议1：不要在常量和变量中出现易混淆的字母](#)

[建议2：莫让常量蜕变成变量](#)

[建议3：三元操作符的类型务必一致](#)

[建议4：避免带有变长参数的方法重载](#)

[建议5：别让null值和空值威胁到变长方法](#)

[建议6：覆写变长方法也循规蹈矩](#)

[建议7：警惕自增的陷阱](#)

[建议8：不要让旧语法困扰你](#)

[建议9：少用静态导入](#)

[建议10：不要在本类中覆盖静态导入的变量和方法](#)

[建议11：养成良好习惯，显式声明UID](#)

[建议12：避免用序列化类在构造函数中为不变量赋值](#)

[建议13：避免为final变量复杂赋值](#)

[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](#)

[建议15：break万万不可忘](#)

[建议16：易变业务使用脚本语言编写](#)

[建议17：慎用动态编译](#)

[建议18：避免instanceof非预期结果](#)

[建议19：断言绝对不是鸡肋](#)

[建议20：不要只替换一个类](#)

[第2章 基本类型](#)

[建议21：用偶判断，不用奇判断](#)

[建议22：用整数类型处理货币](#)

[建议23：不要让类型默默转换](#)

[建议24：边界，边界，还是边界](#)

[建议25：不要让四舍五入亏了一方](#)

[建议26：提防包装类型的null值](#)

[建议27：谨慎包装类型的大小比较](#)

[建议28：优先使用整型池](#)

[建议29：优先选择基本类型](#)

[建议30：不要随便设置随机种子](#)

[第3章 类、对象及方法](#)

[建议31：在接口中不要存在实现代码](#)

[建议32：静态变量一定要先声明后赋值](#)

[建议33：不要覆写静态方法](#)

[建议34：构造函数尽量简化](#)

[建议35：避免在构造函数中初始化其他类](#)

[建议36：使用构造代码块精炼程序](#)

[建议37：构造代码块会想你所想](#)

[建议38：使用静态内部类提高封装性](#)

[建议39：使用匿名类的构造函数](#)

[建议40：匿名类的构造函数很特殊](#)

[建议41：让多重继承成为现实](#)

[建议42：让工具类不可实例化](#)

[建议43：避免对象的浅拷贝](#)

[建议44：推荐使用序列化实现对象的拷贝](#)

[建议45：覆写equals方法时不要识别不出自己](#)

[建议46：equals应该考虑null值情景](#)

[建议47：在equals中使用getClass进行类型判断](#)

[建议48：覆写equals方法必须覆写hashCode方法](#)

[建议49：推荐覆写到toString方法](#)

[建议50：使用package-info类为包服务](#)

[建议51：不要主动进行垃圾回收](#)

[第4章 字符串](#)

[建议52：推荐使用String直接量赋值](#)

[建议53：注意方法中传递的参数要求](#)

[建议54：正确使用String、StringBuilder、StringBuffer](#)

[建议55：注意字符串的位置](#)

[建议56：自由选择字符串拼接方法](#)

[建议57：推荐在复杂字符串操作中使用正则表达式](#)

[建议58：强烈建议使用UTF编码](#)

[建议59：对字符串排序持一种宽容的心态](#)

[第5章 数组和集合](#)

[建议60：性能考虑，数组是首选](#)

[建议61：若有必要，使用变长数组](#)

[建议62：警惕数组的浅拷贝](#)

[建议63：在明确的场景下，为集合指定初始容量](#)

[建议64：多种最值算法，适时选择](#)

[建议65：避开基本类型数组转换列表陷阱](#)

[建议66：asList方法产生的List对象不可更改](#)

[建议67：不同的列表选择不同的遍历方法](#)

[建议68：频繁插入和删除时使用LinkedList](#)

[建议69：列表相等只需关心元素数据](#)

[建议70：子列表只是原列表的一个视图](#)

[建议71：推荐使用subList处理局部列表](#)

[建议72：生成子列表后不要再操作原列表](#)

[建议73：使用Comparator进行排序](#)

[建议74：不推荐使用binarySearch对列表进行检索](#)

[建议75：集合中的元素必须做到compareTo和equals同步](#)

[建议76：集合运算时使用更优雅的方式](#)

[建议77：使用shuffle打乱列表](#)

[建议78：减少HashMap中元素的数量](#)

[建议79：集合中的哈希码不要重复](#)

[建议80：多线程使用Vector或HashTable](#)

[建议81：非稳定排序推荐使用List](#)
[建议82：由点及面，一叶知秋——集合大家族](#)
[第6章 枚举和注解](#)
[建议83：推荐使用枚举定义常量](#)
[建议84：使用构造函数协助描述枚举项](#)
[建议85：小心switch带来的空值异常](#)
[建议86：在switch的default代码块中增加AssertionError错误](#)
[建议87：使用valueOf前必须进行校验](#)
[建议88：用枚举实现工厂方法模式更简洁](#)
[建议89：枚举项的数量限制在64个以内](#)
[建议90：小心注解继承](#)
[建议91：枚举和注解结合使用威力更大](#)
[建议92：注意@Override不同版本的区别](#)
[第7章 泛型和反射](#)
[建议93：Java的泛型是类型擦除的](#)
[建议94：不能初始化泛型参数和数组](#)
[建议95：强制声明泛型的实际类型](#)
[建议96：不同的场景使用不同的泛型通配符](#)
[建议97：警惕泛型是不能协变和逆变的](#)
[建议98：建议采用的顺序是List<T>、List<?>、List<Object>](#)
[建议99：严格限定泛型类型采用多重界限](#)
[建议100：数组的真实类型必须是泛型类型的子类型](#)
[建议101：注意Class类的特殊性](#)
[建议102：适时选择getDeclaredxxx和getxxx](#)
[建议103：反射访问属性或方法时将Accessible设置为true](#)
[建议104：使用forName动态加载类文件](#)
[建议105：动态加载不适合数组](#)
[建议106：动态代理可以使代理模式更加灵活](#)
[建议107：使用反射增加装饰模式的普适性](#)
[建议108：反射让模板方法模式更强大](#)
[建议109：不需要太多关注反射效率](#)
[第8章 异常](#)
[建议110：提倡异常封装](#)
[建议111：采用异常链传递异常](#)
[建议112：受检异常尽可能转化为非受检异常](#)
[建议113：不要在finally块中处理返回值](#)
[建议114：不要在构造函数中抛出异常](#)
[建议115：使用Throwable获得栈信息](#)
[建议116：异常只为异常服务](#)
[建议117：多使用异常，把性能问题放一边](#)
[第9章 多线程和并发](#)
[建议118：不推荐覆写start方法](#)
[建议119：启动线程前stop方法是不可靠的](#)
[建议120：不使用stop方法停止线程](#)
[建议121：线程优先级只使用三个等级](#)
[建议122：使用线程异常处理器提升系统可靠性](#)
[建议123：volatile不能保证数据同步](#)
[建议124：异步运算考虑使用Callable接口](#)
[建议125：优先选择线程池](#)
[建议126：适时选择不同的线程池来实现](#)
[建议127：Lock与synchronized是不一样的](#)
[建议128：预防线程死锁](#)
[建议129：适当设置阻塞队列长度](#)
[建议130：使用CountDownLatch协调子线程](#)
[建议131：CyclicBarrier让多线程齐步走](#)
[第10章 性能和效率](#)
[建议132：提升Java性能的基本方法](#)
[建议133：若非必要，不要克隆对象](#)
[建议134：推荐使用“望闻问切”的方式诊断性能](#)
[建议135：必须定义性能衡量标准](#)
[建议136：枪打出头鸟——解决首要系统性能问题](#)
[建议137：调整JVM参数以提升性能](#)
[建议138：性能是个大“咕咚”](#)
[第11章 开源世界](#)
[建议139：大胆采用开源工具](#)
[建议140：推荐使用Guava扩展工具包](#)
[建议141：Apache扩展包](#)
[建议142：推荐使用Joda日期时间扩展包](#)
[建议143：可以选择多种Collections扩展](#)
[第12章 思想为源](#)
[建议144：提倡良好的代码风格](#)
[建议145：不要完全依靠单元测试来发现问题](#)
[建议146：让注释正确、清晰、简洁](#)
[建议147：让接口的职责保持单一](#)
[建议148：增强类的可替换性](#)
[建议149：依赖抽象而不是实现](#)
[建议150：抛弃7条不良的编码习惯](#)
[建议151：以技术人员自律而不是工人](#)
[编写高质量代码：改善JavaScript程序的188个建议](#)
[前言](#)
[为什么要写这本书](#)
[本书特色](#)
[读者对象](#)
[如何阅读本书](#)
[本书的期望](#)
[勘误和支持](#)
[致谢](#)
[第1章 JavaScript语言基础](#)
[建议1：警惕Unicode乱码](#)
[建议2：正确辨析JavaScript句法中的词、句和段](#)
[建议3：减少全局变量污染](#)
[建议4：注意JavaScript数据类型的特殊性](#)
[建议5：防止JavaScript自动插入分号](#)
[建议6：正确处理JavaScript特殊值](#)
[建议7：小心保留字的误用](#)
[建议8：谨慎使用运算符](#)
[建议9：不要信任hasOwnProperty](#)

[建议10: 谨记对象非空特性](#)
[建议11: 慎重使用伪数组](#)
[建议12: 避免使用with](#)
[建议13: 养成优化表达式的思维方式](#)
[建议14: 不要滥用eval](#)
[建议15: 避免使用continue](#)
[建议16: 防止switch贯穿](#)
[建议17: 块标志并非多余](#)
[建议18: 比较function语句和function表达式](#)
[建议19: 不要使用类型构造器](#)
[建议20: 不要使用new](#)
[建议21: 推荐提高循环性能的策略](#)
[建议22: 少用函数迭代](#)
[建议23: 推荐提高条件性能的策略](#)
[建议24: 优化if逻辑](#)
[建议25: 恰当选用if和switch](#)
[建议26: 小心if嵌套的思维陷阱](#)
[建议27: 小心if隐藏的Bug](#)
[建议28: 使用查表法提高条件检测的性能](#)
[建议29: 准确使用循环体](#)
[建议30: 使用递归模式](#)
[建议31: 使用迭代](#)
[建议32: 使用制表](#)
[建议33: 优化循环结构](#)
[第2章 字符串、正则表达式和数组](#)
[建议34: 字符串是非值操作](#)
[建议35: 获取字节长度](#)
[建议36: 警惕字符串连接操作](#)
[建议37: 推荐使用replace](#)
[建议38: 正确认识正则表达式工作机制](#)
[建议39: 正确理解正则表达式回溯](#)
[建议40: 正确使用正则表达式分组](#)
[建议41: 正确使用正则表达式引用](#)
[建议42: 用好正则表达式静态值](#)
[建议43: 使用exec增强正则表达式功能](#)
[建议44: 正确使用原子组](#)
[建议45: 警惕嵌套量词和回溯失控](#)
[建议46: 提高正则表达式执行效率](#)
[建议47: 避免使用正则表达式的场景](#)
[建议48: 慎用正则表达式修剪字符串](#)
[建议49: 比较数组与对象同源特性](#)
[建议50: 正确检测数组类型](#)
[建议51: 理解数组长度的有限性和无限性](#)
[建议52: 建议使用splice删除数组](#)
[建议53: 小心使用数组维度](#)
[建议54: 增强数组排序的sort功能](#)
[建议55: 不要拘泥于数字下标](#)
[建议56: 使用arguments模拟重载](#)
[第3章 函数式编程](#)
[建议57: 禁用Function构造函数](#)
[建立58: 灵活使用Arguments](#)
[建议59: 推荐动态调用函数](#)
[建议60: 比较函数调用模式](#)
[建议61: 使用闭包跨域开发](#)
[建议62: 在循环体和异步回调中慎重使用闭包](#)
[建议63: 比较函数调用和引用本质](#)
[建议64: 建议通过Function扩展类型](#)
[建议65: 比较函数的惰性求值与非惰性求值](#)
[建议66: 使用函数实现历史记录](#)
[建议67: 套用函数](#)
[建议68: 推荐使用链式语法](#)
[建议69: 使用模块化规避缺陷](#)
[建议70: 惰性实例化](#)
[建议71: 推荐分支函数](#)
[建议72: 惰性载入函数](#)
[建议73: 函数绑定有价值](#)
[建议74: 使用高阶函数](#)
[建议75: 函数柯里化](#)
[建议76: 要重视函数节流](#)
[建议77: 推荐作用域安全的构造函数](#)
[建议78: 正确理解执行上下文和作用域链](#)
[第4章 面向对象编程](#)
[建议79: 参照Object构造体系分析prototype机制](#)
[建议80: 合理使用原型](#)
[建议81: 原型域链不是作用域链](#)
[建议82: 不要直接检索对象属性值](#)
[建议83: 使用原型委托](#)
[建议84: 防止原型反射](#)
[建议85: 谨慎处理对象的Scope](#)
[建议86: 使用面向对象模拟继承](#)
[建议87: 分辨this和function调用关系](#)
[建议88: this是动态指针, 不是静态引用](#)
[建议89: 正确应用this](#)
[建议90: 预防this误用的策略](#)
[建议91: 推荐使用构造函数原型模式定义类](#)
[建议92: 不建议使用原型继承](#)
[建议93: 推荐使用类继承](#)
[建议94: 建议使用封装类继承](#)
[建议95: 慎重使用实例继承](#)
[建议96: 避免使用复制继承](#)
[建议97: 推荐使用混合继承](#)
[建议98: 比较使用JavaScript多态、重载和覆盖](#)
[建议99: 建议主动封装类](#)
[建议100: 谨慎使用类的静态成员](#)
[建议101: 比较类的构造和析构特性](#)
[建议102: 使用享元类](#)
[建议103: 使用单例类](#)

建议104: [谨慎使用伪类](#)
建议105: [比较单例的两种模式](#)
第5章 DOM编程
建议106: [建议先检测浏览器对DOM支持程度](#)
建议107: [应理清HTML DOM加载流程](#)
建议108: [谨慎访问DOM](#)
建议109: [比较innerHTML与标准DOM方法](#)
建议110: [警惕文档遍历中的空格Bug](#)
建议111: [克隆节点比创建节点更好](#)
建议112: [谨慎使用HTML集合](#)
建议113: [用局部变量访问集合元素](#)
建议114: [使用nextSibling抓取DOM](#)
建议115: [实现DOM原型继承机制](#)
建议116: [推荐使用CSS选择器](#)
建议117: [减少DOM重绘和重排次数](#)
建议118: [使用DOM树结构托管事件](#)
建议119: [使用定时器优化UI队列](#)
建议120: [使用定时器分解任务](#)
建议121: [使用定时器限时运行代码](#)
建议122: [推荐网页工人线程](#)
第6章 客户端编程
建议123: [比较IE和W3C事件流](#)
建议124: [设计鼠标拖放方案](#)
建议125: [设计鼠标指针定位方案](#)
建议126: [小心在元素内定位鼠标指针](#)
建议127: [妥善使用DOMContentLoaded事件](#)
建议128: [推荐使用beforeunload事件](#)
建议129: [自定义事件](#)
建议130: [从CSS样式表中抽取元素尺寸](#)
建议131: [慎重使用offsetWidth和offsetHeight](#)
建议132: [正确计算区域大小](#)
建议133: [谨慎计算滚动区域大小](#)
建议134: [避免计算窗口大小](#)
建议135: [正确获取绝对位置](#)
建议136: [正确获取相对位置](#)
第7章 数据交互和存储
建议137: [使用隐藏框架实现异步通信](#)
建议138: [使用iframe实现异步通信](#)
建议139: [使用script实现异步通信](#)
建议140: [正确理解JSONP异步通信协议](#)
建议141: [比较常用的服务器请求方法](#)
建议142: [比较常用的服务器发送数据方法](#)
建议143: [避免使用XML格式进行通信](#)
建议144: [推荐使用JSON格式进行通信](#)
建议145: [慎重使用HTML格式进行通信](#)
建议146: [使用自定义格式进行通信](#)
建议147: [Ajax性能向导](#)
建议148: [使用本地存储数据](#)
建议149: [警惕基于DOM的跨域侵入](#)
建议150: [优化Ajax开发的最佳实践](#)
建议151: [数据存储要考虑访问速度](#)
建议152: [使用局部变量存储数据](#)
建议153: [警惕人为改变作用域链](#)
建议154: [慎重使用动态作用域](#)
建议155: [小心闭包导致内存泄漏](#)
建议156: [灵活使用Cookie存储长信息](#)
建议157: [推荐封装Cookie应用接口](#)
第8章 JavaScript引擎与兼容性
建议158: [比较主流浏览器内核解析](#)
建议159: [推荐根据浏览器特性进行检测](#)
建议160: [关注各种引擎对ECMAScript v3的分歧](#)
建议161: [关注各种引擎对ECMAScript v3的补充](#)
建议162: [关注各种引擎对Event解析的分歧](#)
建议163: [关注各种引擎对DOM解析的分歧](#)
建议164: [关注各种引擎对CSS渲染的分歧](#)
第9章 JavaScript编程规范和应用
建议165: [不要混淆JavaScript与浏览器](#)
建议166: [掌握JavaScript预编译过程](#)
建议167: [准确分析JavaScript执行顺序](#)
建议168: [避免二次评估](#)
建议169: [建议使用直接量](#)
建议170: [不要让JavaScript引擎重复工作](#)
建议171: [使用位操作符执行逻辑运算](#)
建议172: [推荐使用原生方法](#)
建议173: [编写无阻塞JavaScript脚本](#)
建议174: [使脚本延迟执行](#)
建议175: [使用XHR脚本注入](#)
建议176: [推荐最优化非阻塞模式](#)
建议177: [避免深陷作用域访问](#)
建议178: [推荐的JavaScript性能调优](#)
建议179: [减少DOM操作中的Repaint和Reflow](#)
建议180: [提高DOM访问效率](#)
建议181: [使用setTimeout实现工作线程](#)
建议182: [使用Web Worker](#)
建议183: [避免内存泄漏](#)
建议184: [使用SVG创建动态图形](#)
建议185: [减少对象成员访问](#)
建议186: [推荐100 ms用户体验](#)
建议187: [使用接口解决JavaScript文件冲突](#)
建议188: [避免JavaScript与CSS冲突](#)

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可谓是有情有独钟，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——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Linux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能被顶礼膜拜，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，要是错的，要是对的，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（[cb@life@126.com](mailto:cb@life.iteye.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://cb@life.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

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=1l;  
  
        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，而且类型要确定，不可能条件为真时返回int类型，条件为假时返回float类型，编译器是不允许如此的，所以它就会进行类型转换了，int型转换为浮点数90.0，也就是说三元操作符的返回值是浮点数90.0，那这当然与整型的90不相等了。这里可能有读者疑惑了：为什么是整型转为浮点，而不是浮点转为整型呢？这就涉及三元操作符类型的转换规则：

若两个操作数不可转换，则不做转换，返回值为Object类型。

若两个操作数是明确类型的表达式（比如变量），则按照正常的二进制数字来转换，int类型转换为long类型，long类型转换为float类型等。

若两个操作数中有一个是数字S，另外一个表达式，且其类型标示为T，那么，若数字S在T的范围内，则转换为T类型；若S超出了T类型的范围，则T转换为S类型（可以参考“建议22”，会对该问题进行展开描述）。

若两个操作数都是直接量数字（Literal）[\[1\]](#)，则返回值类型为范围较大者。

知道是什么原因了，相应的解决办法也就有了：保证三元操作符中的两个操作数类型一致，即可减少可能错误的发生。

[\[1\]](#)"Literal"也译作“字面量”。

欢迎访问：电子书学习和下载网站（<https://www.shgis.cn>）

文档名称：《编写高质量代码之Java》秦小波 著.epub

请登录 <https://shgis.cn/post/325.html> 下载完整文档。

手机端请扫码查看：

