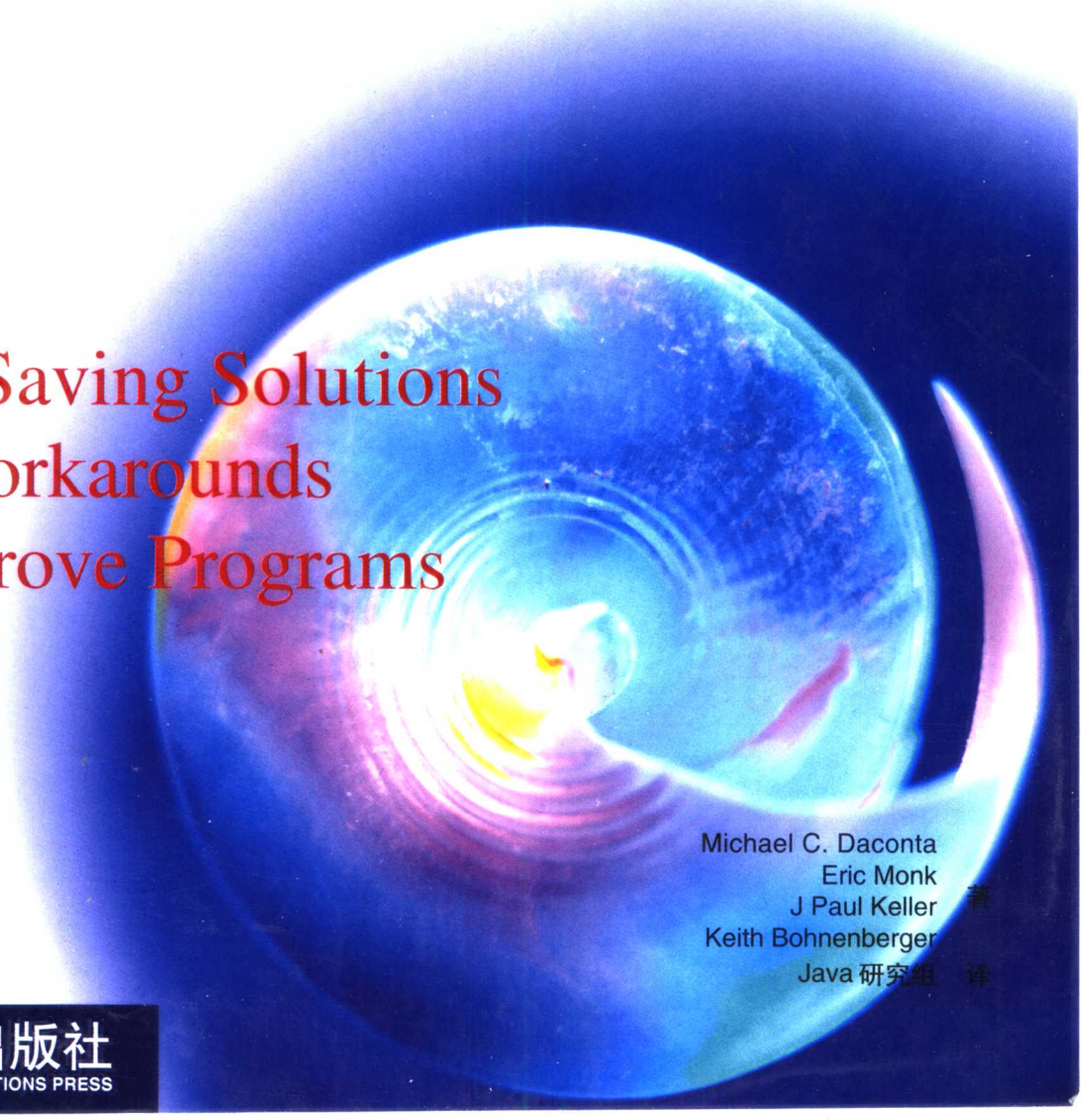
WILEY

Java Pitfalls 中文版

Time-Saving Solutions and Workarounds to Improve Programs



Michael C. Daconta
Eric Monk
J Paul Keller
Keith Bohnenberger
著
Java 研究组 译

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

Java Pitfalls 中文版

Michael C. Daconta
Eric Monk
J Paul Keller
Keith Bohnenberger
著

Java 研究组 译

人民邮电出版社

图书在版编目（CIP）数据

Java 缺陷 /（美）迪康塔（Paconta，M．C.）著；Java 研究组译.
—北京：人民邮电出版社，2003.4
书名原文：Java Pitfalls
ISBN 7-115-10882-X

I. J… II. ①迪…②J… III. JAVA 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2003）第 025038 号

Java Pitfalls 中文版

◆ 著 Michael C.Daconta Eric Monk
J Paul Keller Keith Bohnenberger
译 Java 研究组
责任编辑 陈冀康

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www. ptpress.com.cn
读者热线 010-67132705
北京汉魂图文设计有限公司制作
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000 1/16
印张：21
字数：472 千字 2003 年 4 月第 1 版
印数：1-5 000 册 2003 年 4 月北京第 1 次印刷

著作权合同登记 图字：01－2002－4864 号
ISBN 7-115-10882-X/TP・3201

定价：45.00 元

本书如有印装质量问题，请与本社联系 电话：(010)67129223

内容提要

本书旨在引导程序员绕过 Java 编程中潜在的“雷区”，帮助你避免受挫并节省时间，以从而高效地完成编程任务。

全书分为 7 大部分。作者根据多年来丰富的实践经验，以广受欢迎的条例的篇章结构，总结和阐述了 Java 编程中 50 个富有难度和具有实际意义的问题，一一用代码举证并提出了有效的解决方案。这 50 个问题涉及到 Java 语言陷阱、API 陷阱以及常见难题的最佳解决方案。

本书适合 Java 程序员和 Java 软件开发的项目管理人员阅读。

译者的话

当前 Java 已经成为一种成熟的编程语言，在计算机科学的许多方面都得到了广泛的应用。Java 图书也已汗牛充栋。在书店浏览那一排排琳琅满目的图书时，您是否会觉得自己迷失其中？但我相信，《Java Pitfalls 中文版》是那种使你眼前一亮的独特的 Java 好书。

大家都知道 Java 天生具有很多优良“基因”：跨平台、面向对象、安全机制、垃圾收集等等。所有这些都使 Java 获得了今天的“繁荣”。许多计算机巨头公司都采用 Java 语言开 发自己的产品，数以千万计的程序员在使用 Java。如果你使用 Java 已超过一年的时间，相信一定碰到过“莫名其妙”的问题，有时程序的运行结果和自己预想的完全不同，可仔细检查了所有的代码也没有发现任何错误。这也许会耗费你几天的时间，你会被它搞得烦躁、情绪低落，在一遍遍地检查后，你会丧失信心，认定肯定是出了“鬼”。也许你足够幸运，在达到你心理承受极限的最后 1 秒，找到了它。哈!原来在这儿!原来是这样!也可能，你还没有遇到过这样的情况；但我想，偶然中存在必然，我相信你早晚会遇到它。

如果遇到这种情况，你可以做两件事：

1．保持坚定的信念(**非常重要**)。你一定要使自己确信问题确实存在，并且**就在你的代码中**(有时，人会非常奇怪，几遍检查后，你会在**潜意识中**固执地认为你的程序没有问题。这将阻止你找出 Bug)。要坚信，计算机是非常诚实的。如果出现了“种瓜得豆”的情况，那么，别再犹豫了，一定是你本来就种下了豆。

2．阅读《Java Pitfalls 中文版》，也许你可以从这本书中找到答案。看看书名，你就会觉得这本书非常特别、非常实用。实际上本书是一个高效解决方案的集合，它的目的就是节约你的时间。这些解决方案都是经验丰富的 Java 程序员们在长期工作实践中的结晶。本书提出了 50 个具体的问题，并逐一进行代码描述；针对每一个问题，作者都进行了深入的讨论，并用代码阐明解决方案。这些问题都是作者和众多的 Java 程序员从实际开发过程中遇到的无数问题里精心挑选出来的，它们往往就是通向高效 Java 编程这一目标的“绊脚石”，并且它们的存在往往有一定的必然性。也就是说，在你的整个 Java 程序员生涯中，你早晚都会遇到

这些问题。读完本书之后，你就可以像专家一样，将这些问题一一化解。

你可以一页一页读完本书，但也许本书更好的用途是作为备查的工具书或你睡觉前的催眠书。你的案头有几本陪你入眠的工具书?有《Java Pitfalls 中文版》吗?如果还没有，那么，建议你添置一本吧。也许，你今天入睡前看的一页将会为你今后节约大量宝贵的时间呢。

本书由“Java 研究组”组织翻译，参加翻译的人员都是多年从事开发的 Java 程序员。他们是邓柯、李江媛、王大鲁、关承豫和吴其华；赵科及 Java 研究组的其他相关人员负责了本书的技术审校。

由于时间仓促，加之译者水平有限，书中难免有翻译不妥之处，恳请广大读者和同行批评指正。关于本书的一些学习讨论、勘误服务等等，会及时在“Java 研究组”网站（www.javaresearch.org）上发布，敬请大家关注，并与我们互动交流。

Java 研究组

2003 年 4 月

前　言

不断的研究表明，真正杰出的软件设计师能花费较少的精力，设计出速度更快、体积更小、理解更容易、结构更清晰的程序结构。出色的与平庸的程序结构之间的差距，是很难用某个尺度衡量的。

——*Frederick P. Brooks*

The Mythical Man-month

真正的高效率取决于两个因素：产品以及生产资料或生产力。

——Stephen R. Covey

The 7 Habits of Highly Effective People

高效程序的定义是：能够运用最直接、最有效的方式，精确地取得期望结果的程序；而高效编程就是贯彻在规定的时间和预算内，持续地编写出高效程序的原则。我要强调的是：高效与否并不能仅仅以最终的产品来衡量，还应评测产品的生产过程。换句话说，我们要用高效的方法来编写高效的程序。

Java：高效编程的“银弹”

几十年来，IT 经理们一直在寻找一种法宝（就像用来消灭狼人的银弹[1]），以便解决那些长期困扰软件项目，却又无法预见的“怪兽”（就像狼人）。Java 就是那枚银弹吗？虽然

[1] 在斯蒂芬·金著名的恐怖小说《狼人》中，狼人最终被一枚银弹消灭。

Frederick Brooks 以及其他大多数人都认为，任何一项单一技术都不可能成为“银弹”，但是实践证明：Java 的确能够提高程序员的效率。1998 年，国际数据集团（International Data Corporation，IDC）进行了一项调研，调查对象是 9 家从使用 C++转为使用 Java 编程的软件公司。调研结果显示，使用 Java 后，平均整个项目节约了 25%的时间，而在编码阶段更是节约了 40%的时间[1]。Java 带来的生产力的提高通常归功于它的一些显著特性：自动内存管理、庞大的跨平台包以及纯面向对象的特点。

那么，仅仅使用 Java 难道还不足以提高我们的开发效率吗？如果你是一个正在从 C++转到 Java 的程序员，那么的确如此。到目前为止，Java 已经出现 5 年多了（注：本书英文版出版于 2000 年），你还应该同其他的 Java 开发人员比较一下，看看 Java 对你们编程效率的影响有多大。此外，虽然 Java 作为一个编程平台已经取得了普遍的成功，但一些项目的失败也有它的责任。甚至用 Java 开发项目也存在许多系统性问题，例如不切实际的进度表、缺乏有经验的开发人员、过多可供选择的新技术等等。对于这些无法通过 Java 语言本身解决的问题，本书将教你如何使它们不会成为效率的障碍。

实际上本书是一个高效解决方案的集合，它的目的就是节约你的时间。这些解决方案都是经验丰富的 Java 程序员们在长期工作实践中的结晶。本书还对 Java 语言本身及其 API 中的问题进行了描述：这些问题是由存在于 Java 平台中的复杂且并不直观的因素造成的，所以处理起来要格外仔细，这样才能保证有较高的效率。最后要说明的是，本书力求简洁易懂，它对常见的问题提出了实际的解决方案，而不是令读者陷入理论的泥沼。

提高 Java 编程效率的途径

正如可以运用多种方法训练出优秀的运动员一样，培养高效的程序员也有很多途径。我们倾向于通过练习和实践达到这一目的。本书提出了 50 个具体的问题，并逐一进行代码举证。每次提出问题后，我们便深入讨论，并用代码阐明解决方案。这些问题都是我们从实际开发过程中遇到的无数问题里精心挑选出来的。它们有一个共同的特征：都是通向高效 Java 编程这一目标途中的典型“绊脚石”。我们将这些问题分为 3 类：陷阱，令人迷惑的 API，以及解决问题的最佳方案。陷阱这一类问题所占的比例最大，其次是令人迷惑的 API，最后是最佳方案。每一类问题都有其自身特点。

陷阱，指的是那些能够正常编译，但在执行时却产生了意想不到的，有时甚至是灾难性结果的程序代码。陷阱对于那些过分依赖编译检查的初级程序员来说，无疑是致命的。例如本书中的这些单元：试图在其他组件顶层绘制一个组件时产生的 Z 轴次序问题（**Item30**）；隐藏实例的数据成员（**Item6**）；Java 是强类型语言（**Item3**）；什么时候“被覆盖的”方法并非真地被覆盖了（**Item1**），声明与构造函数同名的方法（**Item4**）等。

[1] *Java Pays—Positively*，Evan Quinn 和 Chris Christiansen，IDC 1998 年 5 月公报#16212，参见 http://www.idcresearch.com/。

是什么导致 API 令人迷惑？虽然导致 API 难以理解的因素很多，但其中最主要的原因不外乎类库的体积，过多的类层次，拙劣的文档、设计和方法命名。大多数情况下，这些因素都是由于设计者错误地使用抽象，或者试图让 API 实现过多的功能造成的。例如本书中的这些单元：异常的层次结构和 OutOfMemeryError（**Item17**）；通过套接字传输序列化的对象（**Item25**）；垂直堆叠 GUI 组件（**Item32**）；怎样合理使用 GridBagLayout（**Item33**）。

最佳方案。当项目面临进度压力时（这很常见），有太多可选择的实施方案并不是件好事。在尝试过很多条开发道路（有时还会遇到死胡同）之后，我们根据经验列出了一些常见难题的最有效的解决方案。例如本书中的这些 Item：通过反射、接口和匿名类来调用方法（**Item16**）；用对象池解决大量对象的生成（**Item44**）；在循环中选择 StringBuffer 还是连接字符串的"+"（**Item47**）；用永久性缓冲机制处理大规模的集合（**Item19**）。

这些问题中有的可以归属到多个类别之中，还有一些实际上是非常有用的提示。不管如何分类，它们都有一个共同的特征：**必然性**。也就是说，在你整个的 Java 程序员生涯中，早晚都会遇到这些问题。读完本书之后，你就可以像专家一样，将这些问题一一化解。

本书的组织结构

本书分为 8 个部分，共由 50 个单元组成。各部分内容如下：

第 1 部分，语法。包括 10 个单元，教你如何正确地使用这门语言。

第 2 部分，语言支持。包括 7 个单元，讨论 java.lang 包及其子包的相关内容。

第 3 部分，工具和集合。包括 5 个描述 java.util 包的单元（如反射）。

第 4 部分，输入与输出。包括 5 个描述 java.io 包的单元。

第 5 部分，GUI 的显示。包括 8 个单元，讲述如何在屏幕上显示图形化用户界面的相关问题，涉及 AWT 和 Swing。

第 6 部分，GUI 的控制。包括 7 个有关如何控制（后台操作）图形化用户界面的单元。

第 7 部分，性能。包括 5 个有关提高 Java 程序性能的单元。

第 8 部分，杂项。包括 3 个无法归类到上述 7 个部分的单元。

欢迎参与评论

本书是程序员写给程序员的。非常感谢所有来自计算机界的评论、建议和问题。过去的一段时间，来自读者的反馈意见对提高本书的质量所起到的极大的作用，甚至超过了任何一位编辑和技术评论家。衷心感谢所有在过去和将来分享我们成果的读者。就像开放源码社区证实的那样，众人拾柴火焰高。

您可以通过电子邮件 mdaconta@aol.com 或下面的地址同我联系：

Michael Daconta

c/o Robert Elliot

John Wiley & Sons，Inc.
605 Third Avenue
New York，NY 10158

Best Wishes，

Mike Daconta
Bealeton，VA

目　录

第1部分　语法

程序员们总是被层出不穷的复杂问题所困扰……假如我们最基本的开放工具——设计和编写程序的语言本身就是复杂的，那么这个语言自己也会成为这些复杂问题的一部分，而非它们的解决方案了。

——C. A. R. Hoare, *The Emperor's Old Clothes*

Java 语言从 C++派生而来，并借鉴了 Objective C、Eiffel、Smalltalk、Mesa 和 Lisp 这些语言的一些特性。当使用其他语言的程序员转用 Java 来编程时，他们很快就会发现 Java 的一些特性和自己以前所熟悉的语言非常相似，因此这些程序员通常会认为这些特性在 Java 中和在以前所使用的语言中表现一致，其实完全不是这样。这些想法在 C++程序员中尤其普遍。这一部分重点强调了 Java 语言上经常会“绊倒”新手的陷阱和语言行为。

本部分包括以下 10 个单元：

Item 1：什么时候“被覆盖的”方法并非真的被覆盖了。本单元解释了调用子类的实例方法和静态方法之间的微妙差别。

Item 2：String.equals()方法与“==”运算符的用法比较。”本单元解释了这两种方法比较字符串的不同之处，并指出了常量池是如何混淆这两种用法的。

Item 3：Java 是强类型语言。本单元解释了基本类型的转换和提升的规则。这对从 C++ 转到 Java 的程序员尤为重要。

Item 4：那是构造函数吗？本单元给出了一个经典的，然而又非常简单的语言陷阱。当我们培训新的 Java 学员时，这个陷阱总是让学员们问出这样的问题“编译器怎么会没发现它？

Item 5：不能访问被覆盖的方法。本单元又一次讨论了 Java 语言中的方法调用，读完以后，你将完全理解这个知识点。

Item 6：避免落入“隐藏变量成员”的陷阱。本单元讨论了这一最常见的陷阱。所有 Java 语言的入门课程都应该提及这个问题，并且和 this 引用一起讨论。

Item 7：提前引用。这一较短的单元向我们演示了什么是“提前引用”，以及如何去避免它。

Item 8：设计可继承的构造函数。本单元是取自来之不易的实践经验。对于每一个想开发可重用 Java 类的程序员来说，这个单元是必读的。

Item 9：通过引用传递基本类型。本单元对从 C++转换到 Java 的程序员特别有价值。它解答了在 Java 中传递引用的相关问题。

Item 10：布尔运算符与“短路”运算符。本单元解释了 Java 编程中另一个常见的陷阱：使用逻辑运算符。单元中也举了一个使用短路（short-circuit）运算符的清晰例子。

Item 1: 什么时候“被覆盖的”方法并非真的被覆盖了

好吧，我承认，本单元的标题确实带有一定的欺骗性。虽然它的本意并非欺骗你，而是帮助你理解方法覆盖的概念。想必你已经阅读了一两本这样的 Java 书籍，它们在开头都指出了面向对象编程的 3 个主要概念：封装、继承和多态。理解这 3 个概念对于领会 Java 语言来说至关重要，而搞懂方法的覆盖又是理解继承概念的关键部分。

覆盖实例方法会在 Item 5 谈到。本单元介绍静态方法的覆盖。如果你还不明白两者的区别，那么 Item 1 和 Item 5 正适合你。假如你已经急不可待地喊出“不能覆盖静态方法！”，那么，你也许需要放松片刻再继续往下看。不过，**在此之前先看看你是否能够猜出下面例子的输出结果。**

这个例子摘自 Java 语言规范 8.4.8.5 节。

```
01:  class Super
02:  {
03:      static String greeting()
04:      {
05:         return "Goodnight";
06:      }
07:
08:      String name()
09:      {
10:         return "Richard";
11:      }
12:  }
```

```
01:  class Sub extends Super
02:  {
03:      static String greeting()
04:      {
05:         return "Hello";
06:      }
07:
08:      String name()
09:      {
10:         return "Dick";
11:      }
12:  }
```

```
01:  class Test
02:  {
03:      public static void main(String[] args)
04:      {
05:         Super s = new Sub();
06:         System.out.println(s.greeting() + ", " + s.name());
07:      }
08:  }
```

运行 Test 类的结果如下：

```
Goodnight, Dick
```

要是你得出了同样的输出结果，那么你或许对方法的覆盖有了较好的理解；如果你的结果和答案不一致，那就让我们一起找出原因。我们先分析一下各个类。Super 类由方法 greeting 和 name 组成。Sub 类继承了 Super 类，而且同样含有 greeting 和 name 方法。Test 类只有一个 main 方法。

在 Test 类的第 5 行中，我们创建了一个 Sub 类的实例。在这里，你必须明白的是：虽然变量 s 的数据类型为 Super 类，但是它仍旧是 Sub 类的一个实例。如果你对此有些迷惑，那么可以这样理解：变量 s 是一个被强制转换为 Super 型的 Sub 类的实例。下一行(第 6 行)显示了 s.greeting()返回的值，加上一个字符串“,”，紧随其后的是 s.name()的返回值。关键问题就在这里：“我们调用的到底是 Super 类的方法，还是 Sub 类的方法？”。让我们首先判断调用的是哪个类的 name()方法。两个类中的 name()方法都不是静态方法，而是实例方法。因为 Sub 类继承了 Super 类，而且有一个和它父类同样标识的 name()方法，所以 Sub 类中的 name()方法覆盖了 Super 类中的 name()方法。那么前面提到的变量 s 又是 Sub 类的一个实例，这样一来，s.name()的返回值就是“Dick”了。

至此，我们解决了问题的一半。现在我们需要判断被调用的 greeting()方法究竟是 Super 类的还是 Sub 类的。需要注意的是，两个类中的 greeting()方法都是静态方法，也称为类方法。尽管事实上 Sub 类的 greeting()方法具有相同的返回类型，相同的方法名，以及相同的方法参数（无），然而它并不覆盖 Super 类的 greeting()方法。由于变量 s 被强制转换为 Super 型，并且 Sub 类的 greeting()方法没有覆盖 Super 类的 greeting()方法，因此 s.greeting()的返回值为“Goodnight”。还是很迷惑？请记住这条规则：“实例方法被覆盖，静态方法被隐藏”。假如你就是刚才大喊“不能覆盖静态方法!”的读者之一，那么你完全正确。

现在你可能会问：“隐藏和覆盖有什么区别？”。你也许还未理解这点，然而实际上我们刚刚在这个 Super/Sub 类的例子中已经解释了两者的不同。使用类的全局名可以访问被隐藏的方法。即使变量 s 是 Sub 类的一个实例，而且 Sub 类的 greeting()方法隐藏了 Super 类的同名方法，我们仍旧能够将 s 强制转换为 Super 型，以便访问被隐藏的 greeting()方法。与被隐藏的方法不同，对被覆盖的方法而言，除了覆盖它们的类之外，其他任何类都无法访问它们。这就是为何变量 s 调用的是 Sub 类的，而非 Super 类的 name()方法。

本单元简要解释了 Java 语言中一个不时引起混淆的问题。也许对你来说，理解隐藏静态方法和覆盖实例方法的区别的最佳方式，就是自己创建几个类似于 Sub/Super 的类。再重复一次规则：实例方法被覆盖而静态方法被隐藏。被覆盖的方法只有覆盖它们的类才能访问它们；而访问被隐藏的方法的途径是提供该方法的全局名。

现在你终于明白标题里问题的答案了吧？什么时候“被覆盖的”方法并非真地被覆盖了呢？答案就是：“永远不会”。另外，我还有几个要点告诉大家，请谨记：