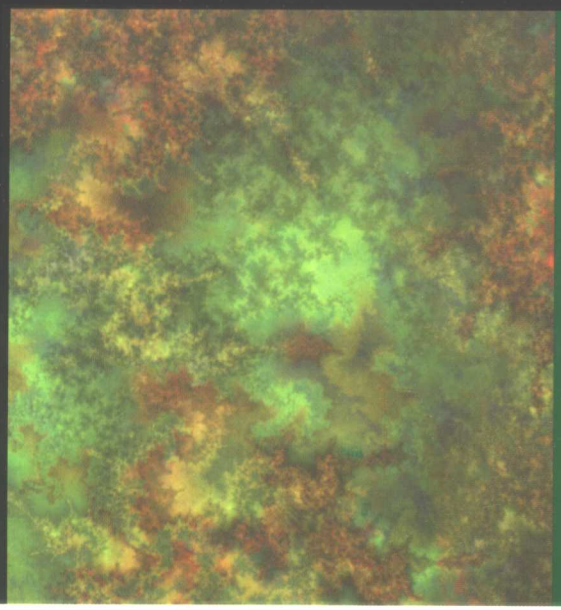

Bug Patterns
in Java

Java Bug 模式详解

(美) Eric Allen 著
王 蕾 译

清华大学出版社

Java Bug 模式详解

(美) Eric Allen 著

王 蕾 译

清华大学出版社

北 京

内 容 简 介

本书着重讨论和日常编程工作息息相关的 Java Bug 模式。Bug 模式是一种非常实用的概念，它能提高用户检测和修正代码错误的能力。

本书共分为三个部分：第 I 部分是有关 Bug 模式的理论基础，介绍 Bug 模式的基本概念及应用这种方法的意义所在；第 II 部分是对最为常见的 13 种 Bug 模式的详细讲解，从中可以学会如何识别、预防这些典型 Bug 的方法；第III部分通过表格的形式对全书内容进行了总结。

本书适合于希望通过利用 Bug 模式来提高代码质量和效率的开发工程师和编程爱好者。

EISBN：1-59059-061-9

Bug Patterns in Java

Eric Allen

Original English language edition published by Apress L. P., 2560 Ninth Street, Suite 219, Berkeley, CA 94710 USA.

北京市版权局著作权合同登记号 图字：01-2003-4570

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目(CIP)数据

Java Bug 模式详解/(美)艾伦(Eric Allen)著；王蕾译.—北京：清华大学出版社，2003

书名原文：Bug Patterns in Java

ISBN 7-302-07443-7

Ⅰ.J… Ⅱ.①艾…②王… Ⅲ.JAVA 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2003)第 094383 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969

组稿编辑：曹康
文稿编辑：王黎
封面设计：康博
版式设计：康博
印 刷 者：北京市清华园胶印厂
装 订 者：北京国马印刷厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：12.25　**字数**：254 千字
版　　次：2003 年 11 月第 1 版　2003 年 11 月第 1 次印刷
书　　号：ISBN 7-302-07443-7/TP・5499
印　　数：1～4000
定　　价：25.00 元

前　言

矛盾并不存在。在遇到矛盾的时候，请检查您的前提。您会发现前提中的某一部分必然发生了错误。

——Ayn Rand, Atlas Shrugged

近来，无论是在业界还是在学术界，安全和健壮的软件系统设计已经成为时尚。确保一个健壮的、安全的设计需要采取很多预防性措施——简单的设计方案、单元测试等。但是，就算是最有效的预防措施也不能避免 bug(故障)。

假如那样的话，快速而有效地诊断故障对软件的总成本有非常大的影响。本书介绍了一些有效诊断 bug 的措施，并引入了一系列的 bug 模式——程序中已报告的 bug 和潜在 bug 之间重复出现的相互关系。通过研究这些模式，我们对 bug 的发生可以做出快速反应。同样，我们还可以设计一些办法来防止 bug 的发生。

本书并没有把调试操作当作一个独立的行为，由一个没有参与初期开发工作的专门小组来执行。在现实世界中，大多数软件都是由编写它们的开发人员调试的。有关调试的有效方法只能在特定开发方式的上下文中讨论。读者将看到所使用的开发方法会对有效地调试软件造成很大的影响。

读者对象

本书是为任何熟悉 Java 编程技术，并希望更有效地预防、解释、交流或者修正 bug 的人员而编写的，包括业界、政府或者学术界的开发人员，以及研读计算机科学的学生。每类开发人员都具有各式各样丰富的调试技术。

对于研读计算机科学的学生来说，在调试中使用到的技术和课程本身紧密相连。一些课程注重大规模的软件开发，这些课上的学生在早期会获得很多调试经验。另一些课程则注重计算机科学的理论知识，这些课上的学生会学到很多高级的理论知识，但是他们一旦真正投入到软件项目中，就会学到很多调试技术。

在业界，很多公司试图通过雇用没有在软件开发方面受过正规教育的人，来缓解高级程序员的短缺问题，特别是那些在其他领域具有高级技术能力的程序员，这使得他们可以很快适应编程过程。但是，软件开发在许多方面和其他工程形式完全不同，一些编程技术在很长一段时间内都仍然晦涩难懂。其中一种就是如何有效地进行测试和调试。

由于程序员的背景知识有很大差别，因此本书并不局限于具有任何特殊经验的程序员。本书只是假设读者拥有使用 Java 语言的工作经验。拥有面向对象设计模式的知识

对书中一些示例的理解很有帮助(当然不是必需的)。同样，具有软件开发极限编程方法的经验也有助于加深理解，但是文中也对涉及到调试的部分极限编程方法做了简单解释。总的来说，本书可适用于具有不同知识背景的程序员。

结构

第 1～5 章将讨论有效的调试技术和有效的软件开发之间的关系，并介绍一种严格调试软件的方法。尤其是，我们将强调有效的调试技术对于大量单元测试的依赖性。

第 6～11 章侧重于 Java 编程语言中的特定 bug 模式，这些模式适用于很多上下文。不过，本书不会列出一个非常详尽的列表，只是希望开发者可以在了解这些 bug 模式的基础上建立和识别发生在更多特定上下文中的其他模式。

第 22 章将讨论应如何更好地利用传统设计模式，使软件调试工作变得更加简单。讨论将根据书中已介绍的 bug 模式展开。

尽管本书中讨论的所有 bug 模式是应用于 Java 语言的，但是其中的很多模式都可以应用于其他语言，尤其是面向对象的语言。当然，bug 模式是代码级的模式，所以在某些方面，它们必须与特定语言相关。但是，有一些模式的某些内容对于所有语言来说都是共同的，笔者鼓励程序员用其他语言对这些模式进行改写。

网络资源

本书首先介绍“Java 诊断”——IBM developerWorks Java Zone 为笔者开设的专栏——中讨论的 bug 模式概念。该专栏中的文章和其他一些有用资源可以在 Java Zone Web 站点上在线得到：http://www.ibm.com/developerworks/java

本书中的大部分示例引自 DrJava 并已经由 DrJava 授权，DrJava 是一个开放源代码的 Java 开发环境，由 GNU 通用公共许可证授权。在 Rice 大学，DrJava 是一个由笔者的博士生导师 Robert Cartwright 指导的极限编程技术项目。读者可以访问他的个人主页 SourceForge，了解关于 DrJava 的更多信息：

http://drjava.sourceforge.net

书中的所有代码示例可以在作者的网站上获得：

http://ww.cs.rice.edu/~eallen

其他网络资源请参阅第 23 章：参考资料。

目　录

第1章　混乱环境下的灵活方法

我们将讨论用于现代软件开发的上下文，并了解原有开发和调试方法的一些不足。

1.1　软件设计、实现和维护的趋势

在过去的几年里，软件的设计、实现和维护方法出现了一些新趋势，这些新趋势产生了极大的影响，如下文所示：

- 对稳定、安全系统的需求增加。
- 认识到传统软件工程技术的局限性。
- 开放源代码的软件项目的可利用性。
- 对具有独立平台语义语言的需求。

我们来仔细看看每一个问题。

1.1.1　对于稳定、安全系统的需求增加

对于稳定、安全系统的需求增长非常迅速。术语“稳定(safe)”和“安全(secure)”根据用户的不同要求，代表着不同的含义。“稳定”是指当用户犯了一个错误时，系统不会发生中断。而“安全”是指即使有人对系统蓄意攻击，系统也不会发生中断。例如，考虑一个航空公司订票系统的 Web 服务。假设有一个用户偶然试图对一个并不存在的日期(比如说，2 月 30 日)进行订票操作。如果系统是稳定的，则不会崩溃；相反，系统会显示一个错误信息并允许用户再试一次(或者更好的方法是，用户界面本身就会禁止用户输入这些错误日期)。现在我们假设一个攻击者在订票时试图窥探和再现(playback)他人的信用卡信息。如果系统是安全的，就会有适当的措施来防止这种行为的发生。当然，这两种概念并不是截然不同的。一个稳定的系统也可以简单地挫败低级的攻击者。某些糊涂的使用者可能会做出一系列安全措施所禁止的行为。在笔者早期使用 Unix 的时候，我为命令行窗口编写了一段启动脚本，偶然输入了一条用于启动另一个命令行窗口命令。当打开一个命令行窗口时，系统不停地创建其他的窗口，这样就占用了越来越多的网络资源，并导致几台服务器无法正常运行。直到系统管理员认为他的网络也受到了攻击，并切断了出现问题的账户之后，这一切才宣告结束。结果，系统管理员很长一段时间以后才允许我重新登录。

随着网络企业的不断增加，网络计算服务的普遍使用，以及嵌入式系统的发展(很多嵌入式系统是为那些具有很少或者不具有计算机经验的使用者设计的)，显而易见：对于稳定、安全系统的需求会与日俱增。

1.1.2 传统软件工程技术的局限性

人们越来越多地认识到，软件工程的传统方法——注重繁重的预先设计和对程序员角色的严格分工(例如，设计人员，编码人员，测试人员等)——通常会在使用时受到限制。随着开发小组对不同方法优缺点的更多了解，软件的设计方案会作出相应修改。这些知识只能在实际建立系统的过程中获得；预先设计的模型没有替换方案。因此，前期设计方案的很多内容将被推翻，花费在模型构造方面的努力也就浪费了。

另外，对程序员进行不同角色的分工往往会妨碍知识的交流。即使编码人员发现一个设计方案存在缺陷，也很难说服设计人员，而测试人员对代码并不熟悉，所以不能全面地测试代码。这种角色分工使得所开发的软件性能极差，而且没有用户愿意使用。另外这种软件根本就不能正常运行。

即使在理想的条件下——拥有一个合理、灵活安排的开发周期，拥有一个人员充足的开发团队，软件的设计需求也已经得到详细说明(并且再也不会修改)——传统的软件开发工程也不能很好地运作。但是，真正的软件永远也不可能在理想条件下开发。它受到各种复杂情况下的影响，例如：

- 商业现实

商业现实经常导致系统尚未经过良好的文档注释和测试就匆忙投入市场。如果产品已经延迟了两个星期才开发出来，并且除了测试以外其他工作都已完成，那么测试工作肯定就只是敷衍了事。

- 缺乏设计经验

很多新系统的开发人员都缺乏设计类似系统的经验。这并不是因为管理不善造成的，而是由于严重缺少经验丰富的软件开发人员而产生的必然结果。

- 和用户交流困难

一般来说，软件客户通常不能确定自己到底需要什么，因此不能很好的定义自己的需求。在大多数情况下，这并不代表他们是“不好的”客户，只是由于很多软件项目是初次投产的全新系统，客户没有实际经验来决定哪个特征是最有用的。

实际的软件开发是一项没有顺序可言的工作。因此，软件工程技术传统的严格过程开始让位给更灵活、更有适应性的方法，例如：极限编程技术(Extreme Programming，XP)。灵活的方法可以满足日益增长的可靠性需求。尤其是人们对单元测试的日益重视，可以大大提高软件系统的可靠性。

1.1.3　开放源代码的软件项目的可利用性

影响稳定性的另一个因素是免费的、开放源代码软件项目的日益增长。开放源代码的理念和由开放源代码组织编写的代码向传统软件公司的业务模式发出了挑战，他们可以免费提供有竞争力的产品，并且这些产品通常具有更优良的性能(由于源代码的开放性)。最著名的例子是 Linux 操作系统，它可以和现有的商业操作系统相媲美，Linux 操作系统比很多同类产品更稳定、更健壮。

1.1.4　对于跨平台语言的需求

目前，对于跨平台语言以及执行它们的虚拟机的需求越来越大。处于这一范畴中最著名的语言是 Java 语言。Java 程序员编译的二进制文件具有很强的可移植性，这种可移植性可以大大降低软件的开发成本，因为开发人员不需要维护单独的源代码，也无需为每个目标平台编译单独的二进制文件。

1.2　在快节奏的社会中学习

灵活编程的趋势和开放源代码的项目有助于满足对可靠性软件日益增长的部分需求。但是，软件的质量和可靠性必然要依赖于程序员的技术和经验。我们现有的资深程序员人数还远远不能满足需求。

提示：

使开发人员学会识别 bug 模式是一种利用众多程序员的经验提高个人工作效率的方法。

要想解决这个问题，我们需要一种办法，可以很快教会新的程序员除传统计算机科学理论知识之外的开发经验。我们需要传授开发健壮系统的实际技术，这些技术通常需要经过多年的经验积累才能获得。

这些实践经验包括各种已被证明、能够在多种上下文中成功运行的设计模式，现在，计算机科学课程除了介绍基本的算法和数据结构之外，已经开始定期讲授这些设计模式。但是，并非资深设计人员的所有宝贵经验都已包括在这些设计模式中。

这些经验知识中包括有效的诊断和修复软件系统中 bug 的能力——换句话说，就是有效的调试技术。

有效的调试技术并不是一项无关紧要的技术。事实上，跟踪和排除 bug 在软件项目

中占用了很大一部分开发时间。如果能更有效地完成这项工作，软件的开发将会更快，而且更可靠。

在培训业界和学术界的新开发人员时，笔者发现他们学习调试软件方法的一个共性。初学程序员在考虑可能产生 bug 的原因时，经常产生消极的情绪。他们往往：

(1) 埋怨底层系统(而非反对他们自己的代码)运行太快了；

(2) 说服自己所见的 bug 不可能发生(什么都不是，只是一种错觉)；

(3) 翻来覆去、漫无目的地修改代码，直到 bug 消失为止。

所有这些坏习惯将会随着经验的增加而减少。如果初学程序员随时提醒自己注意这些问题，他们在开始真正的编程以前就可以学会避免以上问题。但是，要成为一个好的调试者(这是成为灵活高效的开发人员的一部分)仅仅克服以上坏习惯是远远不够的。

1.3 bug 模式简述

正如好的编程技术涉及到很多设计模式的知识(同样可以在不同的上下文中组合应用这些模式)的知识，好的调试技术也涉及关于导致 bug 的共同原因以及如何修复 bug 的知识。在 IBM developerWorks 的专栏中，我首次将这些共同原因称为 bug 模式(bug pattern)。

bug 模式是程序中已发生的 bug 和潜在 bug 之间重复出现的相互关系。有了这些模式和 bug 现象的知识，程序员就可以很快识别新发生的 bug，还可以预防这些 bug 的发生。

bug 模式与反模式(anti-pattern)有关，反模式是指被多次证明是失败的软件通用设计模式。这些设计的反面示例是传统正面设计模式的必要补充。虽然反模式也是一种设计模式，但 bug 模式却是一种和编程错误相关的不正确的程序行为模式。这种关系与设计毫不相关，但是与编写代码和调试过程有关。

问题是程序员要想通过自己的经验逐渐学会识别这些模式可能需要很多年的时间。如果我们可以识别这些模式并将它们明确地教给程序员，就可以利用众多程序员的经验来提高个人的工作效率。

这种概念并非是编程技术中的新生技术。医生在诊断疾病时可以从重复出现这种症状的类似病例中找出治疗方案。他们在实习期间通过和资深医生共同工作来学习这些知识。他们的学习主要集中于学会如何做出诊断。

初学程序员一旦可以识别这些模式，就可以诊断 bug 的起因并很快加以纠正。此外，通过对这些相互关系的标识和交流，开发人员可以在调试中相互吸取经验，因此能更快地提高程序员的熟练程度。

1.4　小结

本章主要介绍以下内容：

- 讨论了以当前方式进行软件开发的原因。
- 研究了软件设计的变化趋势
- 了解了讲授编程技术的传统方法和新方法。
- 阐述 bug 模式对于利用现代语言进行软件开发的程序员和设计者至关重要的原因。

在第 2 章中，我们将学习如何定义 bug 的概念，解释规范对于控制软件 bug 为何至关重要的原因，了解规范和实现方案之间的区别，在开发规范时使用素材(story)和单元测试，并介绍开发规范时可以有效降低成本的方法。

第2章 Bug、规范和实现方案

本章将严格定义 bug 的概念，解释规范对控制软件 bug 至关重要的原因，突出规范和实现方案之间的不同之处，并讨论可以有效降低规范开发成本的方法。

2.1 bug 的概念

本书是一本介绍软件调试技术的书籍。为了讨论调试技术，我们有必要精确定义出 bug 是由哪些因素组成的。

从本书的角度出发，我将 bug 定义为“偏离规范的程序行为”。该定义不包括：

- 拙劣的性能，除非将性能的底限作为规范的一部分。
- 不友好或者效率低下的用户界面。尽管用户界面的设计是一个很重要的主题，却不是本书的主题。
- 缺少功能，缺乏特殊的有用功能，或缺少任何未包含在程序规范中的功能(即使该功能本将添加到规范中)。

“缺少功能”说明了 bug 定义中的一个重要方面：bug 不可避免地和程序规范联系在一起。如果没有程序规范，就没有字面意义上的 bug。当然，有一些公认的属性是人们希望所有软件都具备的，例如，不会崩溃，不会毫无结果的重复运行等。类似这些属性是任何软件规范的默认组成部分。但是有些属性则是例外；大多数程序行为都必须有明确定义。因为规范定义了行为，而行为定义了 bug，我们最好先就规范的构成进行讨论。

提示：

bug 最简单的定义就是“偏离了程序规范的程序行为”。

直观地说，程序规范是对程序行为的描述。因此，规范对于确定系统何时发生非正常操作是很重要的。我们希望规范采取何种形式呢？首先，我们要考虑传统的软件工程技术是如何回答这个问题的。

提示：

bug 和程序规范紧密相连。因为规范定义了行为，所以没有规范就不可能有所谓的 bug。