TURING 图灵程序设计丛书

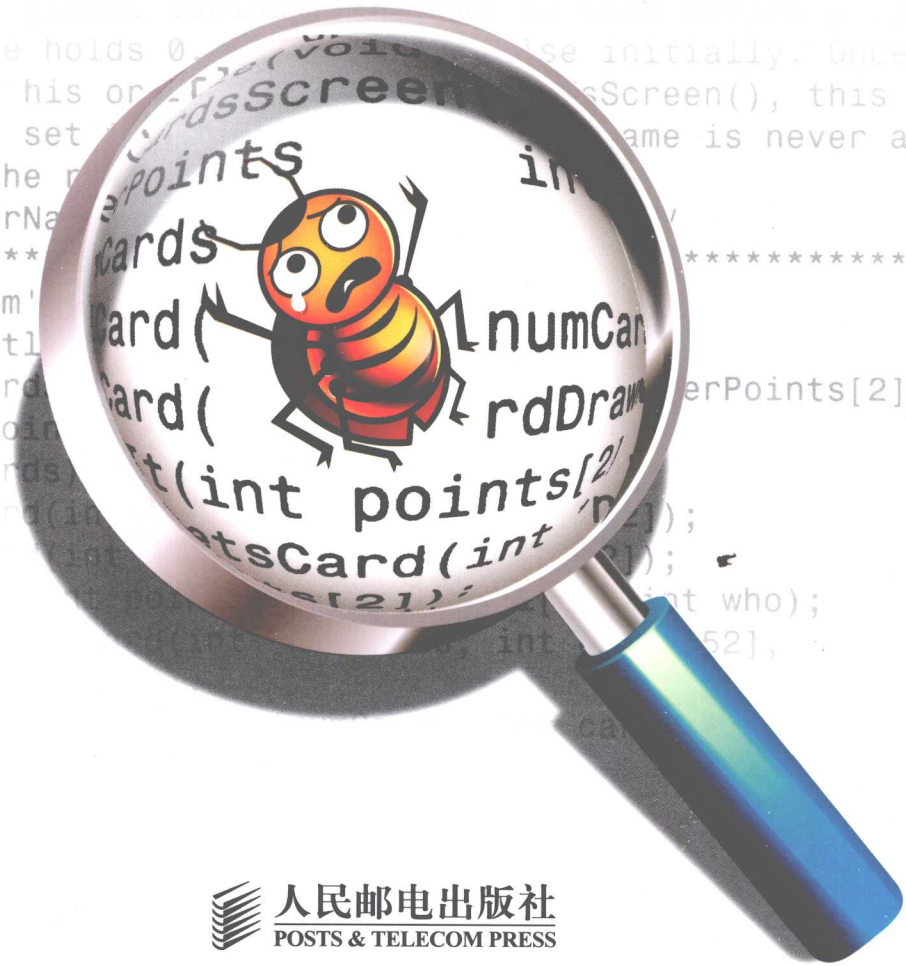
Springer

The Developer's Guide to Debugging

软件调试实战

[德] Thorsten Grötker Ulrich Holtmann Holger Keding Markus Wloka 著

赵俐 译



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

The Developer's Guide to Debugging

软件调试实战

[德] Thorsten Grötker Ulrich Holtmann Holger Keding Markus Wloka 著

赵俐 译

人民邮电出版社

北 京

图书在版编目（CIP）数据

软件调试实战 /（德）格勒特克（Grötker, T.）等著；赵俐译．—北京：人民邮电出版社，2010.2
（图灵程序设计丛书）
书名原文：The Developer's Guide to Debugging
ISBN 978-7-115-21885-8

Ⅰ.①软… Ⅱ.①格…②赵… Ⅲ.①软件－调试
Ⅳ.①TP311.5

中国版本图书馆CIP数据核字（2009）第227533号

内 容 提 要

本书主要讲述 C/C++ 程序的调试和分析，书中的调试技术也可以应用于其他语言编写的程序。本书在讲述简单的源代码分析和测试的基础上，讲述了现实的程序中经常遇到的一些问题（如程序链接、内存访问、并行处理和性能分析）并给出了解决方案。

本书适合软件开发人员、调试人员阅读和使用。

图灵程序设计丛书

软件调试实战

◆ 著 [德] Thorsten Grötker Ulrich Holtmann
Holger Keding Markus Wloka
译 赵 俐
责任编辑 傅志红
执行编辑 杨 爽

◆ 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
北京隆昌伟业印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：12.75
字数：302 千字 2010年2月第1版
印数：1－3 000册 2010年2月北京第1次印刷

著作权合同登记号 图字：01-2009-4819号
ISBN 978-7-115-21885-8

定价：45.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

译 者 序

软件并不是从一开始就完美无缺，它包含各种各样的缺陷和错误，因此需要不断调试，找到问题，然后修改代码。好的调试技巧可以帮助我们创建高质量的软件，因此在软件开发项目中，花时间进行全面而细致的调试工作是非常值得的。

软件调试早已经不是什么陌生的字眼了，大多数开发人员都或多或少做过一些调试工作，例如源代码调试、内存调试、性能调试等。然而要在实际工作中出色地完成调试工作却并不容易，因为调试需要耗费大量时间，而且具有很大的不确定性。在很多情况下，我们很难预知修复一个bug需要多长时间，甚至根本不知道能否修复它。由于这些原因，我们必须采用系统性的调试方法并配备正确的工具，这正是本书的核心。

本书主要以C和C++程序为例详解如何通过调试来分析和改进程序代码。书中揭示了大量的软件缺陷并介绍了各种调试技术，同时给出了如何编写可调试代码的建议。相信所有开发人员，特别是调试人员，都能够从本书中获益匪浅。

翻译从来都是一项充满挑战和趣味性的工作，本书的翻译更是如此，每当我字斟句酌地思考应该如何用最准确简练的语言表达几位作者思想的时候，总是能够从他们的智慧和洞察力中得到启示，从而对调试有了越发深刻的认识，这使得北京炎热的夏天也成了最美好的时光。我把这本书当成了我的老师，也希望它能够成为各位读者的老师，把几位作者的思想精华传授给你们。

最后，衷心感谢北京双飞软件公司的同仁们在翻译中给予的帮助，以及在一些高级调试主题方面的技术支持。由于译者水平有限，在翻译过程中难免会出现一些错误，恳请读者批评指正。

2009年9月

序

在所有软件开发工作中，调试或许是最令人苦恼的。调试工作极易受到指责，因为技术失败即意味着做人失败，而且矛头直指调试人员，昭示出他们所犯下的错误。由于必须反复思考每个假设，反复斟酌从需求到实现的每个步骤，因此调试将耗费大量时间。最糟糕的是，调试还无法预测，我们永远无法知道修复一个bug需要多长时间，甚至根本不知道是否能够修复它。

问一下开发人员他们生活中最沮丧的时刻，大多数回答将与调试有关。也许现在正是深夜11点，你仍在忙着调试，正当对程序进行走查时，你的家人打电话给你，问你到底什么时候才能回家，而你只希望尽快放下电话，因为好不容易得到的观察结果和推断正要从脑中溜走。此时，你可能最后有两种选择：一是重新调试，二是过后再试图重修旧好。据我个人估计，调试是导致程序员离婚的第一大原因。

然而，调试也蕴含着乐趣，就像解出难题、猜出谜语或破获谋杀案一样令人激动，但前提条件是必须采用系统性的方式并配备正确的工具。这正是本书的用武之地。本书四位作者直接与那些固持己见的开发者对话，直截了当地提出解决调试问题的建议，并给出了真正快速的解决方案。无论是解决内存问题，调试并行程序，还是处理工具链引入的问题，本书都能够提供“急救措施”，书中的建议都是经过反复尝试和验证的。

如果我最初开始调试程序时就能有这样一本书，该多好啊！我想我会屏息注视，看看这些调试工具将带给我什么惊喜，而且采纳书中的建议，必然会节省大量手工调试的时间，并可将这些时间投入到其他工作中。譬如说，可以使代码更可靠，这样最后可能根本不必做任何调试了。

当然，这是专业编程的长期目标，即从一开始就编写正确的代码，通过某种确认或验证方法来杜绝所有错误（或至少检测到错误）。目前，断言和单元测试已经在提高程序可信度方面提供了很多帮助。未来可能会有一些用于行业级系统的成熟验证方法。我们现在尚未实现这样的方法，这可能需要很多年，而且当达到这个目标时，所实现的方法肯定不会适用于现在的编程语言。在处理当今的程序，特别是那些C和C++程序时，我们仍将在调试上花费一定时间——这正是本书的宝贵价值所在。

Andreas Zeller

2008年春于萨尔兰德大学

前　言

在创作本书时，我们几位作者都就职于一家生产软件产品的技术公司，只是所属的项目或产品不同。然而，我们都曾经被派去支持客户和同行解决C和C++程序的调试问题，这是我们软件工程工作的一部分，因为我们开发了一些用来帮助用户编写优化仿真程序的工具，或者只是因为我们凑巧开发了调试工具。在这个过程中，我们一次又一次地重复着相同的基本技巧，因为在调试方面没有好的图书可供参考。

现在就不一样了。

本书的网站

我们建立了一个网站来补充本书的内容，网址为http://www.debugging-guide.com，站点上列出了软件调试主题的最新参考资料，包括工具、图书、杂志、研究论文、会议、教程和Web链接。站点还提供了本书中的示例和更多可供下载的资料。

致　谢

没有众人的帮助，本书将无法出版。

首先感谢来自Springer的Mark de Jongh先生，他总是耐心地鼓励和支持我们创作本书，他的耐心经历了我们几个人无数次的考验。

还要感谢许多人，其中包括我们在Synopsys公司的同事，感谢他们源源不断地提出软件调试领域的问题，并教给我们攻克难关的技巧。这些帮助正是本书萌芽的温床。在此，请恕我们无法完整地列出所有人员的名单。

这里要特别提一下Andrea Kroll，因为她是第一个要求我们编写用于调试仿真程序的结构化方法的人，还要特别感谢Roland Verreet给予我们的鼓励和他在营销方面的真知灼见。另外感谢Joachim Kunkel和Debashis Chowdhury提供的支持。

软件有bug，书也不例外，特别是早期的草稿。感谢那些勇敢提出意见的人们，他们使得本书的质量和可读性有了很大提高。感谢以下人员在此过程中做出的贡献（以姓氏字母排序）：Ralf Beckers、Joe Buck、Ric Hilderink、Gernot Koch、Rainer Leupers、Olaf Scheufen、Matthias Wloka和Christian Zunker。

另外，感谢Scott Meyers在如何组织各章内容以及如何呈现关键材料方面提出的建议。

还要感谢Andrea Hölter，她在反复审阅本书的过程中写下了很有见地的修改意见。

同时感谢Mike Appleby、Simon North和Ian Stephens，他们帮助我们将大量零散的信息组织为较易理解的内容，而且对我们的很多英语表述错误给予了纠正。本书剩下的任何错误和缺点当然归咎于我们自己。

最后，如果没有我们各自家庭的不懈支持，本书也将无法出版。

感谢你们！

目　录

第1章 谁编写软件，谁制造bug（为什么需要本书）

本书讲的是C和C++程序的分析和改进，是由软件开发者写给软件开发者的。

在软件开发工作中，我们经常被派去帮助客户和同行发现bug。他们都记得在学校中学过的一些概念，例如面向对象、代码评审和黑盒与白盒测试，但大部分人都不怎么了解调试工具，甚至有时会犯糊涂，例如不知道某一特定方法应该在何时使用，或者不清楚在调试工具给出混淆或错误结果时应该如何处理。

因此，我们一次又一次地发现不得不教会人们如何追踪bug。奇怪的是，很多程序员都没有将调试转化为一种系统性的方法。虽然软件开发中的很多步骤可以归纳到某一过程中，但当谈到调试时，多数人的观点都是这不仅要求对代码有深入洞察力，而且在追踪bug时还需要灵光一现。遗憾的是，理查德·费曼（Richard Feynman）的“写下问题、努力思考、写下答案”的方法并不是修复软件问题最有效和最成功的方法。

当意识到我们总是不断地写下相同的步骤规则，并为每份bug报告解释相同工具的操作和限制时，我们萌生了一个念头，应该总结所有实践经验、收集这些建议并编纂成书——结果，就是读者现在手捧的这本书。现在，在有人面对查找bug的任务时，我们就可以将本书推荐给他了。我们也相信这样一本讨论系统性调试模式的书将是编程课程的有趣补充，或者作为解决软件问题的课程的补充。原因其实很简单：

软件有bug。仅此而已。

很遗憾这么说，但事实的确如此。甚至每位C和C++程序员都熟知的`"hello, world"`程序也不例外[①]。软件开发意味着必须处理各种缺陷，包括旧缺陷、新缺陷、开发人员自己制造的缺陷以及其他人在代码中留下的缺陷。

① 如果在调用`printf()`期间，程序接收到异步信号，而又没有代码检查其返回值，那么就可能产生不完整的输出。

软件开发人员每天都要调试程序。

因此，好的调试技巧是一项必备技能。虽说如此，理工科院校中却很少开设调试课程，这不免令人遗憾。

本书既适合希望扩充自己技能的软件开发人员，也适合想要从基础开始学习一门专业技能的学生。书中提供了很多小的示例和练习，因此非常适合作为计算机科学课程的补充。同时，也可以将本书作为参考指南，随时解决出现的问题。

本书不仅仅讲述简单的源代码调试，还涵盖各个领域最常见的实际问题，包括程序链接、内存存取、并行处理和性能分析。最后几章还讨论了静态检查器和一些较好地运用了调试的代码编写技巧。

但是，本书并不能替代调试器手册，它也不是一本主要讨论Microsoft的Visual Studio或GNU的GDB的书，尽管书中频繁地提到这些工具。实际上，在可能的情况下，我们尽量介绍独立于操作系统和编译器/调试器组合的基本调试和高级调试。当然，在比较依赖编程环境的地方我们也会指出来。

大多数示例使用GCC编译器和GDB调试器。原因很简单：这些工具是免费的，而且可以在很多系统（包括UNIX、Linux、Windows和大量嵌入式平台）中使用。大多数示例可以用附录A中的表A-1来“翻译”，这张表给出的是等效的Visual Studio命令。当无法进行这种简单转换时，表A-1会提供更多的说明。

那么，阅读本书的最好方法是什么呢？这要取决于具体情况。

如果想从基础学起，从头到尾阅读本书不失为一种好方法。第2章概述了收集信息和分析问题的各种机会。第3章更详细地介绍一些关键技术，例如在调试器中运行程序、分析数据以及控制执行流。接下来，第4章介绍如何处理那些由于内存bug而莫名其妙失败的程序。接下来的两章重点关注广义上的优化。第5章讨论内存消耗。第6章解释如何分析执行速度。第7章介绍与多线程程序和异步事件相关的一些难点。接下来是第8章，介绍查找环境和编译器问题。随后的第9章介绍如何处理程序无法开始链接的情况。第9章还讨论链接程序时可能会遇到的其他问题。至此，就可以准备解决一些问题了，例如分析初始化时间问题或者调试那些在没有调试信息情况下编译的代码，这些内容将在第10章中讨论。第10章还将介绍一些其他技术，例如条件断点、观察点和捕获异步事件。最后，第11章和第12章可以帮助读者正确地编写自己的源代码。

此外，如果受到某种实际调试问题的困扰，读者可以在本书中找到相关部分来解决问题。遇到问题时翻阅一下第4章是个不错的想法，特别是当所面对的问题看起来无法用逻辑规则解决的时候。

第2章 系统性调试方法

2.1 为什么要遵循结构化的过程

理查德·费曼是个妙人，他的传奇故事读起来总是妙趣横生。他所提出的那个著名方法帮助他解决了大量问题。

诺贝尔物理学奖得主默里·盖尔曼在《纽约时报》上这样总结费曼的方法：“写下问题、努力思考、写下答案。”

这种方案不无感召力。它是一种简单的通用方法，只需要纸笔和一个善于思考的大脑。

将这种方法应用于软件调试时，几乎可以不必知道任何与系统性调试过程或工具有关的事情。但是，必须要全面了解问题。

如果问题（软件）过大，或者软件是由其他人编写的，那么这种方法就不适用了。而且它也不是一种经济的做法，因为此项目的知识无法用于彼项目，彼项目只能“重回起点”。

如果想以软件开发谋生，那么在系统性调试方法上加大投入是值得的。事实上，我们将体会到投资回报是相当可观的。

2.2 充分利用机会

本章概述bug查找过程的结构。解决各类问题的细节将在后续各章中讨论。

首先让我们在图2-1所示的简化流中找出调试的机会。

源代码（包括头文件）或多或少地都可以用一种可调试的方式来编写(1)。开发人员也可以编写额外代码，通常称为“插装”（instrumentation），来提高软件的可观察性和可控制性(2)。通常，这是通过宏定义(3) 实现的，宏定义被提供给负责处理源代码和包含头文件的预处理器。编译器标志(4)可用于生成代码和信息，这些代码和信息是源代码调试和剖析工具所必需的。

除了对编译器警告信息加以注意之外，也可以运行静态检查器工具(5)。在链接时，可以选择

带有一些调试信息的库(6)。可以使用链接器选项，比如，可以强制将更多测试例程链接到最后得到的可执行程序中(7)。也可以使用一些对可执行程序进行自动插装的工具，插装是指添加或修改用于分析性能或内存存取的对象代码(8)。

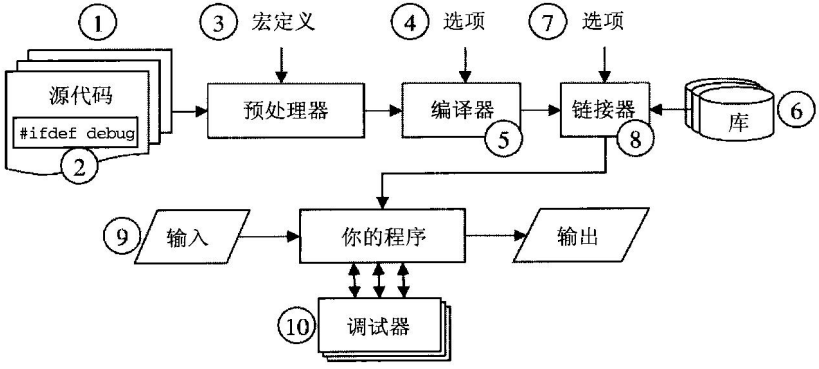


图2-1　简化的构建和测试流

有了可执行程序之后，就可以考虑如何改进它了(9)。选择好的测试用例可以大大减少分析时间。在运行时可以使用各种调试工具(10)，包括源代码调试器、剖析工具、内存检查器，以及一些可以跟踪OS调用的程序。一些工具甚至可以用于“事后剖析”，即在可执行程序已经运行（或崩溃）之后进行剖析。

现在，请花点时间考虑以下三个方面。这将有助于充分利用本书。

(1) 构建和测试过程中有调试机会，如图2-1所示。每个人用来构建和测试软件的具体流程可能略有不同，但基本元素都应该有。

(2) 有13条黄金规则通常适用于构建和测试过程中的任何阶段。这些规则将在2.3节中介绍。

(3) 后续各章将讨论可能遇到的具体问题。例如，第3章讨论源代码调试，第9章讨论链接器问题。

请注意本书是按照面向解决方案的方式组织的，既包括基本技巧，也涉及高级主题。各章的顺序并未采用图2-1所示的流程顺序。以下是各个主题对应的章。

发现bug的机会

(1) 可调试的源代码：第11章。
(2) 插装：第5、6、7、11章。
(3) 宏定义：第11章。
(4) 编译器标志：第3、6、8、9、12章。
(5) 静态检查器：第12章。

(6) 选择的库：第4、5、6、11章。
(7) 链接器选项：第9、11章。
(8) 代码插装工具：第4、5、6章。
(9) 测试用例/输入数据：第2章。
(10) 调试器。
  (a) 源代码：第3、10章。
  (b) 剖析：第5、6章。
  (c) 内存存取：第4、5章。
  (d) OS调用跟踪器（例如truss或strace）：第8章。

当然，利用哪些机会取决于具体问题。因此，我们无法定义一个通用的、一步一步的调试过程。

既然知道了到哪里去查找bug，接下来需要确定查找方式。如上文所述，我们将通过两个步骤来查找。首先，我们将在下一节列出一组“黄金规则”。这些都是指导方针，如果运用得当的话，可在各种调试情况中起到帮助作用。后续几章将讨论面向解决方案的方式中的具体问题。

2.3 13条黄金规则

经验告诉我们，不应忽视很多普遍适用的规则。“13条黄金调试规则”可以被看做是对D.J. Agans在[Agans02]中所阐述的“用于发现最难捕捉的软件和硬件问题的9条必备规则”的一个扩展。

13条黄金调试规则

(1) 理解需求。
(2) 制造失败。
(3) 简化测试用例。
(4) 读取恰当的错误消息。
(5) 检查显而易见的问题。
(6) 从解释中分离出事实。
(7) 分而治之。
(8) 工具要与bug匹配。
(9) 一次只做一项更改。
(10) 保持审计跟踪。
(11) 获得全新观点。
(12) bug不会自己修复。
(13) 用回归测试来检查bug修复。

2.3.1 理解需求

在开始调试和修复任何错误之前，一定要保证理解需求。有没有标准文档或规格说明可供查阅？有没有其他文档？或许软件根本就没有故障，只是产生了误解，而不是bug。