Broadview
www.broadview.com.cn

调软试件

Software Debugging

张银奎 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn


Software
Debugging
调试
试件

软件调试

Software Debugging

■ 张银奎 著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

围绕如何实现高效调试这一主题，本书深入系统地介绍了以调试器为核心的各种软件调试技术。本书共30章，分为6篇。第1篇介绍了软件调试的概况和简要历史。第2篇以英特尔架构（IA）的CPU为例，介绍了计算机系统的硬件核心所提供的调试支持，包括异常、断点指令、单步执行标志、分支监视、JTAG和MCE等。第3篇以Windows操作系统为例，介绍了计算机系统的软件核心中的调试设施，包括内核调试引擎、用户态调试子系统、异常处理、验证器、错误报告、事件追踪、故障转储、硬件错误处理等。第4篇以Visual C/C++编译器为例，介绍了生产软件的主要工具的调试支持，重点讨论了编译期检查、运行期检查及调试符号。第5篇讨论了软件的可调试性，探讨了如何在软件架构设计和软件开发过程中加入调试支持，使软件更容易被调试。在前5篇内容的基础上，第6篇首先介绍了调试器的发展历史、典型功能和实现方法，然后全面介绍了WinDBG调试器，包括它的模块结构、工作模型、使用方法和主要调试功能的实现细节。

本书是对软件调试技术在过去50年中所取得成就的全面展示，也是对作者本人在软件设计和系统开发第一线奋战10多年的经验总结。本书理论与实践紧密结合，选取了大量具有代表性和普遍意义的技术细节进行讨论，是学习软件调试技术的宝贵资料，适合每一位希望深刻理解软件和自由驾驭软件的人阅读，特别是从事软件开发、测试、支持的技术人员和有关的研究人员。

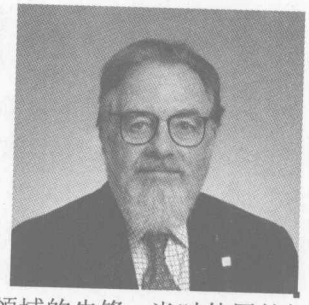
图书在版编目（CIP）数据

软件调试 / 张银奎著. —北京：电子工业出版社，2008.6
ISBN 978-7-121-06407-4
I. 软… II.张… III.软件—调试 IV.TP311.5
中国版本图书馆CIP数据核字（2008）第052852号

策划编辑：周 筠
责任编辑：陈元玉
印 刷：北京智力达印刷有限公司
装 订：三河市金马印装有限公司
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编 100036
开 本：787×1092 1/16 印张：65 字数：1 200千字
印 次：2008年6月第1次印刷
印 数：5 000册 定价：128.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。
服务热线：（010）88258888。

历史回眸



我是1949年进入麻省理工学院（MIT）的。就在那一年，第一台存储程序计算机在英国的剑桥和曼彻斯特开始运行。我的一个本科同学Kenneth Ralston是学数学的，他偶尔会和我如痴如醉地谈起一台神秘的机器，这台机器正在MIT附近的Smart街上的Barta楼内组装。我的好奇心后来在1954年的秋天得到了满足，那时我开始学习我的第一门计算机课程《数字计算机编码与逻辑》。那门课程是Charles Adams教的，他是自动编程（现在被称为编译）领域的先锋。当时使用的机器叫做旋风，被放置在一间充满了真空管电路的房间内。它由海军投资建立，用来研究飞机模拟。

因为我的知识背景及我所完成的电子工程专业的硕士课程，一个助研基金约请我在旋风计算机上用“最速下降法”解决一个最优化问题。这让我彻底熟悉了那一套繁琐的程序准备工作。我们以旋风机器的汇编语言编写程序，然后使用Friden电传打字机将以字符和数字表示的代码以打孔的方式输出到纸带上。纸带是用一个Ferrante光电读出器读入计算机的，然后交给“综合系统2”的“系统软件”进行处理。处理结果是一个二进制纸带，以大约每秒钟10行的速度打孔出来，每行代表一个6位字符。而后，用户可以调用一个简单的装载程序将二进制的纸带装入到2048字的内存中，装载程序是保存在几个可以换来换去的内存单元中的。之后就是期待程序的正常运行。用户也可以在控制台的电传打字机上调用“综合系统”的输出例程来把结果打印出来，或者把它们写到一个原始的磁带单元中，留待以后离线打印。

那时最漂亮的输出设备是CRT显示屏，用户可以在上面一个点一个点地画出图表和图片。上面装备了一个照相机，可以把显示的图片录制在胶片上。系统程序员们已经开发好了“崩溃照相”功能，可以把程序出错时内存中的内容显示在CRT上。用户可以在第二天早上取到显影后的胶片，然后用一个缩微胶卷阅读器来研究上面的八进制数字。在那时，这是调试旋风程序的最主要方法，除此之外就是把中间结果打印出来。

大多数我们这样的普通用户不知道的是，在 Barta 楼里有一个后屋，在那里第一个基于计算机的飞机跟踪和威胁检测系统上的分类工作正在进行。那儿设置了一些更先进的设备，有很多台 PPI（计划和位置标示器）显示器；并且已经开发出了第一个定点设备——光笔，用来跟计算机实时交互。

旋风计算机最初的主内存是威廉斯管型的，这还不足以满足实时操作的可靠性标准。这一需求带动了相关研究工作并导致了磁芯内存的产生。旋风工程师建造了一个非常简单的计算机，称作内存测试机（MTC），用来测试新的内存。新内存表现良好，所以被立刻安装在旋风计算机上，而后 MTC 也就功成身退了。

旋风计算机上的工作导致了 MIT 林肯实验室的形成，实验室的主要责任是基于旋风计算机上的实时系统技术开发一个国家空中防御系统。同时，林肯实验室也进行计算机技术的研究，并建立了两台使用新的晶体管技术的机器，TX-0 和 TX-2。之所以编号都是偶数，是因为奇数在英文中同时有古怪的意思，主管设计者之一 Wesley Clark 曾经说："林肯不做奇数的（古怪的）计算机"。TX-0 和 TX-2 的关系类似于 MTC 和旋风的关系：TX-0 被用作测试非常大的（按当时标准）内存，然后这些内存再被用于功能更强大的 TX-2。这些新机器继承了旋风系统中使用 CRT 显示屏和发光笔这些与用户实时交互的能力，同时也保留了使用纸带作为程序的主要介质。

在开发 TX-0 的同时，一台 IBM 704 机器在 MIT 安装起来。它用来补充、并最终接替了旋风作为 MIT 一般用户的主计算机。当 Lincoln 实验室不再需要 TX-0 后，MIT 电子工程系长期租用了它。MIT 的人们，特别是电子研究实验室，都为拥有了一台计算机而大喜过望，因为从此研究人员们便可以自由使用并亲手操作这台计算机，这要比 704 采用的批处理方式方便得多。

我于 1958 年 8 月完成了我的博士论文，成为一个四处寻找机遇的学校教员。我的新办公室在康普顿实验室楼（26 号楼）的第二层。有一天那里发生的事情引起了我的注意，人们正在一块宽广的区域安装一台 TX-0，它的位置就在 IBM 704 设施的正上方。

与 TX-0 一起到来的软件工具只有两个，一个是简单的汇编器程序，另一个是"UT-3"（3 号工具纸带）。两个程序都是二进制打孔纸带的形式，没有源代码。因为他们是以八进制代码手工输入的。UT-3 通过一个控制台打字机与用户交互（这里仍然是一个电传打字机，它包含了普通打字机的功能，可以被用户或被 TX-0 所驱动，将输入的字符传递到计算机或打印在纸上；这台打字机还带有一个机械纸带打孔器和阅读器，可以将字符打在纸带上或从纸带读字符到计算机上）。用户可以以八进制的方式输入数据到指定的内存位置，也可以要求打印指定内存位置或区域的内容。在 MIT，我们马上着手给这两个程序增加功能。汇编器最后演化为一个叫做 MACRO 的程序，除了有其他熟悉的汇编语言功能外，它还支持宏指令（宏功能是从 Doug McIlroy 在贝尔实验室的研究工作中得到启发的）。

有了汇编器后，就使得大范围重写和扩展 UT-3 成为可行。Tom Stockham 和我使新的程序支持符号，它可以使用汇编器生成的符号表。我们把这个程序称作 FLIT（电

传打字机询问纸带），这个名字仿用了当时一个很常用的杀虫喷雾剂的名字。（当 Grace Hopper 在哈佛的继电器计算机上工作时，跟踪到一次故障是继电器触点上的一只飞蛾造成的，从此人们开始把计算机的问题称作 Bug，即“臭虫”）FLIT 的最重要功能是为调试程序（“除虫”）提供了断点设施。用户可以要求 FLIT 在被测试程序中向指定的指令位置插入最多四个断点。当被测试的程序遇到一个断点时，FLIT 会通知用户，并且允许用户分析或修改内存的内容。分析结束后，用户可以要求 FLIT 恢复程序继续执行，就像没有中断过一样。FLIT 程序是后来的 DDT（另一种杀虫剂）调试程序的典范，DDT 是 MIT 的学生为 DEC 公司生产的 PDP-1 计算机开发的。

FLIT（以及 TX-0）的弱点之一是，没有办法防止被测试程序向调试程序占用的内存里存储数据，这会使调试程序停止工作。当我们给 DEC PDP-1 建立分时系统时，我们做了特别的设计，使得 DDT 与待测试的程序在各自的地址空间中执行，但 DDT 仍可以观察和改变被测试程序中的信息，我们把它称为“隐身调试器”。为了提供这种保护，需要对 PDP-1 增加一些逻辑，它们是随着为支持分时系统而作的更改和增加一起安装的。这个系统在 1963 年前后开始运行。

PDP-1 上的分时系统为 UC Berkely 在 SDS 940 上建立的分时系统提供了典范（L. Perter Deutsch 兜里装着那个小操作系统从 MIT 到了 Berkely）。我相信（虽然不很有把握），隐身调试器的机制对于 DEC PDP-11/45 的设计产生了重要影响，贝尔实验室就是为这个系统开发了 UNIX。

Jack B. Dennis

2008 年 4 月，Belmont, Massachusetts

联系博文视点

您可以通过如下方式与本书的出版方取得联系。

读者信箱：reader@broadview.com.cn

投稿信箱：bvtougao@gmail.com

北京博文视点资讯有限公司（武汉分部）

湖北省 武汉市 洪山区 吴家湾 邮科院路特 1 号 湖北信息产业科技大厦 1402 室

邮政编码：430074

电话：（027）87690813　　传真：（027）87690813 转 817

若您希望参加博文视点的有奖读者调查，或对写作和翻译感兴趣，欢迎您访问：

http://bv.csdn.net

关于本书的勘误、资源下载及博文视点的最新书讯，欢迎您访问博文视点官方博客：

http://blog.csdn.net/bvbook

序言

2007 年，我和 Raymond（张银奎）第一次在上海见面。他对 Windows 操作系统的浓厚兴趣给我留下了深刻的印象，他的兴趣遍及有关 Windows 的所有细节，包括这个产品背后的人们及其演变过程。

Raymond 已经在软件开发岗位工作了十几年。现在他把多年的经验和对 Windows 操作系统的深刻理解结合起来，创作了这本关于调试的惊世之作。调试是计算机领域中最耗费时间和充满挑战的任务之一，也是许多软件工程师都需要提高的一个领域。

这本书所覆盖主题的广度是惊人的。从最低层硬件对调试的支持落笔，Raymond 带你遍历了系统中支持调试的所有层面——从用户态到内核态。此外，他还全面深入地介绍了编译器的调试支持、调试工具和各种基础设施。

据我多年在 VMS 操作系统开发团队工作的经验，我发现有些工程师掌握调试技术，而有些工程师并不具备这样的能力。利用可用工具插入合适的断点和分析追踪信息都需要很特殊的技巧。

细细品味这本书，会帮助你获得这些重要的软件开发技巧，增强你控制软件和编写代码的能力。

我多么希望这本书是用英文写的！

David Solomon
co-author, Windows Internals (Microsoft Press)
President, David Solomon Expert Seminars, Inc.
www.solsem.com

Preface

Raymond's intense interest in the details of the Windows operating system — including the people behind the product and its evolution—impressed me from our first meeting in Shanghai in 2007.

Raymond has been working as a software developer for more than 10 years. Now he's taken his years of software development experience and intimate knowledge of the Windows operating system and created this incredible book on debugging. Debugging is one of the most time-consuming and challenging tasks in computer world. It's an area to improve for a lot of software engineers.

The breadth of topics in this book is impressive. Starting from the lowest machine level hardware support for debuggers, Raymond takes you through the entire stack of debugging support - from user to kernel mode - as well as a comprehensive look at compiler's debug support, the debugging tools and infrastructure.

In my years working in the VMS operating system development group, there were those who could debug and those that couldn't. It takes a special skill to be able to leverage the available tools to insert the appropriate breakpoints and debug trace messages.

Digesting this book will help you gain these important software development skills, strengthen your capability to control software and write code.

I only wish it was in English!

David Solomon
co-author, Windows Internals (Microsoft Press)
President, David Solomon Expert Seminars, Inc.
www.solsem.com

专家寄语

Writing software is one thing. Being sure it works as intended is another. Raymond Zhang's new book on software debugging enables us to make that next step with confidence. It will prove invaluable to software engineers.

— Professor David J. Hand, Imperial College London

Raymond Zhang has written a thorough and comprehensive guide to software debugging, perhaps the most critical step in any successful software project. He demystifies the subject, moving from essential basics to advanced techniques. This book should be part of every working programmer's library.

— G. Pascal Zachary, author of *"Showstopper: Windows NT and the Next Generation at Microsoft"*

In my experience, Raymond Zhang is an extremely accomplished individual who has most graciously provided feedback to me on several of my books, occasionally pointing out instances where I was in error (also very graciously). I'm quite sure that his monumental new book on debugging techniques will prove of great value to the engineering community.

— Tom Shanley, President of Mindshare

Indeed, a debugger is an essential tool to master if you're going to do any sort of system programming.

— Matt Pietrek, *Under the Hood columnist for MSDN Magazine*

调试程序比编写程序更像一门艺术。程序员在调试程序时，想象力的基础是各种调试技术，张银奎先生的这本书系统地介绍了各个层次上的程序调试技术，我相信每一位阅读这本书的程序员都可以丰富自己的调试知识库，从而在实践中碰到程序问题时有更丰富的想象力，快速地"逮"到程序代码中的"臭虫（Bug）"。

—潘爱民，研究员，微软亚洲研究院

感谢张银奎给 Syser Debugger 开发提供了指导性的意见。张先生这本调试巨著详细介绍了关于软件调试的方方面面，是目前为止软件调试方面的最权威著作之一。相信这本书一定能让各位读者在软件调试和开发方面受益匪浅。这本书应该成为每个软件开发人员的必备宝典。

—吴岩峰、陈俊豪，Syser 调试器设计者

调试技术是成为高素质软件开发人员必备的一项关键技术，可惜在中国技术界却没有得到应有的重视。本书秉承了 Raymond 一贯的技术传播特点与风格：循循善诱，深入底层，切中肯綮，酣畅淋漓。相信本书会成为国内调试技术领域的扛鼎之作，每一位严肃程序员之案头必备。

—李建忠，IT 技术作译者，祝成科技培训讲师

要在速成的年代里，转向对艰辛的思考和品位的召唤，并不容易。

——文怀沙

前言

现代计算机是从20世纪40年代开始出现的。当时的计算机比今天的要庞大很多，很多部件也不一样，但是有一点是完全相同的，那就是靠执行指令而工作。

一台计算机认识的所有指令被称为它的指令集（Instruction Set）。按照一定格式编写的指令序列被称为程序（Program）。在同一台计算机上，执行不同的程序，便可以完成不同的任务，因此，现代计算机在诞生之初常被冠以“通用”字样，以突出其通用性。在获得通用性带来好处的同时，人们也很快意识到了两个严峻的问题：首先是编写程序需要很多时间；其次是程序在执行时很可能出现意料之外的怪异行为。

程序对计算机的重要性和编写程序的复杂性让一些人看到了商机。大约在20世纪50年代中期，专门编写程序的公司出现了。几年后，模仿硬件（Hardware）一词，人们开始使用软件（Software）这个词来称呼计算机程序和它的文档，并把将用户需求转化为软件产品的整个过程称为软件开发（Software Development），将大规模生产软件产品的社会活动称为软件工程（Software Engineering）。

如今，几十年过去了，我们看到的是一个繁荣而庞大的软件产业。但是前面描述的两个问题依然存在：一是编写程序仍然需要很多时间；二是编写出的程序在运行时仍然会出现意料外的行为。而且后一个问题的表现形式越来越多，可能突然报告一个错误，可能给出一个看似正确却并非需要的结果，可能自作聪明地自动执行一大堆无法取消的操作，可能忽略用户的命令，可能长时间没有反应，可能直接崩溃或者永远僵死在那里……而且总是可能有无法预料的其他意外情况出现。这些“可能”大多是因为隐藏在软件中的设计失误而导致的，即所谓的软件臭虫（bug），或者称软件缺陷（defect）。

计算机是在软件指令的控制下工作的，让存在缺陷的软件控制硬件是件危险的事，可能导致惊人的损失和灾难。2003年8月14日发生的北美大停电（Northeast Blackout of 2003）使50万人受到影响，直接经济损失60亿美元，其主要原因是软件缺陷导致报警系统没有报警。1999年9月23日，美国的火星气象探测船因为没有进入预定轨道而受到大气压力和摩擦被摧毁，其原因是不同模块使用的计算单位不同，使计算出的轨道数据出现严重错误。1990年1月15日，AT&T公司的100多台交换机崩溃并反复重新启动，导致6万用户在9个小时中无法使用长途电话，其原因是新使用的软件在接收到某一种消息后会导致系统崩溃，并把这种症状传染给与它相邻的系统。1962年7月22日，水手一号太空船发射293秒后因为偏离轨道而被销毁，其原因也与软件错误有直接关系。类似的故事还有很多，尽管我们不希望它们发生。

一方面，软件缺陷难以避免；另一方面其危害又很大，这使得消除软件缺陷成为软件工程中的一项重要任务。消除软件缺陷的前提是要找到导致缺陷的根本原因。我们把探索软件缺陷的根源并寻求其解决方案的过程称为软件调试（Software Debugging）。

本书的写作目的

在复杂的计算机系统中寻找软件缺陷的根源不是一个简单的任务，需要对软件和计算机系统有深刻的理解，选用科学的方法，并使用强有力的工具。这些正是作者写作本书的初衷。具体来说，写作本书有三个主要目的。

第一，论述软件调试的一般原理，包括CPU、操作系统和编译器是如何支持软件调试的，内核态调试和用户态调试的工作模型，以及调试器的工作原理。软件调试是计算机系统中多个部件之间的一个复杂交互过程，要理解这个过程，必须要了解每个部件在其中的角色和职责，以及它们的协作方式。学习软件调试原理不仅对提高软件工程师的调试技能至关重要，而且有利于提高它们对计算机系统的理解，将计算机原理、编译原理、操作系统等多个学科的知识融会贯通在一起。

第二，探讨可调试性（Debuggability）的内涵和实现软件可调试性的原则和方法。所谓软件的可调试性就是在软件内部加入支持调试的代码，使其具有自动记录、报告和诊断的能力，从而更容易被调试。软件自身的可调试性对于提高调试效率、增强软件的可维护性，以及保证软件的如期交付都有着重要意义。

第三，交流软件调试的方法和技巧。尽管论述一般原理是本书的重点，本书仍穿插了许多实践性很强的内容。包括调试用户态程序和系统内核模块的基本方法，如何诊断系统崩溃（BSOD）和应用程序崩溃，如何调试缓冲区溢出等与栈有关的问题，如何调试内存泄漏等与堆有关的问题。特别是，本书非常全面地介绍了WinDBG调试器的使用方法，给出了大量使用这个调试器的实例。

总之，笔者希望通过本书让读者懂得软件调试的原理，意识到软件可调试性的重要性，学会使用基本的软件调试方法和调试工具，并能应用这些方法和工具解决问题和掌握更多软硬件知识。

本书的读者

首先，本书是写给所有程序员的。程序员是软件开发的核心力量。他们花大量的时间来调试他们所编写的代码，有时为此工作到深夜。笔者希望程序员朋友们读过本书后能提高调试能力，并自觉地在代码中加入调试支持，使调试效率大大提高，减少因为调试程序而加班的次数。本书中关于CPU、中断、异常和操作系统的介绍，是很多程序员需要补充的知识，因为对硬件和系统底层的深刻理解有利于写出更好的应用

程序，对于程序员的职业发展也是非常有帮助的。之所以说写给“所有”程序员是因为本书主要讨论的是一般原理和方法，没有限定某种编程语言和某个编程环境，也没有局限于某个特定的编程领域。

第二，本书是写给从事测试、验证、系统集成、客户支持、产品销售等工作的软件工程师或 IT 工程师的。他们的职责不是编写代码，因此软件缺陷与他们不直接相关，但是他们也经常受累于软件缺陷。他们不负责解决问题，但他们需要知道找谁来解决。因此，他们需要把错误定位到某个模块，或者至少定位到某个软件。本书介绍的工具和方法对于实现这个目标是非常有益的。另外，他们也可以从关于软件可调试性的内容中得到启发。本书关于 CPU、操作系统和编译器的内容对于提高他们的综合能力，巩固软硬件知识也是有益的。

第三，本书适合从事反病毒、网络安全、版权保护等工作的技术人员阅读。他们经常面对各种怪异的代码，需要在没有代码和文档的情况下做跟踪和分析。这是计算机领域中最富挑战性的工作之一。关于调试方法和 WinDBG 的内容有利于提高他们的效率。很多恶意软件故意加入了阻止调试和跟踪的机制，本书介绍的软件调试原理有助于理解这些机制。

第四，本书也适合计算机、软件、自动控制、电子学等专业的研究生或高年级本科生来研读。他们已经学习了程序设计、操作系统、计算机原理等课程，阅读本书可以帮助他们把这些知识联系起来，并深入到一个新的层次。学会使用调试器来跟踪和分析软件，可以让他们在指令一级领悟计算机软硬件的工作方式，深入核心，掌握本质，把学到的书本知识与计算机系统的实际情况结合起来；同时，可以提高他们的自学能力，使他们养成乐于专研和探索的良好习惯。软件调试是从事计算机软硬件开发等工作的一项基本功，在学校里就掌握了这门技术，对于以后快速适应工作岗位是大有好处的。

第五，本书是写给勇于挑战软件问题的硬件工程师和计算机用户的。他们是软件缺陷的受害者。除了要忍受软件缺陷带来的不便之外，有时软件设计者还可能将责任推卸给他们，推诿是硬件问题或使用不当。使用本书的工具和方法，他们可以找到充足的证据来证明这是软件的问题。本书的大多数内容不需要很深厚的软件背景，有基本的计算机知识就可以读懂。

最后，或许还有不属于上面 5 种类型的读者也可能会阅读本书。比如，软件公司或软件团队的管理者、软件方面的咨询师和培训师、大学和研究机构的研究人员、非计算机专业的学生、自由职业者、编程爱好者、黑客等等。

前面说过，本书的大多数内容不需要深厚的软件开发背景，但如果读者具备以下基础，将更容易读懂和领会本书的内容：

- 曾经亲自参与编写程序，包括输入代码、编译，然后执行。

- 使用过某一种类型的调试器，用过断点、跟踪、观察变量等基本调试功能。
- 参加过某个软件开发项目，对软件工程有基本的了解。认同软件的复杂性，即开发一个软件产品与写一个 HelloWorld 程序根本不是一回事，

尽管本书给出了一些汇编代码和 C/C++代码，但是其目的只是在代码层次直截了当地阐述问题。本书的目标不是讨论编程语言和编程技巧，也不要求读者已经具备丰富的编程经验。

本书的主要内容

本书共有 30 章，分为以下 6 篇。

第 1 篇：绪论（第 1 章）

作为全书的开篇，这一部分介绍了软件调试的概念、基本过程、分类和简要历史，并综述了本书后面将详细介绍的主要调试技术。

第 2 篇：CPU 的调试支持（第 2～7 章）

CPU 是计算机系统的硬件核心。这一部分以 IA-32 CPU 为例，系统描述了 CPU 的调试支持，包括如何支持软件断点、硬件断点和单步调试（第 4 章），如何支持硬件调试器（第 7 章），记录分支、中断、异常和支持性能分析的方法（第 5 章），以及支持硬件可调试性的错误检查和报告机制——MCA（机器检查架构）（第 6 章）。为了帮助读者理解这些内容，以及本书后面的章节，第 2 章介绍了关于 CPU 的一些基础知识，包括指令集、寄存器和保护模式，第 3 章深入介绍了与软件调试关系密切的中断和异常机制。

第 3 篇：操作系统的调试支持（第 8～19 章）

操作系统（OS）是计算机系统的管理者和软件核心，也是应用软件运行的基础。第 8 章介绍了 Windows 操作系统的基本知识，包括架构、关键模块和系统进程等。然后以 Windows 操作系统为例，描述了操作系统的调试支持，包括如何支持应用程序调试（第 9 章和第 10 章），如何支持调试系统内核和驱动程序（第 18 章），以及支持可调试性的错误提示机制（第 13 章）、错误报告机制——WER（第 14 章）、错误记录机制（第 15 章）、事件追踪机制——ETW（第 16 章）、硬件错误处理机制——WHEA（第 17 章）。第 19 章介绍了提高测试和调试效率的程序验证（Verifier）机制和有关工具。第 11 章介绍了中断和异常的分发与管理。第 12 章介绍了未处理异常和 JIT 调试。

第 4 篇：编译器的调试支持（第 20～25 章）

编译器是软件生产的主要工具，它帮助我们将程序语言翻译为可以被 CPU 所理解的机器码。支持软件调试始终是编译器的一个设计目标。在编译过程中，编译器会帮我们检查程序中的静态错误（编译期检查）（第 20 章）。为了帮助发现只有在运行时才体现出来的问题，编译器可以在程序中插入代码并报告运行时的可疑情况（运行期检

查）（第 21 章）。很多软件缺陷是与局部变量、缓冲区和内存使用有关的，对此，编译器设计了很多种检查和保护栈（第 22 章）及堆（第 23 章）的机制。编译器对软件调试的另一个重大支持就是调试符号。调试符号是软件调试时的灯塔，是观察数据结构和进行源代码级调试所必需的。第 25 章详细介绍了调试符号的产生过程、种类、文件格式和用法。第 24 章介绍了异常处理代码是如何编译的。在介绍以上内容时，本篇还覆盖了有关函数调用规范，栈的布局，以及堆的内部结构等与软件调试密切相关的基础内容。

第 5 篇：可调试性（第 26～27 章）

提高软件调试效率是一项系统工程，除了 CPU、操作系统和编译器所提供的调试支持外，被调试软件本身的可调试性也是至关重要的。这一篇，我们先介绍了提高软件可调试性的意义、基本原则、实例和需要注意的问题（第 26 章）。然后讨论了如何在软件开发实践中实现可调试性（第 27 章），包括软件团队中各个角色应该承担的职责，实现可追溯性、可观察性和自动报告的方法。

第 6 篇：调试器（第 28～30 章）

调试器（Debugger）是软件调试的核心工具。借助调试器，我们可以将软件冻结（中断）在我们指定的位置，然后观察它的内部状态、了解它的运行轨迹和即将执行的操作。根据需要，我们可以分析它的任一条指令，查看它使用的任一个内存单元。分析后，我们可以让它从原来的地方恢复执行，也可以让它“飞”到一个新的地方继续执行，或者干脆将其终止。这一部分分为 3 章。第 28 章介绍了调试器的历史、主要功能、分类方法、实现模型、架构和一个公开标准——HPD（High Performance Debugger）。第 29 章分析了 WinDBG 调试器的架构和主要功能的实现原理。第 30 章分为 18 个主题，系统介绍了 WinDBG 调试器的使用方法。

除了以上 6 篇，附录 A 列出了与本书配套的工具和源程序，附录 B 列出了 WinDBG 的标准命令。

本书的三条线索

本书的内容是根据以下三条线索来组织的。

第一条线索是软件调试的“生态”系统（ecosystem）。我们介绍了这个系统中的所有“成员”，描述了每个成员的职责以及成员间的协作方式。CPU（第 2 篇）为关键的调试功能提供了硬件级的支持；操作系统（第 3 篇）把 CPU 的支持进行必要的封装，并构建一整套软件调试所需的基础设施，然后以 API 的形式提供给调试器和应用软件；编译器（第 4 篇）负责产生更易于调试的调试版本，以及包含调试信息的调试符号文件；被调试软件（第 5 篇）则应该努力提高自身的可调试性；调试器（第 6 篇）负责把所有“成员”的努力化为成果，以简单方便的形式呈现给用户（调试者），让他们利