本书为加速程序开发进程提供了大量实用的技术和方法。它非常简明扼要，必将会给你的时间和金钱投资以超值的回报。

——Mark Russinovich，Winternals公司首席软件设计师

Windows程序调试

Debugging Windows Programs

Everett N. McKay Mike Woodring 著

何健辉 许俊娟 董伟 译

Strategies, Tools, and Techniques for Visual C++ Programmers

Addison-Wesley

中国电力出版社
www.infopower.com.cn

Windows程序调试

Debugging Windows Programs

Everett N. McKay　Mike Woodring　著

何健辉　许俊娟　董伟　译

Strategies, Tools, and Techniques for
Visual C++ Programmers

中国电力出版社

内 容 提 要

调试 Windows 程序是一件繁琐而又复杂的事情，掌握必要的调试策略却可以使这些工作变得轻松起来。本书精选了进行 Windows 调试所需的基础调试技巧，共分为三个大部分。第一部分介绍调试策略，帮助理解调试过程，以及如何使用 C++语言、断言、跟踪和异常来预防、揭示、诊断和消除错误。第二部分介绍 C++和 Windows 中的调试工具。第三部分介绍调试技术，帮助读者充分利用 Visual C++的调试工具，并特别论述了与调试内存相关的问题、多线程问题和 COM 问题。本书将重点放在调试概念上，而不是简单地介绍工具，因此具有很强的实用性，是您在程序开发时的最佳选择。

本书适合有一定 Visual C++基础知识的程序开发员及计算机爱好者阅读。

图书在版编目（CIP）数据

Windows 程序调试/（美）麦凯，（美）伍德林 编著；何健辉 等译. —北京：中国电力出版社，2002.3

ISBN 7-5083-0942-1

Ⅰ.W… Ⅱ.①麦…②伍…③何… Ⅲ. 窗口软件，Windows－基本知识 Ⅳ. TP316.7

中国版本图书馆 CIP 数据核字（2002）第 007852 号

著作权合同登记号 图字：01-2001-2337 号

本书英文版原名：Debugging Windows Programs

Published by arrangement with Addison Wesley Longman, Inc.

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

北京市地矿印刷厂印刷

各地新华书店经售

*

2002 年 5 月第一版 2002 年 5 月北京第一次印刷

787 毫米×1092 毫米 16 开本 28 印张 628 千字

定价 **49.00** 元

前　言

调试 Windows 程序是一项浩大、复杂的工程。一些介绍有关 Windows 调试的所有方面的书，动辄就能达到这本书的两倍厚。但是全面介绍的书籍有一个缺点，就是它太厚了，令人望之生畏，恐怕很少有人想读它。所以，这本书中，我们圈定了一个范围，将重点放在 Windows 调试的某些方面。下面介绍这个范围是如何挑选的。

本书最基本的动机建立在这样的信念上：如果程序员能更好地获取调试知识，他们就可以提高调试技能。虽然有大量调试知识，但它们目前仍然非常零散，不利于程序员阅读并掌握。这些知识常常是含糊的，不完整的，要么就是把重点放在调试工具而不是调试概念上。最终的调试工具永远是程序员的头脑，但这个工具常常被忽略了。掌握 Windows 调试的基础概念，有助于预防错误，同时也可以有效地发现残留的错误。

本书精选了进行 Windows 调试所需的基础调试技巧。读完每一章，读者都应该将该章所介绍的主题与已有的知识联系起来。本书第一部分介绍了调试策略以帮助理解调试过程，以及如何使用 C++语言、断言、跟踪和异常来预防、揭示、诊断和消除错误。第二部分介绍了 C++和 Windows 中的调试工具。第三部分介绍了调试技术，帮助读者充分利用 Visual C++的调试工具，并特别论述了与调试内存相关的问题、多线程问题和 COM 问题。

这里介绍的有些问题跨越了编程技术和调试技术的界限。虽然错误预防更可能是属于调试中的部分，但是为了避免犯错，也需要懂得编程常犯的错误。很多编程文章里没有提到调试和错误预防的内容，只有在这里介绍了。

很容易看出有些内容我们没有提及。我们基本上没有涉及到任何不属于 Visual C++或 Windows 的第三方调试工具，如 WinDbg。这样做有几个原因。最明显的原因就是我们把重点放在介绍调试概念上，而不是介绍工具。另一个重要的原因是，Visual C++程序员需要较好的知识，来利用已有的工具。最后，我们怀疑，我们所能介绍的那点东西能不能比厂商介绍的对大家更有帮助。

可我用的是 BoundsChecker 呀……

有些读者会想了："可我用的是 BoundsChecker 呀，为什么要读这本书呢？"问得好。调试工具，例如 Compuware NuMega 公司的 BoundsChecker 和 Rationale Software 公司的 Purify，可以很好地发现各种运行期错误，如不良指针和句柄、内存破坏（corruption）和泄漏（leak）、错误的 Windows API 参数，等等。但是它们做不到的是帮助理解调试过程，

包括如何使用 C++语言、断言、跟踪和异常来预防和消除错误，如何充分利用 Visual C++和 Windows 自带的调试工具来调试多线程程序和 COM 程序。这些本书全部都做到了。

而且，这些工具一定不能检测到所有的错误，并且不能用于预防错误。如果完全依赖调试工具，当遇到工具找不到的错误时，你就会很无助。另外，使用这些工具需要执行额外的开发步骤，它们对性能影响也很大，但是本书中介绍的错误检测技术在调试版本时自动进行，对性能的影响也比较小。调试是个很复杂的解谜过程，这些工具往往只能给与片面的帮助。

如何阅读本书

虽然我们希望读者能够一页页地阅读本书，但我们也了解读者通常都很忙，一般没有时间通读。所以，每章本身都是完备的，你可以根据需要只读某些章。完备，意味着可能有些冗余的部分，虽然我们尽力缩小了这些冗余，但它们仍然不只出现在一章之中。我们希望你能赞同这种方式，它使得本书成为一个更好用的手册。

在目录后，有一个常见问题的列表，可以帮助你很快地找到问题的答案。第 8 章是用同样的方式组织的。这一章和常见问题列表都能帮助你很快地找到某个调试问题的解决方法。

Windows 程序有很多种，所以也有许多不同的调试技术。我们将从 Windows API、MFC 和 ATL 应用程序基本结构的观点来介绍调试。我们会清楚地标出那些用于 MFC 和 ATL 的调试技术，所以当你不用到这些框架时，如果时间很紧，就不用看了。如果有时间，当然最好阅读一下，因为你可以从中了解到其他观点的调试技术。

最后，每章都有一个关键的调试建议，我们把它用这种形式表示：

这些提示突出了调试概念中最有用的地方，很容易看到，这可以帮助你快速找到重要的调试主题。

Windows 版本和硬件

为了简化陈述，我们只选用了当前版本的 Windows，在我写书的时候，主流操作系统是 Windows2000 和 Windows 98。这里针对 Windows2000 的内容也适用于 Windows NT 4.0，针对 Window 98 的内容也适用于 Windows 95。我们在说 Windows NT 4.0 和 Windows 95 时，仅仅指的是该版本的 Windows。

为了简化陈述，我们假设你在一个 Intel X86 的 CPU 上使用 Windows。虽然这本书大部分与 CPU 无关，但是在阅读 16 进制转储信息和在汇编级别调试时，CPU 就是一个必须考虑的因素了。对那些在非 Intel 平台上进行汇编级别的调试的程序员，我们只能说抱歉了。

“我” 是 谁

虽然封皮上只有两个名字，但每章都是用第一人称介绍的。Mike Woodring 写了第 10 章“调试多线程程序”和第 11 章“COM 调试”，Everett McKay 写了其他部分。所以，在第 10 章和第 11 章的“我”指的就是 Mike Woodring，而其他章节的“我”就是 Everett McKay。

更 新 信 息

这本书是在 Microsoft Visual C++ 6.0 的基础上写的。如果你使用更新的版本，这本书的大部分还能适用，只有编译器和调试器的一些细节会有些改变。为了让这本书的知识能够及时更新，我们在www.windebug.com下会贴出更新和更正信息。如果你发现了错误，或过时信息，请通知我们，发邮件给 corrections@windebug.com。

致 谢

这是一本很难写的书，并不是因为没有关于调试的知识，而是知识点太多了，并且非常分散。从很大程度来说，我们的工作就是收集、吸收，然后有条理地描述出来。本书中有不少原创思想，但只是少数部分。我很想感谢每一位对本书中涉及的调试思想有贡献的人，但是很难知道他们是谁。

如果不对 Windows 和 C++调试器有很深的了解，是不能写出这本书的。我应该感谢 John Robbins、Matt Pietrek 和 Paul Dilascia，他们在《Microsoft Systems Journal》中很多优秀的 Windows 调试文章。我还应该感谢 Steve Maguire 在《Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Programs》中的灵感，以及 Steve McConnell 在《Code Complete: A Pracitical Handbook of Software Construction》中的思想。感谢 Scott Meyers、Tom Cargill 和 Bjarne Stroustrup 关于如何正确使用 C++语言的著作。

我应该感谢调试的先驱：Glenford J. Myers、Brian W. Kernighan 和 P.J. Plauger，他们早期的工作成果至今仍对我们非常有帮助。最后，感谢 Rebort M. Pirsig，他的《Zen and the Art of Motorcycle Maintenance: An Inquiry into Values》对我的质量的概念确立有很大帮助。创造高质量的软件是本书的基本观点。

在构思这本书时，我花了很长时间研究各种新闻组论坛上关于调试的讨论，特别是

microsoft.public.vc.debugger。我应该感谢那些新闻组的参与者，他们在很多调试主题上给我提供了细节信息，而他们有的可能还不知道。感谢 Jay Bazuzi、Bruce Dawson、Raj Rangarajan、Scott McPhillips、Tomas Restrepo、Katy Mulvey、Doug Harison 和 Joseph Newcomer，他们常常发布一些很有价值的文章。

非常感谢那些相信我能写出这本书的人们。感谢前任策划编辑 Gary Clarke 和他的助手 Rebecca Bence，他们最终促成这个项目。特别感谢 Kristin Erickson，另一个编辑，感谢他在这个项目中从开始到结束的卓越工作。同样感谢 Sherri Dietrich、Connie Leavitt of Bookwrights、Joan Kocsis、Marilyn Rash、Angela Stone 和 Judy Strakalatitis，感谢他们杰出的编辑、制作和排版的工作。

如果没有从读者那里得到的有见解的反馈，这本书也不算什么。感谢 Mike Woodring、Eugene Kain、Christophe Nasarre、David Schmitt 和 Robert Ward。

写一本书需要做上千个的决定，我最成功的决定就是使 Mike Woodring 成为合作作者。Mike 写了第 10 章“调试多线程程序”和第 11 章“COM 调试”。他比我更加了解这些专业技术的调试知识。

最后，特别感谢我的妻子 Marie 和两个孩子，Philippe 和 Michele，他们在我写作这本书的几个月里默默奉献，如果没有他们的帮助我是很难完成这个工作的。

Everett N. Mckay

感谢那些持续发表有价值的调试方法的作者们。感谢你们的仁慈，将那些信息给大家共享。我特别感谢 Matt Pietrek 和他的《Windows Internals and Windows 95 System Programming Secrets》。这些作品激发了我对调试和崩溃分析的兴趣，否则我到现在仍是一个苦工。

我也很感激在 Intel 工作的那段日子。特别感谢 Jim Held 和 Kim Toll，我很高兴有机会和他们一起在调试方面工作。很幸运能和 Nate LeSane 一起工作，他让我度过了许多练习和磨练编程技术的快乐时光。虽然我离开 Intel 已经好几年了，但 Intel 从来没有离开我。

最近，我有机会在 DevelopMentor 和一些真正了不起的人们一起工作。在 DevelopMentor 里，我从我的学生们那里感到自己的渺小。我只希望他们能够从课堂练习那里，学到有我从他们身上学到的那么多的东西。

非常感谢 Everett 让我参与这本书的写作工作，感谢他让我对自己感兴趣领域的一本书做点贡献。当我知道 Everett 这本书的目的时，我就感觉到他要完成的是一本很复杂、高质量的书籍，这超过了我的能力。如果第 10 章和第 11 章中存在错误和疏忽，影响了 Everett 清晰的思路，那是我个人问题。

Mike Woodring

简　介

错误是无处不在的。程序产生错误的原因很多，也有很多防止、检测和消除错误的策略、工具和技术。随着软件开发技术的提高，软件的复杂程度也提高了，调试工作更难于进行。程序员们为开发出完美的无错误软件而不懈地努力，但这个目标在现实中是很难实现的。

复杂性是开发无错误软件的首要障碍。即使是开发最普通的程序，不经过细致的测试和调试，也不能断定其中不含任何错误。复杂庞大的程序是错误产生率很高的原因，编程工作却是最不能允许错误或不精确的人类活动。可以考虑一下，在没有编译器帮助的情况下，编写很短的一段没有语法错误的程序就不是一件容易的事情，更何况维护语法正确比保证语义正确相比，还是容易得多了。由此可以看出开发任何复杂度的软件都需要调试。如果你编写的软件是用于商业目的的，那么调试更是生存的唯一出路。

革新是开发无错误软件的另一个障碍。就在人们接近那把用于消除某类程序错误的钥匙时，有人却把锁给换了。现今的程序设计技术与五年前的相比，不管是外表上还是设计上都有本质的区别。许多非常有效的基于 C 语言的 Windows API 程序调试技术，对如今的基于 C++的 MFC 类库程序几乎不起任何作用。同样地，COM 技术、分布式 COM 技术、ATL 技术和多线程 Windows 程序也给调试领域带来了新的挑战。具有讽刺意味的是，含有大量错误的软件的持续存在，可以说是快速发展的技术革新带来的一大成就。也可以这么说，能够轻松地编写无错误软件的时候就是新技术不再出现的那一天。这个结论不是逃避，而是对于“我们创造错误的能力远远大于我们消除它们的能力”这一事实的认知。

紧密的开发进度也是开发无错误软件的障碍之一。理想中我们希望程序没有错误，但我们更希望它能在一个合理的进度安排下上市。管理人员对于维持进度的重视程度远远大于对于质量的重视，这也不是完全没有道理的。理想中的无错误软件往往需要太多的时间，以至于不能在这样一个竞争的商业环境下制作，因为客户不想等。在准确的市场时机上交付使用的即使是含有错误的软件，也可以带来利润，但是没有及时上市的完美程序却连一个子儿也赚不到。软件一般有有限的上市期，程序越晚上市，投资的回报时间就越少。而且，在市场时机已经过去，市场中的领导地位已经确立后，产品就很难挣钱了。你可以在以后经常修补错误，但失去市场时机后就很难取回投资了。现实中有很多远远不是无错误程序却取得成功的例子。

调试的定义

在深入探讨之前，我们先来确定一下调试的定义，并且和相关测试活动对比一下。

错误（bug），是程序执行中的缺陷，也可以说是代码中的差错。如果程序不能如预期的那样运行，我们就说它含有错误。显然，崩溃不是程序预期的动作，所以崩溃是错误。那么，如果程序运行速度很慢，或在拖动窗口时闪烁呢？从实际意图出发，这些性能中不理想的地方都可以称为错误。我习惯称之为小错误，虽然这些问题不令人满意，需要修改，但这些并没有严重到影响它的上市。

那么如何看待设计上的不足呢？比如可用性问题、界面不够友好了，或者不必要的重复命令了。从我们的定义看来，如果程序如预期那样运行，这些问题就不是错误。它们是设计上的失误造成的，不是程序实现上的错误。可能有人坚持认为设计不足也是错误，但是消除设计上的不足的过程，与消除实现上的错误的过程是完全不同的。毕竟，不能通过设置断点或察看调用堆栈来消除设计上的不足。所以，虽然用户和测试人员可能将设计不足看作错误，而记录在错误报告中，但是消除这些问题的方法是通过改变设计方案，而不是通过除去源代码中的错误。

调试就是预防、揭示、诊断和消除错误的过程。充分利用编译器、连接器和其他开发工具可以预防错误的产生，也可以通过使用某种可以预防特定种类错误的编程方式。通过在代码中加入调试语句（如断言语句），可以在程序执行时使错误自动暴露出来。当然，也可以尽量收集有关错误的信息，然后利用调试器寻找问题的根源所在，即诊断错误。当成功地消除了已有的错误，并且没有新的错误产生时，调试过程就结束了。从调试的定义可以看出，它有许多特性：它是前摄的（发现错误之前）、有效的（错误发现之后）、静态的（分析源代码）和动态的（执行程序）。

调试过程几乎包含了消除错误的全部步骤，除了一个很重要的步骤——测试。测试就是积极地检测错误的过程。测试既包括为了发现错误而人工运行程序的过程，也包括制作并运行自动程序使测试自动化的过程。测试过程也需要专门的用于测试的代码的开发。如果不连接那些用于揭露错误的调试代码，比如断言与跟踪等语句，程序仍是完整的，所以测试代码并不是程序源代码的一部分。准确地说，测试代码是独立的，只有在测试过程中才加入到源代码中。

对比一下这两个过程，测试发生在没有发现错误但却试图发现错误的时候，相反地，调试是编写代码预防错误或消除错误。调试经常是由程序员完成，需要修改代码，而任何人都可以测试一个程序，并不需要对源代码的了解。虽然调试和测试紧密结合，但它们的过程却完全不一样。此书集中介绍调试过程，但也讨论了一些与调试有关的测试过程。

调 试 目 标

本书主要讲实现调试目标的各种方法。调试的目的显然在于消除错误，但是如果想掌握高超的调试技能就必须更加努力。调试对于软件开发效率和软件质量至关重要。因此，从软件开发过程的观点来看，调试有以下目标。

- **效率**。调试提供了一种预防错误且尽可能地发现错误的代码开发方式。有效的错误预防、检测和消除是减少人力浪费的必须。
- **质量**。调试有助于发现错误的本质原因，且在不引入新错误的情况下消除它们。有效的错误预防、检测和消除是保证软件质量的必须。

有效的调试可以在许多方面提高软件开发效率，但当你精通相关调试技能后，这种提高就更明显了。很多公司拥有专门的调试和质量监测小组，他们负责查找程序员遗留下来的错误。从效率的角度来说，程序员造成的错误的数量和大小与其他人查找错误的能力有关。当然，更好的方式是，程序员查找并消除尽可能多的错误，这样调试人员就只要查找遗漏下来的很少的错误就可以了。

为了说明查找错误的方式如何影响开发的效率，我们比较一下这样两个过程。下面这个过程用于程序员自己查找并消除错误：

- 测试，发现了错误。
- 跟踪错误的产生，消除错误，验证这次修改的正确性。

比较起来，如果测试人员发现错误，开发小组要使用一个不同的过程：

- 程序的新版本发布了。
- 测试小组安装程序。
- 测试小组提出测试方案。
- 测试人员发现了错误。
- 测试人员书写错误报告，提交给管理部门。
- 管理部门检阅错误报告，交给合适的程序员，假设碰巧是你。
- 你浏览错误报告，很可能询问测试人员一些相关信息。
- 你重现错误。
- 你跟踪错误的产生，消除错误，验证这次修改的正确性。
- 你更新错误报告数据库，记叙这次修改。
- 你更新程序，将它发回给测试人员。
- 测试人员安装修正过的程序，浏览错误报告。
- 测试人员确认错误已经被清除，且没有产生新的错误。
- 测试人员更新错误报告数据库。
- [可选]即使错误被清除了，在程序上市前的每个星期你都必须参加错误情况会议，而且管理人员还会问你错误是否真地被修正了。

这个过程，当然是情况发展比较顺利的时候，测试人员能够发现错误，你可以在第一次尝试的时候就重现并消除它。但如果不是这样，这个过程将变得异常复杂。这两个过程都需要程序员重现和消除错误，但是用到测试人员的过程就牵扯了更多的人，也就需要更多的时间来进行沟通和管理等工作。机能不良的调试过程有这样一个特点，就是它永远都在持续地做着试图结束这个工作的事情。现在你可以了解调试的重要性了吧。

有些人认为调试在软件质量过程中并不占有一席之地，他们认为软件质量是在开始时就固定在程序里的。这种言辞只会误导别人。完美的设计，实现却很草率，甚至充满着错误，这在任何人的概念里都是质量差的软件。

如果软件不可能完全没有错误，那么它应该如何面对余留的错误呢？这个问题从软件本身的角度提出了两个目标。在存在问题的情况下，程序必须保证以下两点：

- **预防垃圾数据**。如果继续运行程序，可能会有损害数据或被这些数据控制的外设的危险，那么这个程序必须被中止，不能破坏这些数据。
- **继续运行**。如果继续运行并不会损害数据或外设，这个程序就可以继续运行（可能是在一个退化状态下）。如果问题很重大，就需要通知用户，由用户来决定关闭程序还是继续运行。

如果这两点目标没有达到，就会使原始错误更加严重。第二个目标与当前大多数程序的作为恰恰相反，这些程序在一检测到错误时就自动中止。但是，这种自动中止并不是用户最想要的，尤其是在工作可以挽回却丢失的情况下。如果程序不能决定数据的状态和它保留这些数据的能力，就可以将数据保存在临时文件中，让用户来决定是否数据值得保留。

对更好的调试技术的需求

调试显然是一项重要的编程技术。程序员常常需要比开发时间一半还多的时间，去测试和调试代码，但是他们很少花力气去提高调试技术。在这方面，调试技术就像天气一样，虽然每人都会谈论它，但没有任何人去对付它。如果认真地面对它，任何程序员的测试技术都会提高。考虑一下程序员在调试上花的时间，你会奇怪这么多的力气居然没有花在提高调试技术上。

缺乏这种努力的一个原因是调试技术很难学习。这并不是说关于调试技术没有足够的知识，相反，这些知识的数量之大足以让你淹没其中，但是它们的质量却从非常有帮助到一点用处也没有都有分布。比如，为了写这本书，我们参考了 40 多本书籍，将近 100 篇杂志论文，许多 MSDN 文章、在线文章、时事通讯文章、会议记录等等，三盘录像带，数以百记的新闻组讨论。典型的讨论调试知识的文章不多，它们要么只是肤浅地介绍一下调试过程，要么过分深入地处理一个狭窄的调试论题，这些调试论题是从其他调试研究和关注分离出来的。对这些信息进行分类需要很多时间和很强的动力，从我们身上就可以得到证明。由于信息缺乏系统性和连贯性，调试不是可以让某人宣称自己是这方面大师的一门技术，但是却可以在不断积累中变得丰富。

战略性调试

写这本书的主要目的是为了确认调试问题和介绍基本调试知识，以便你更加有效地使用 Visual C++调试 Windows 程序。重点介绍可用于商业软件开发的实用解决方案。这本书的最终目的是使你的编程劳动更多产，软件更可靠。

我们试着从战略的角度介绍调试基本技术。就是说，并不是简单地描述调试技术，然后说些“这种技术好极了，你可以不受限制地使用”之类的话，而是介绍了一种战略上的方法，不但讨论如何使用这种调试技术，而且说明了为什么使用、什么时候使用以及这种方法在其他调试开发中如何移植。我们坚信，使用一个系统的方法远比使用那些混乱的调试技术取得的结果要好得多。

我们选择了 Visual C++和它的 MFC 与 ATL 应用程序框架作为调试的环境，其中很多方法也可以适用于其他 Windows C++的开发环境，甚至其他 Windows 编程语言。但是，人的精力毕竟是有限的，我们还是愿意集中在一个特定的环境下进行讲叙，而将到其他环境和编程语言的移植作为练习留给读者。很多情况下，这种移植很容易，这是我们从经验中得到的，因为参考的 40 多本书中只有 3 本专门介绍了 Visual C++中的调试。

常 见 问 题

目　录

第二部分 调试工具

第三部分 调试技术