

本书仅提供部分阅读，如需完整版，请联系QQ: 2404062482

提供各种IT类书籍pdf下载，如有需要，请QQ:2404062482

注：链接至淘宝，不喜者勿入！整理那么多资料也不容易，请多多见谅！非诚勿扰！

[点击购买完整版](#)

Debugging

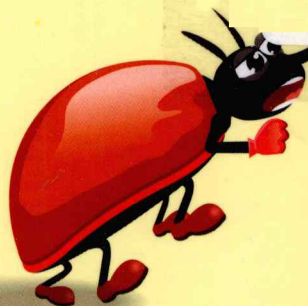
The 9 Indispensable Rules for Finding

Even the Most Elusive Software and Hardware Problems

调试九法

软硬件错误的排查之道

[美] David J. Agans 著
赵俐 译



人民邮电出版社
POSTS & TELECOM PRESS

Debugging

The 9 Indispensable Rules for Finding

Even the Most Elusive Software and Hardware Problems

调试九法



软硬件错误的排查之道

硬件缺陷和软件错误是“技术侦探”的劲敌，它们负隅顽抗，见缝插针。本书提出的九条简单实用的规则，适用于任何软件应用程序和硬件系统，可以帮助调试工程师检测任何bug，不管它们有多么狡猾和隐秘。

作者使用真实示例展示了如何应用简单有效的通用策略来排查各种各样的问题，如芯片过热、由蛋酒引起的电路短路、触摸屏失真等，给出了真正能够隔离关键因素、运行测试序列和查找失败原因的技术。

无论你的系统或程序发生了设计错误、构建错误还是使用错误，本书都可以帮助你用正确的方法来思考，使bug自动暴露，进而一网打尽，斩草除根。

David J. Agans

资深调试专家，善于解决一些最棘手的调试问题，涉及工业控制和监视系统、集成电路设计、掌上电脑、视频会议系统等。1976年毕业于麻省理工学院，现为SeaChange International工程总监。曾经营计算机系统咨询公司PointSource，任Zydacron公司副总裁，还曾就职于Gould、仙童和DEC等知名企业。

- PLC之父鼎力推荐
- 亚马逊全五星畅销图书
- 软硬件调试的通用秘籍

AMACOM

图灵网站: www.turingbook.com 热线: (010)51095186

反馈/投稿/推荐信箱: contact@turingbook.com

有奖勘误: debug@turingbook.com

分类建议 计算机/软件工程

人民邮电出版社网址: www.ptpress.com.cn



ISBN 978-7-115-24057-6



9 787115 240576 >

ISBN 978-7-115-24057-6

定价: 35.00元

TURING 图灵程序设计丛书

Debugging

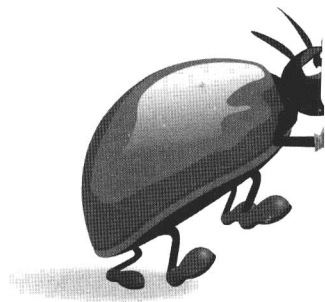
The 9 Indispensable Rules for Finding

Even the Most Elusive Software and Hardware Problems

调试九法

软硬件错误的排查之道

[美] David J. Agans 著
赵俐 译



人民邮电出版社
北京

图书在版编目(CIP)数据

调试九法：软硬件错误的排查之道 / (美) 阿甘斯
(Agans, D. J.) 著；赵俐译. — 北京：人民邮电出版社，
2011.1

(图灵程序设计丛书)

书名原文：Debugging: The 9 Indispensable Rules
for Finding Even the Most Elusive Software and
Hardware Problems

ISBN 978-7-115-24057-6

I. ①调… II. ①阿… ②赵… III. ①电子计算机—
调试 IV. ①TP306

中国版本图书馆CIP数据核字(2010)第209193号

内 容 提 要

本书主要介绍了调试方面的9条黄金法则，并结合实际的环境讲述了如何合理地运用它们。本书的内容没有针对任何平台、任何语言或者任何工具，讲述的重点是找到出错的原因并修复它们，高效地追踪和解决不易察觉的软硬件问题。

本书适合所有软硬件从业人员阅读。

图灵程序设计丛书

调试九法：软硬件错误的排查之道

◆ 著 [美] David J. Agans

译 赵 俐

责任编辑 傅志红

执行编辑 谢灵芝

◆ 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>

北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16

印张：9.75

字数：147千字

印数：1-3 000册

2011年1月第1版

2011年1月北京第1次印刷

著作权合同登记号 图字：01-2010-5547号

ISBN 978-7-115-24057-6

定价：35.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223

反盗版热线：(010)67171154

译者序

有人说调试是一门艺术，这不无道理，但本书作者认为它并不仅仅是艺术，更多的是科学，调试人员也不仅仅是艺术家，还是科学工作者。遵循本书所讲的9条规则，就可以把调试艺术转化为科学。

本书翻译到一半的时候，我已经钦佩不已。它绝对称得上调试领域的经典之作，但显然，在某种程度上它并没有引起国内业界的注意。常言道“千里马常有，而伯乐不常有”，虽然用千里马来形容一本书多少有些不恰当，但我确实觉得本书被埋没了，我想我们应该感谢人民邮电出版社图灵公司，把这样一本好书发掘出来，让国人有机会分享这位拥有二十多年实践经验的调试高手的知识和经验。

把书写厚了容易，写薄了却难，我想这一点大家都会认同。作者正是用这么薄薄的一本书讲述了适用于软件、硬件、工程领域的9条基本调试规则。这些规则甚至还适用于我们的日常生活，例如解决汽车和房屋问题。仔细揣摩，我们会学到不少生活知识，这也是阅读本书的一个额外的好处。

本书就像是一碗心灵鸡汤，也像是一坛陈年佳酿，书中所举的一些案例散发着古朴的气息。虽然我没有怀旧情节，但仍感到亲切而自然，有那么一刻，我与作者灵犀相通，仿佛他就站在那里，正在向我微笑，与我倾谈。

作者是个福尔摩斯迷，我想这是不是与他的职业生涯有关呢？在bug面前，他就是一名侦探。每章的开头都会引用福尔摩斯的一句名言，暗合该章的主题，也为本书平添了一层神

秘的色彩。我想要是把“神探狄仁杰”的故事讲给他听，他也一定会非常感兴趣，尽管他可能连狄仁杰是谁都不知道。

在本书翻译的过程中，有些地方我思索良久，有些则需要查询一些相关知识。我觉得提供一点背景资料会有助于理解，因此加了一些脚注，对于那些不甚了解相关背景或知识的人，可能会有一点帮助作用，但有些读者可能很熟悉硬件、软件和工程领域，如果这些内容都是你所熟知的，那么请恕我赘述之过。

最后，本书在翻译的过程中得到了人民邮电出版社图灵公司编辑们的精心指导和宝贵意见，帮我纠正了翻译中的很多错误，使我受益匪浅。由于水平有限，难免还会留有一些错误，恳请读者批评指正。

致 谢

本书的孕育可追溯至1981年，当时Gould公司的一组测试工程师问我是否能用一篇文档写清楚如何解决硬件产品问题。我听了之后有点不知所措，因为我们的产品是一些由上百块芯片、几个微处理器和无数通信总线组成的主板。我深知没有“魔力配方”，他们必须学会如何调试。我与Mike Bromberg（他一直是我的良师益友）讨论了这件事，我们达成共识——最起码要编写一些通用的调试规则。于是我最后编写了“Ten Debugging Commandments”（调试十诫），这是一页简短的调试规则，测试小组很快把它贴到了墙上。几年后，这份清单已被压缩为一条规则，而且被推广应用到很多软件和系统，但它仍然是本书的核心。因此感谢Mike和那些提出这个请求的基层技术人员。

感谢Doug Currie、Scott Ross、Glen Dash、Dick Morley、Mike Greenberg、Cos Fricano、John Aylesworth（原来提出请求的技术人员之一）、Bob DeSimone和Warren Bayek，他们使得那些富有挑战性的工作变得充满乐趣。一直以来，我都很高兴能够为他们工作并与他们并肩作战，他们带给我无数灵感，帮助我培养了调试技巧，也教会了我幽默。还要感谢三位追求卓越并为我的学习过程带来乐趣的老师，他们是Nick Menutti（虽然这不是诺贝尔奖，但这里让我奉上对你的赞誉）、Ray Fields和Professor Francis F. Lee。还要感谢我一直未曾谋面的几位作者，他们的书对我的创作产生了巨大影响，他们是William Strunk Jr和E. B. White（著有*The Elements of Style*）、Jeff Herman和Deborah Adams（著有*Write the Perfect Book Proposal*）。

感谢包容我28年之久的夏日垒球球队Delt Dawgs，感谢朋友们的审阅和帮助。Charlie Seddon详细审阅了本书并给出很多有益的评论，Bob Siedensticker也审阅了本书并为我提供了案例、主题建议和出版建议，对此我感激不尽。还有几位当时我还不认识的朋友审阅了本书并给我寄来他们的意见，本书的出版离不开他们的帮助。他们是Warren Bayek和Charlie Seddon（上面提到过）、Dick Riley、Bob Oakes、Dave Miller和Terry Simkin教授，感谢他们付出的时间和提出的意见。

感谢Sesame工作室的Tom和Ray Magliozzi。（“侃车^①”节目的主持人Click和Clack，抑或是Clack和Click^②？）另外感谢Steve Martin允许我使用他们的故事和笑话，还要感谢Arthur Conan Doyle（柯南道尔爵士）创造了福尔摩斯这个侦探形象，并通过他表达了如此多的妙语，此外感谢Seymour Friedel、Bob McIlvaine和我的兄弟Tom Agans为我讲述妙趣横生的案例。发现和证明这些规则离不开他们提供的例子，感谢所有案例故事的参与者，感谢“英雄和笨蛋^③”（你们知道我说的是谁）。

与Amacom的编辑们一起工作是一段美妙的经历，而且给了我很多启发。感谢Jacquie Flynn和Jim Bessent，感谢他们的热情和中肯的建议。感谢在出版过程中作出贡献的设计者和才华横溢的人们，正是由于他们的出色工作，才使本书成功出版。

特别感谢我的代理人Jodie Rhodes，感谢他给了我初次写书的机会，让我用别具一格的方式来写这个非同寻常的主题。他了解他的市场，事实也证明他是正确的。

感谢我的亲人Dick和Joan Blagbrough给予的支持、鼓励和大量帮助。特别感谢我的女儿Jen和Liz，我要拥抱和亲吻她们，她们非常可爱，也感谢她们给予我的信任。（还要感谢她们每天晚上在用IM聊天和玩游戏的间隙把电脑让给我用。）

最后，我要把永恒的爱和感激献给我的妻子Gail，感谢她鼓励我把这些调试规则写成一

① 侃车，Car Talk，美国国家公众广播电台的一档节目。（如无特殊说明，本书中的脚注均为译者注。）

② Click和Clack，上述“侃车”节目二位主持人Tom和Ray Magliozzi的昵称，指车子运行时发出的咔咔声。

③ 《英雄和笨蛋》，*heroes and fools*，龙枪系列小说之一，作者在这里借用此名开了个玩笑。

本书，感谢她督促我寻找代理人，感谢她给予我创作的时间和空间。也感谢她校对了大量的初稿，要知道，没有她的校对，这些书稿我甚至不敢拿给别人看。她可以用一个吸尘器点亮吊灯^①，却完全凭借她自己点亮了我的人生。

Dave Agans

2002年6月

^① 第13章中的案例故事有具体的解释。

目 录

第 1 章 简介.....	1	4.3 引发失败.....	25
1.1 本书如何教会你调试.....	1	4.4 不要模拟失败.....	25
1.2 这些规则都很显而易见.....	2	4.5 如何处理间歇性 bug.....	27
1.3 本书适用于任何人.....	3	4.6 如果做了所有尝试之后问题仍然 间歇性发生.....	29
1.4 本书可用于调试各种问题.....	3	4.6.1 仔细观察失败.....	29
1.5 本书的主旨不在预防、保证或筛选.....	4	4.6.2 不要盲目相信统计数据.....	30
1.6 调试不仅仅是故障检修.....	5	4.6.3 是已修复 bug，还是仅仅由于 运气好，它不再发生了.....	31
1.7 有关案例故事.....	6	4.7 “那不可能发生”.....	33
1.8 精彩内容，即将上演.....	6	4.8 永远不要丢掉调试工具.....	34
第 2 章 总体规则.....	8	4.9 小结.....	36
第 3 章 理解系统.....	10	第 5 章 不要想，而要看.....	37
3.1 阅读手册.....	12	5.1 观察失败.....	41
3.2 逐字逐句阅读整个手册.....	13	5.2 查看细节.....	43
3.3 知道什么是正常的.....	15	5.3 问题忽隐忽现.....	46
3.4 知道工作流程.....	16	5.4 对系统进行插装.....	46
3.5 了解你的工具.....	17	5.4.1 设计插装工具.....	46
3.6 查阅手册.....	18	5.4.2 过后构建插装.....	48
3.7 小结.....	20	5.4.3 不要害怕深入研究.....	50
第 4 章 制造失败.....	21	5.4.4 添加外部插装.....	51
4.1 制造失败.....	24	5.4.5 日常生活中的插装.....	51
4.2 从头开始.....	24	5.5 海森堡测不准原理.....	52

5.6 猜测只是为了确定搜索的重点目标	53	第 10 章 获得全新观点	93
5.7 小结	54	10.1 寻求帮助	94
第 6 章 分而治之	55	10.1.1 获得全新观点	94
6.1 缩小搜索范围	59	10.1.2 询问专家	94
6.1.1 确定范围	60	10.1.3 借鉴别人的经验	95
6.1.2 你在哪一侧	61	10.2 到哪里寻求帮助	96
6.2 插入易于识别的模式	62	10.3 放下面子	97
6.3 从有问题的支路开始查找问题	63	10.4 报告症状, 而不是理论	98
6.4 修复已知 bug	64	10.5 小结	99
6.5 首先消除噪声干扰	65	第 11 章 如果你不修复 bug,	
6.6 小结	66	它将依然存在	101
第 7 章 一次只改一个地方	67	11.1 检查问题确实已被修复	103
7.1 使用步枪, 而不要用散弹枪	69	11.2 检查确实是修复措施解决了问题	103
7.2 用双手抓住黄铜杆	71	11.3 bug 从来不会自己消失	104
7.3 一次只改变一个测试	72	11.4 从根本上解决问题	105
7.4 与正常系统进行比较	73	11.5 对过程进行修复	107
7.5 自从上一次能够正常工作以来你更 改了什么	74	11.6 小结	107
7.6 小结	77	第 12 章 通过一个案例讲述所有规则	109
第 8 章 保持审计跟踪	78	第 13 章 牛刀小试	113
8.1 记下你的每步操作、顺序和结果	80	13.1 灯和吸尘器的故事	113
8.2 魔鬼隐藏在细节中	81	13.2 大量出现的 bug	115
8.3 关联	83	13.3 宽松的限制	119
8.4 用于设计的审计跟踪在测试中也非 常有用	84	13.4 识破 bug	123
8.5 好记性不如烂笔头	84	第 14 章 从帮助台得到的观点	128
8.6 小结	85	14.1 帮助台的限制	130
第 9 章 检查插头	86	14.2 规则, 帮助台风格	130
9.1 怀疑自己的假设	88	14.2.1 理解系统	131
9.2 从头开始检查	89	14.2.2 制造失败	132
9.3 对工具进行测试	90	14.2.3 不要想, 而要看	132
9.4 小结	92	14.2.4 分而治之	134
		14.2.5 一次只改一个地方	134

14.2.6 保持审计跟踪	135	第 15 章 结束语	139
14.2.7 检查插头	136	15.1 调试规则网站	139
14.2.8 获得全新观点	136	15.2 如果你是一名工程师	139
14.2.9 如果你不修复 bug, 它将 依然存在	137	15.3 如果你是一名经理	140
14.3 小结	137	15.4 如果你是一名教师	141
		15.5 小结	141

第1章

简介



“你知道，现阶段我非常忙，但我打算在晚年倾力写一本书，把所有侦探艺术都集中写到这本书里。”

——福尔摩斯，《格兰其庄园》

本书告诉你如何快速找到工作中的错误。它很短，也很有趣，因为它必须如此——如果你是一位工程师，你每天都在忙于调试，可能除了看点漫画之外就没时间读别的了。即使你不是工程师，也经常会遇到问题，这时你必须查明如何解决问题。

可能有人从来不需要做调试工作。或许你正忙着赶在公司倒闭之前把通过dot.com IPO^①发行的股票卖出去，因而只是让你手下的人去查找问题。或许你总是很幸运，你的设计一直未发生问题，或者bug总是很容易找到（尽管这不太可能）。有可能在你和你的所有竞争对手的设计中都有—些很难查找的bug，谁能够最快地修复它们，谁就占据了优势。当你快速找到bug时，不仅能够更快地为客户提供更高质量的产品，而且也能够更早下班回家，与家人一起享受美好的时光。

因此，请把这本书放在你的床头柜上或洗手间里，两周后，你就会成为一位调试高手了。

1.1 本书如何教会你调试

为什么这样一本简短且易读的书会这么有用呢？根据我26年的系统设计和调试经验，我

^① dot.com IPO，是指通过互联网公司上市募集资金。IPO，即首次公开募股（Initial Public Offering）。

发现了两件重要的事情（如果你把“从咖啡壶里倒出的第一杯咖啡含有全部的咖啡因”这样显而易见的错误也当成重要的问题，那么在你看来重要的事情就不止两件了）。

(1) 如果查找一个bug花费了大量时间，那么原因可能是忽略了某个最基本的、最重要的规则，一旦应用了那条规则，很快就会找到问题。

(2) 擅于快速调试的人已经深刻理解并应用了这些规则，而那些很难理解或使用这些规则的人则很难找到bug。

我把这些基本规则编写成一个清单，并教给其他工程师，我发现他们的调试技术和速度都提高了。这些规则的的确确起了作用。

1.2 这些规则都很显而易见

当你读到这些规则时，你可能会自言自语地说：“这些都是一些明显的规则啊。”不要着急下结论，这些规则确实都很明显（而且常常是基本的规则），但如何把它们应用于特定的问题就不总是那么显而易见了。而且不要把“显然”和“容易”混淆在一起，这些规则遵守起来并不总是那么容易，因此在解决实际问题时常常被忽略。

关键是记住并应用这些规则。如果这很明显而且容易，那么我就不必总是提醒工程师们应用这些规则，我也不必通过几十个案例故事^①来说明不遵守这些规则将会发生什么情况。能够自如地运用这些规则的调试人员是凤毛麟角。我喜欢问求职者这样一个问题：“你调试时使用什么拇指规则^②？”奇怪的是，很多人都回答说：“艺术。”好极了，那么让毕加索来调试我们的图像处理算法吧。事实上，利用简单和艺术的方法未必就能快速找到问题。

本书把这些“明显的”规则收集到一起，帮助你记住它们，知道它们的益处，并掌握如何运用它们，从而帮助你抵挡住“走捷径”的诱惑，因为捷径往往是陷阱。本书将把调试艺

① 案例故事，war story，原指战争故事，后泛指一些给人留下深刻印象的经历，在本书中则是作者举出的一些经典的、有代表性的案例。

② 拇指规则，英文为rule of thumb，又译为“大拇指规则”或“经验法则”，是一种可用于许多情况的简单的经验性的原则。

术转变为一门科学。

即使你已经是一位非常优秀的调试人员，这些规则仍然能够帮助你更上一层楼。当我把本书早期手稿拿给经验丰富的调试人员审阅时，他们不约而同地表示，本书除了教会一两个他们没有用过（但将来会用到）的规则之外，还帮助他们明确意识到了他们在不知不觉中遵守的规则。团队领导者（当然是顶尖的调试人员）指出，本书为团队提供了一种很好的沟通语言，使他们能够把技巧传授给其他成员。

1.3 本书适用于任何人

本书通篇都用“工程师”这个词来指代读者，但即使你不是工程师，这些规则对你也非常有用。当然，如果你正在查找设计中的错误，本书就更有用了，无论你是工程师、程序员、技师、客户支持代表，还是顾问。

如果你不直接参与调试工作，而是负责管理调试人员，那么也可以把这些规则传授给你的工作人员。你甚至不必理解他们所使用的系统和工具的细节，因为本书所讲的都是些非常基本的规则，因此在读完本书后，即使你是一位“尖发经理^①”，也能够帮助那些比你聪明得多的团队成员更快地找到问题。

如果你是一位教师，你的学生将会非常喜欢书中的案例故事，这些故事将为他们带来真实世界的体验。当他们走出校门时，将会比那些经验丰富（但没有受过调试培训）的竞争对手们更有优势。

1.4 本书可用于调试各种问题

本书具有很强的通用性，它并不是讲特殊的问题、工具、编程语言或特殊的机器。相反，本书讲的都是通用的技术，它们可以帮助你找到任何问题，无论你使用的是什么机器、语言

^① 尖发经理（pointy-haired manager），指发型向上翘起，这是斯科特·亚当斯（Scott Adams）绘制的漫画Dilbert中的一个非常有趣的角色，他管理一家高科技公司的一个部门，但看上去对他下属所做的事情却毫不知情。

和工具。本书讲了一种查找问题的全新方法，例如，它不是告诉你如何在Glitch-O-Matic数字逻辑分析器上设置触发器，而是告诉你为什么必须使用分析器，即使把它挂接到系统需要费很大一番工夫。

本书也适用于修复各类问题。你的系统可能在设计、构建和使用中有错误，或者只是被破坏了，无论是什么情况，这些技术都将帮助你快速找到问题的核心。

本书介绍的方法甚至不仅限于工程领域，虽然它们都是从工程环境中总结出来的。这些方法也可以帮助你查找其他方面的问题，例如汽车、房屋、音响设备、管道，甚至是你的身体（本书中会有例子）。但不可否认的是，有些系统并不适合使用这些技术，例如经济学就不适用，因为它太复杂了。

1.5 本书的主旨不在预防、保证或筛选

虽然本书介绍的方法和系统都是通用的，但它们都紧紧围绕一个重点，那就是查找bug的根源并修复。

本书所讲的并不是像ISO-9000、代码评审或风险管理这样的质量改进过程，这些过程主要强调的都是防止bug的产生。如果你对这方面的内容感兴趣，我可以推荐几本好书，例如*The Tempura Method of Totalitarian Quality Management Processes*和*The Feng Shui Guide to Vermin-Free Homes*。质量保证过程所涉及的技术都很有价值，但它们往往不易实现，即使实现了，系统中仍然会留有一些bug。

一旦有了bug，就必须检测它们，这项任务一般由质量保证（QA）部门来完成，如果没有这个部门，那么就只能由客户方来做了。本书也不会讨论这个阶段，因为已经有很多资源详细讨论了测试覆盖分析、测试自动化和其他质量保证技术。当你在产品线上检查6 467 826种选项组合时，可以找本这方面的经典书来打发时间，例如*How Do I Test Thee, Let Me Count the Ways*。

迟早会有一种组合失败，这时某位质量保证人员或客户就会起草一份bug报告。接下来，一些经理、工程师、销售人员和客户支持人员可能会召开一次“bug筛选会议（triage

meeting)”，激烈地讨论这个bug的重要性，以及是否需要修复它（何时修复）。虽然这个主题与你的市场、产品和资源密切相关，但本书绝对不会去触及它。但是，当人们决定修复bug时，肯定会看一下bug报告并且想弄清楚“这究竟是怎么发生的”，这时就到了阅读本书的时候了（参见图1-1）。

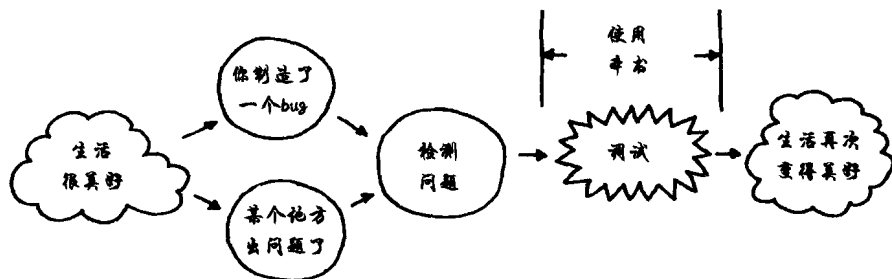


图1-1 何时使用本书

下面的章节将教给你如何准备查找bug，如何挖掘并仔细审查各种线索，以便找到根源，追踪实际问题，并修复它，然后确认你已经修复问题，这样你就可以高高兴兴地回家了。

1.6 调试不仅仅是故障检修

虽然调试和故障检修（troubleshooting）这两个词常常混用，但它们实际上还是有区别的，而且本书作为一本调试书，与其他数以百计的故障检修指南也是存在区别的。调试通常是查明为什么一个设计没有按计划工作，而故障检修通常是在已知设计没有问题的情况下，查明一件产品出了什么问题——可能是某个文件被删除了，某条线断了或者是某个元件出了问题。软件工程师做调试工作，汽车机修工做故障检修工作，而汽车设计师做调试工作（在理想世界中）。医生给病人看病就相当于“故障检修”，医生永远没有调试的机会了。（因为上帝已经完成了调试的任务，他花了一天时间设计了人类并造出原型，又在同一天把他的产品投放到人间，看起来上帝好像很赶时间！我想我们可以原谅他优先处理重要的方面，而给我们留下了像“拇指外翻^①”和“男性规律性脱发”这样的小bug。）

^① 医学术语，拇指由于囊肿胀而外翻，也称为拇囊炎。

本书中所讲的技术既适用于调试，也适用于故障检修。这些技术并不关心问题是怎么产生的，而是告诉你如何找到它。因此，无论是设计出了问题，还是部件有了毛病，都可以利用这些技术来查找。相反，有关故障检修的书只是在部件有问题的情况才有用。它们用几十张表格列出某个系统可能出现的一切症状、问题和修复方法。这些都很有用，它们是该类型的系统过去曾经出现过的一切问题的汇总，并且说明了症状和修复方法。它们把很多人积累的经验提供给故障检修人员，并帮助快速找到已知的问题。但是，当出现了新的、未知的问题时，它们就没有多大作用了。因此，这些书对设计问题几乎没什么作用，因为工程师们都非常有创造力，他们“喜欢”制造新的bug，而不会再用那些旧的bug来考验你。

因此，如果你正在检修一个标准系统的故障，那么不要忽略了规则8，查询一下故障检修指南，看看其中是否已经列出了你的问题。但如果它没有列出来，或者给出的修复方法不起作用，又或者你正在调试世界上第一个数字化的传输系统，因此根本没有故障检修指南，你不必担心，因为本书中所讲的规则将帮你找到新问题的核心。

1.7 有关案例故事

我出生于1954年，是一名美国电子工程师。在问题的讨论中，我所讲述的“案例故事”都是真实的，这些故事是我那个时代的人所熟知的。有些故事可能是你不了解的，因此有些我提到的事情你可能不理解。如果你是一位汽车机修师，可能不知道中断（interrupt）是什么。如果你生于1985年，你可能不知道什么是电唱机。但这没关系，重要的是我要通过这些故事说明的原则，而且我在讨论的过程中会给出足够多的解释，确保让你能够理解这些原则。

你知道，有些细节被我改动了，以便保护个人隐私，特别是保护那些犯了错的人。

1.8 精彩内容，即将上演

本书将介绍9条调试的黄金规则，每章介绍一条。每章的开头将讲述一个案例故事，通过它来说明规则对成功的重要性。然后描述规则并证明它如何应用于前面的故事。我会讨论

思考和使用规则的各种方式，你在面对复杂的技术问题时能很容易就想起它们（当然，简单问题也同样如此）。我还会给出规则的一些变化形式，证明它们也适用于其他方面，例如汽车和房屋。

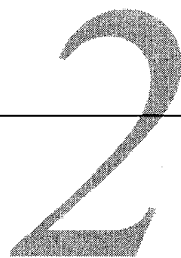
在最后几章中，我提供了一组案例故事，用来检验你对本书的理解，还有一节是练习在一个试验性的帮助台环境中使用这些规则，最后给出一些有助于把学到的东西应用于实际工作的小技巧。

读完本书后，你的调试效率将会大大提高。你甚至会喜欢到处转转，看看哪些工程师陷入困境，然后施以援手，帮助他们化解问题。但我给你提个小建议：紧身衣和披风还是不要穿了，把它们留在家里吧^①。

^① 作者开的一个玩笑，意思是不必打扮成“蜘蛛侠”的样子，蜘蛛侠是美国电影《蜘蛛侠》里的角色，热衷于拯救别人于危难，每次出行都要穿上紧身衣和披风。

第2章

总体规则



“我在这里要讲的理论（可能你认为它们非常荒谬），实际上都是非常实用的，我就是靠着它们挣得我这份面包和奶酪的。”

——福尔摩斯，《血字的研究》

下面就是本书要讲的规则。记住它们，把它们贴到你房子里的所有墙上。（你可能立即会想到：“调试规则”墙纸，用它们来装饰出一间别具匠心的家庭办公室，但“风水先生”肯定不会推荐你这样做。）

调试规则

规则1 理解系统

规则2 制造失败

规则3 不要想，而要看

规则4 分而治之

规则5 一次只改一个地方

规则6 保持审计跟踪

规则7 检查插头

规则8 获得全新观点

规则9 如果你不修复bug，它将依然存在

第3章

理解系统

“人们要想掌握本书中所有有用知识也并非完全不可能，事实上我就是尽全力这样做的。”

——福尔摩斯，《杀人的五个橘核》



案例故事 我刚出大学校门那会儿，急于积累点经验（当然还有别的目的），于是找了份夜间的兼职工作，构建一个用微处理器操控的进料阀控制器（valve controller）。这台设备的用途是控制添加到模具中的金属粉末的进料量，它通过一台天平来称重（参见图3-1）。像很多工程师一样（特别是那些“乳臭未干”的工程师），我复制了一个基本设计（从我大学联谊会的一位好友那里弄来的，他的毕业论文使用的就是这种处理器芯片）。

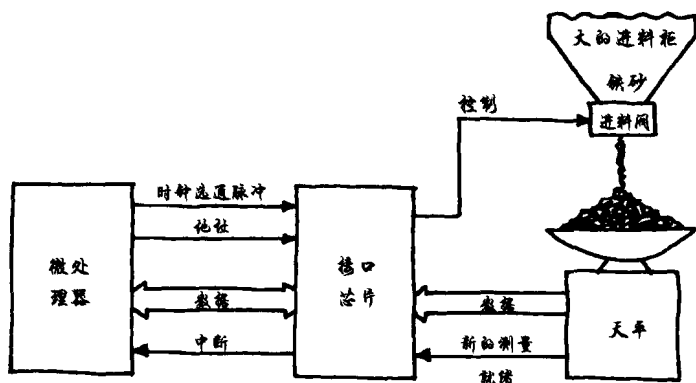


图3-1 由微处理器操控的进料阀控制器

但我的设计却无法工作。当天平为了进行一次新的测量而试图中断处理器的时候，处理器不响应它的中断请求。由于我是在晚间工作，没有很多工具可用来调试，因此费了好长时间才查明原因——芯片接收到来自天平的中断信号后，并没有把信号传递给处理器。

我当时与一位软件工程师合作，在凌晨1点的时候，我们都非常疲倦了，他坚持让我从头至尾读一遍数据手册。我照做了，在第37页，我读到“芯片将在第一个取消选中的时钟选通脉冲上中断处理器”，这等于是说“当电话铃声第一次响起时，中断就会发生，但这通电话不是打给芯片的”。然而，在我的设计中永远也不会发生中断，原因在于，为了节省硬件，我把时钟和地址线合并到一起了（我那位朋友也是这样做的）。继续电话这个比喻，只有当电话打给芯片时，电话铃才会响起。我那位朋友的系统不需要任何中断，因此可以正常工作。当我增加了额外的硬件后，我的系统也正常运转了。

后来，我同我父亲谈起了这次马拉松式的调试经过（他懂一点电子知识，但对微处理器一无所知）。他说：“这只是常识——当所有方法都不管用时，读读指令。”他的话给了我第一个启示，使我隐隐感觉到有一些通用的调试规则可以应用于计算机和软件之外的其他更多事情。（我还认识到，虽然我受过大学教育，也不足以让我父亲对我刮目相看。）这里，我要讲的规则是“理解系统”。■

在理解芯片的工作原理之前，我根本无法对问题进行调试。一旦理解了它，问题也就显而易见了。记住，系统的主要部分我都理解，这很重要。我知道设计的工作原理，我知道天平必须中断处理器的运行，也知道时钟选通脉冲和地址线都是做什么的。但我没有仔细研究所有细节，因此遇到了麻烦。

你必须掌握系统的工作原理以及它是如何设计的。在某些情况下，还要知道为什么这样设计。如果你没有理解系统中的某个部分，那么这通常就是出问题的地方。（这不仅仅是“墨

菲定律”^①的问题，如果你不能理解你所设计的系统，你的工作可能会变得一团糟。)

顺便说一下，理解系统并不等于理解问题，这有点像Steve Martin所讲的那个变成百万富翁的愚蠢方法：“首先，得到100万……”（此处的引用已得到Steve Martin的许可。）当然，你现在还不理解问题，但如果你想要查明系统为什么不工作的话，必须先理解它的工作原理。

3.1 阅读手册

理解系统的基本方法就是阅读手册。我父亲的观点也不尽然，我们应该首先阅读手册，而不是等到所有办法都不管用之后才去读它。当你买来一件东西时，手册告诉你怎么操作它，以及它是用来干什么的。我们需要一页一页读完手册并理解它，以使用它来完成我们需要做的工作。有时你会发现它不能做你需要的工作，因为你买错东西了。因此，我刚才的观点又被推翻了——在买回一块没用的废物之前，先阅读手册。

如果你的除草机不能发动，读手册可能会提醒你在拉紧除草绳之前先按一下“primer bulb”按钮。我们家有一台除草机，它开动的时候会很热，以至于会把除草绳烧化，这时就无法再工作了。而这是因为使用除草机的小伙子没有阅读手册中给除草头上润滑油的那部分内容。我读了手册之后发现了这一点。如果你做的砂锅豆腐很难吃，就要重新看一遍菜谱了。（实际上，在这种情况下看菜谱可能也不会有帮助，你最好读一下“Chu-Quik Chou House”^②的外卖菜单。）

如果你是一位工程师，正在调试自己公司的产品，那么你需要读一读内部手册。工程师们设计它是用来做什么的？读一下功能说明以及所有的设计规范，研究一下图表、时序图和状态机。分析它们的代码，还要读一下注释。（是的，读一下注释，这非常重要。）一定要检查产品的设计。查明构建它的工程师们打算用它来做什么（除了用它来赚钱买辆宝马车以外）。

① 墨菲定律 (Murphy's Law)，事情如果有变坏的可能，不管这种可能性有多大，它总会发生。

② 一家中餐馆的名字。

注意，手册上的信息也不可全信。手册（以及那些只想着赚钱买宝马车的工程师们）可能也是错的，很多难以发现的bug就出现在这里。但你仍需要了解他们的想法，哪怕其中有些信息是很难接受的。

有时，这些信息正是你需要看的：



案例故事 我们正在调试一个用汇编语言编写的嵌入式固件程序。这意味着我们必须直接分析微处理器的寄存器。我们发现寄存器B被破坏了，进一步缩小范围后发现问题是由于调用了子例程引起的。当我们查看该子例程的源代码时，发现在代码开头有以下注释：`/* Caution—this subroutine clobbers the B register */`（注意，这个子例程会破坏寄存器B）。我们修改了这个子例程，把B寄存器独立出来，于是bug被修复了。（当然，编写这段代码的程序员直接修复这个问题比输入这句注释还容易，但不管怎样他至少记录了这个问题。）

在另一家公司，我们检查过一个看起来是由于顺序错误导致的问题。我们查看了源代码，这些代码是我先前编写的，我告诉同事说我曾担心过会出这样的问题。于是，我们在代码中搜索“bug”，发现在两个函数上面有这样一条注释：`/* DJA—Bug here? Maybe should call these in reverse order? */`（DJA——这里是否有bug？或许应该把它们的调用顺序颠倒一下）。事实上，改变调用顺序后确实修复了bug。■

理解了你自己的系统后，还会获得一个额外的好处。当你找到bug时，必须在不破坏其他地方的前提下修复它们。理解系统行为是不破坏系统的第一步。

3.2 逐字逐句阅读整个手册

人们在调试的时候，通常都不会彻底地阅读系统手册。他们采取跳读的方式，查看他们

认为重要的一些章节，但问题的线索可能就隐藏在被略过的那些章节中。是的，我就是凌晨1点的时候找到了线索，最后终于发现了进料阀控制器的bug。

编程指南和API可能非常厚，但你必须深入挖掘它，查找你认为有问题的函数。图表部分可以忽略，它们会干扰你。但数据表要仔细查看，可能表中不起眼儿的一行指定了一个模糊的时序参数，而它就是问题所在。



案例故事 我们构建了几个版本的通信板 (communications board)，有些板上安装了3个电话电路，有些板上则安装了4个。3个电路的系统在现场用了一段时间后，效果良好，这时我们在实验场地引入了有4个电路的系统。这些系统所做的内部测试较少，因为两种系统所使用的电路是相同的，只是多安装了一个电路而已。

但是，在实验场地上，4个电路的系统在高温下发生了故障。我们很快在实验室中再现了故障，并发现主处理器崩溃了。这通常是因为程序内存被破坏，于是我们运行测试，结果发现内存在读回数据时发生错误。令我们感到奇怪的是，为什么安装了同样的3个电路的板子却没有出现这个问题。

硬件设计师查看了几块通信板，注意到4个电路的通信板使用了另一种不同牌子的内存芯片（它们的生产批号不同）。他查看了规格说明。两种芯片都符合工程标准，速度也都很快，而且它们读写的时序也相同。这位设计者考虑到这些规格是正确的，只是他是从处理器时序的角度来考虑的。

我把整个数据表读了一遍。我发现发生故障的内存有一个不同的规格，它指定了两次访问之间需要等待的时间。这段时间很短，看起来也不怎么重要，而且两种内存的等待时间的差别也不大，但处理器时序设计没有考虑它，而且也不满足这两种芯片的规格。因此，它在速度较慢的芯片上会频繁发生故障，而在较快的芯片上发生故障也是迟早的事情。

我们通过稍微减慢处理器的速度解决了这个问题，而且修订后的设计使用了更快的内存，我们还仔细检查了数据表的每一行。■

应用说明和实现指南提供了丰富的信息，它们不仅描述了系统是如何工作的，而且专门给出了先前已发生过的问题。常见错误的警告具有难以置信的价值（即使你犯的错误都很不常见）。从供应商的站点获取最新的文档，并阅读网站上所列出的最近一星期发现的常见错误。

参考设计和样本程序给出了产品的一种使用方式，有时这些就是能获得的全部文档了。但是，在使用这些设计时一定要注意，创建它们的人往往只了解他们的产品，而没有遵循好的设计实践，或者不是为真实应用而设计的（最常见的缺点是不能进行错误恢复）。不要照搬这些设计，如果你没有在开始的时候发现bug，那么将来也会发现。此外，即使是最好的参考设计可能也不会完全符合应用程序的特定需求，而不符合的地方可能就是出问题的地方。当我照搬了朋友的微处理器设计时，就发生了问题，因为他的设计无法处理中断。

3.3 知道什么是正常的

当你检查系统时，必须知道系统的正常工作状态。如果你不知道低位字节首先由使用了Intel芯片的PC程序来处理，那么你会认为所有长字（longword）都是随意处理的。如果你不知道缓存是干什么的，就会非常奇怪有些数据为什么没有马上写入内存。如果你不了解三态（tri-state）数据总线的工作原理，你将会认为它们可能是主板上的故障信号。如果你从未听说过电锯，你可能会认为那个发出讨厌的嗡嗡声的东西一定是出了什么毛病。知道什么是正常的可以帮助你注意到什么是不正常的。

你必须掌握一些你所工作的技术领域的基础知识。如果我不知道时钟选通脉冲和地址线是做什么的，那么即使我读了手册之后也无法理解中断问题。本书中几乎所有的（即使不是全部的话）示例都假定人们已经掌握了系统工具原理的一些基本知识。（如果我在前面使你误认为读完本书就可以调试任何技术领域的bug，那么请恕我无心之过。如果你是一位游戏

编程人员，最好不要去管核电厂的调试。如果你不是医生，那么就不要试图诊断你手臂上的灰绿色斑点是什么。如果你是一位政客，那么就不要介入任何有关bug的事情。)



案例故事 与我共事的一位软件工程师在一次调试策略会议结束后，边摇头边走出来。他们讨论的bug是微处理器的崩溃问题。那只能勉强算是一个微处理器，因为它没有操作系统，没有虚拟内存，也没有任何其他的东西，他们知道它崩溃的唯一线索就是它无法重新设置一个监视定时器，因此导致定时器最终超时。软件工程师们正在试图查明它在哪里发生了崩溃。有一位硬件工程师建议他们在崩溃之前设置一个断点，当到达该断点时，查看一下发生了什么情况。显然，他并没有真正理解起因和结果，如果知道在哪里设置断点，那么就已经找到问题了。

这就是软件人员和硬件人员在试图调试对方的工作时总会惹恼对方的原因。■

缺乏基础知识解释了为什么很多人找不到自己家用电脑的毛病：他们只是没有理解计算机的基本原理。如果你无法学习那些需要掌握的知识，可以遵照调试规则8，向有专业知识或经验的人请教。十几岁的孩子过马路是没问题的，但你要想让他帮你处理录像机上总是闪烁不停的“12:00”，还是等他大一些再说吧。

3.4 知道工作流程

当你尝试寻找bug时，必须知道要查找的路线。开始时，你需要猜测在哪里把系统分隔开，以便隔离问题，这种猜测完全取决于你对系统功能划分的了解。你至少要大体上知道所有的模块和接口都是做什么的。如果你的烤箱把面包烤焦了，你需要知道哪个黑色旋钮是用来控制烤制时间的。

你应该知道系统中的所有API和通信接口都是用来交换什么数据的。还应该知道每个模块或程序如何处理它们通过这些接口收发的数据。如果代码是高度模块化或面向对象的，那

么接口将很简单，模块也有良好的定义。观察接口就很容易解释你看到的东西是否正确。

当系统有一些部分是“黑盒子”时，这意味着你不知道它内部有什么，但应该知道它们如何与其他部分交互，这至少可以帮助判断问题是在内部还是外部。如果问题发生在黑盒子内部，你必须更换盒子，但如果问题出在外部，就可以修复它了。在面包烤焦的例子中，你可以控制黑色旋钮，试着把它调小一些。如果烤制时间并未缩短，说明烤箱内部出问题了，那么就扔掉它，再买个新的（也可以拆开修理一下，修不好再换个新的）。

假如你在开车时听到“嗒嗒嗒”的声音，你开得越快，声音也越急。这可能是由于轮胎面上嵌了块小石头（很好修理），也可能是由于发动机出了问题（很难修理）。当汽车高速行驶时，发动机和轮胎是同步加速和减速的。但如果你知道发动机是通过传动轴与轮胎连接的，就应知道，如果调低档位，那么在保持轮胎转速不变的情况下发动机将转得更快。于是你可以调低档位，如果声音的频率保持不变，可以推断问题出在轮胎上，于是你在路边停车，发现轮胎面上嵌了块小石头。你只需调低档位来检查传动轴，而节省了去修理店的高昂费用。

3.5 了解你的工具

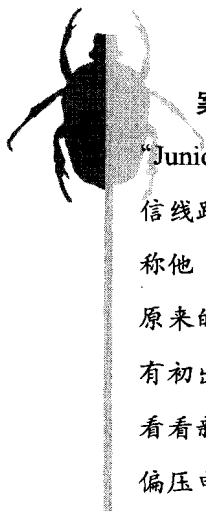
调试工具是用来观察系统的眼和耳，你必须选择正确的工具，正确地使用工具，并正确地解释得到的结果。（如果把温度计不正确的一头放到你的舌下，是不会测出正确体温的。）很多工具提供了非常强大的功能，但只有精通它们的用户才了解。你越是精通工具，就越容易查明系统中发生了什么事情。要花时间学习与工具有关的一切，通常，查明系统行为的关键（参见规则3）是你的调试器设置得怎样，或者是否正确地触发了分析器。

我们还必须了解工具的局限性。走查源代码可以显示逻辑错误，但无法显示时序和多线程问题；剖析工具可以暴露出时序问题，但显示不出逻辑错误。模拟示波器（analog scope）可以看到噪声，但无法存储太多数据；数字逻辑分析器可以捕获大量的数据，但看不到噪声。普通的体温计无法告诉你太妃糖是不是太热了，而糖果温度计的精确度甚至不足以测出你是否发烧。

那位建议在崩溃之前设置一个断点的硬件人员并不知道断点的局限性（或者他有某种奇妙的时间旅行技术）。软件人员最后采取的方法是在系统上接入一个逻辑分析器，用来记录微处理器的地址和数据总线的变化痕迹，同时把监视定时器设置成一个非常短的时间。这样，当定时器超时的时候，地址和数据总线的变化痕迹也就保存下来了。他知道必须把发生的事情记录下来，因为直到定时器超时后，他才能知道处理器已崩溃。把定时器缩短的原因是他知道分析器无法记录太多的信息，达到它的记忆量后，旧信息就会被替换掉。通过查看跟踪记录，可以看到程序在什么地方发生了问题。

你还必须了解开发工具。这当然包括用来编写软件的语言，如果你不知道C语言中的“+ =”操作符是做什么的，代码的某个地方就会出问题。但除此之外，你还需要了解一些更微妙的知识：编译器和链接器在把代码发给机器之前会进行什么处理。数据是如何匹配的，引用是如何处理的，以及内存是如何分配的，这都将对你的程序产生影响，而这些通过源程序并不能明显看出来。硬件工程师必须知道如何按照高级芯片设计语言中的定义来设计芯片上的寄存器和门。

3.6 查阅手册



案例故事 在一家大型计算机制造企业工作的一位初级工程师（我们称他“Junior”）正在设计一个系统，系统中使用一种1489A芯片。这种芯片接收来自通信线路的信号，类似于计算机与调制解调器之间的连接。一位高级工程师（我们称他“Kneejerk”）看到他的设计后指出：“哦，你不应该使用1489A，而应该使用原来的1489。” Junior问为什么，Kneejerk回答说：“因为1489A会变得过热。”像所有初出茅庐的年轻人一样，Junior怀疑所有长者的话，于是他决定了解一下电路，看看新版本的芯片为什么会发热。他发现这两种芯片之间的唯一区别是一个内部偏压电阻的值不同，新的芯片在电路中具有更强的抗噪声干扰性。现在，1489A中

的这个电阻较小，因此，如果对它施加过大的电压，它就会发热。但是在目前这个设计中，电阻的连接方式并不会对它施加过大的电压，当然也就不足以令它过热。理解了这个电路后，Junior没有采纳Kneejerk的建议，仍旧使用了1489A。事实也证明它没有过热。

几个月后，Junior开始检查小组先前设计的电路。当他按照原来的1489的图示把示波器探针放在输入引脚上时，引脚的读数看起来是错误的。他查了查已经被他翻得很旧的数据手册，确信是引脚线接错了——他们把输入接到了偏压电阻上，而没有接到输入引脚上。偏压电阻的引脚通常是不接线的，但如果把输入接到它的引脚上，它也能工作，但这样电路就完全失去了抗噪声干扰的能力。这个错误连接还使得电流绕过了输入电阻，导致大量电流流经内部偏压电阻。事实上，Junior注意到了一个有趣的现象——元件会发热，而且如果使用1489A的话发热量会更大。■

在这个故事中，有两个地方违反了“理解系统”规则。首先，原来的工程师在设计电路时没有查阅引脚的编号以确保连接正确。随后，Kneejerk使问题进一步复杂化，他没有通过理解电路来查明为什么新元件会发热。结果，这个团队所设计的电路使用了一种已经过时且很难找到的元件，这种元件散发大量热量而又丝毫没有抗噪声干扰的能力。除此之外，其他的工作完成得都很出色。

相反，Junior没有相信示意图上的引脚连线，他查阅了数据手册中的正确连接方法，因此知道了元件为什么会发热。

不要猜测，而要查阅手册。芯片制造商或软件工具的开发人员已经把详细信息写到手册中，而你不应该盲目相信自己的记忆。养成良好的查阅习惯，无论是芯片的引脚连接，还是函数的参数，甚至是函数名称。我们要学爱因斯坦，他从不记忆自己的电话号码，“干嘛要费事记它呢？它不就在电话簿中吗？”

如果你单凭猜测去观察芯片上的信号，那么当你看到错误的信号时，可能会把它当成正确的。如果你假定函数的参数调用顺序是正确的，那么可能会像原来的设计者那样把问题忽略过去了。这会导致信息的混淆，甚至再一次确认了错误的信息。不要把调试的时间浪费在那些错误的信息上。

最后提醒你一点，如果在深夜2点你家的地下室因为水管坏了而被淹没，当你修不好而决定打电话求助时，不要乱猜电话号码，去查电话簿吧。

3.7 小结

理解系统

这是第一条规则，因为它是最重要的。

- **阅读手册。**它会告诉你在使用除草机时，要在除草头上涂润滑油，这样除草绳就不会被烧化。
- **仔细阅读每个细节。**有关微处理器如何处理中断的详细信息就隐藏在数据手册的第37页。
- **掌握基础知识。**电锯本来就会发出很大的噪声。
- **了解工作流程。**引擎的转速可能与轮胎的转速不同，这是由传动轴造成的。
- **了解工具。**弄清楚体温计的哪一端才是用来测量体温的，弄清楚Glitch-O-Matic逻辑分析器的强大功能是如何使用的。
- **查阅细节。**连爱因斯坦都会去查阅细节，而Kneejerk却盲目相信自己的记忆力。

第4章

制造失败

4

“什么也比不上直接取得的证据来得重要。”

——福尔摩斯，《血字的研究》



案例故事 1975年的一天深夜，我独自一人在实验室里调试一款电视网球游戏的一个问题。这是第一批家庭电视游戏当中的一个，它是由当地的一位企业家出资，在MIT创新研发中心开发出来的。游戏中有一面用来练习击球的墙，我要解决的bug就发生在球从墙上反弹回来的那个时刻，但它只是偶尔才发生。我把示波器调好（这台示波器有点像老的科幻电影中的那些机器，它在一个小的圆形屏幕上画出波形曲线），准备观察bug，但发现我很难在示波器上观察到bug。因为如果我把球速调得很慢，它隔好几秒钟才能从墙上弹回一次，但如果把球速调得太快，又不得不把所有注意力都放在击球上。如果漏球了，就得等到下一次发球再去观察。这并不是一种高效的调试方式，我不由想到：“算了，这只是一个游戏而已。”但我很快打消了这个念头，我想如果能让游戏自己玩起来，那我就能集中注意力观察示波器了。

我发现击球板的上下位置和球在各个方向（上下左右）的位置都是用电压表示的（参见图4-1）。（如果你对硬件知识不够了解，那么我来解释一下什么是电压。电压就像是一个容器中的水平面，我们通过注水或排水来改变水面的高度。电压与这类似，只是它没有水，而是电荷。）

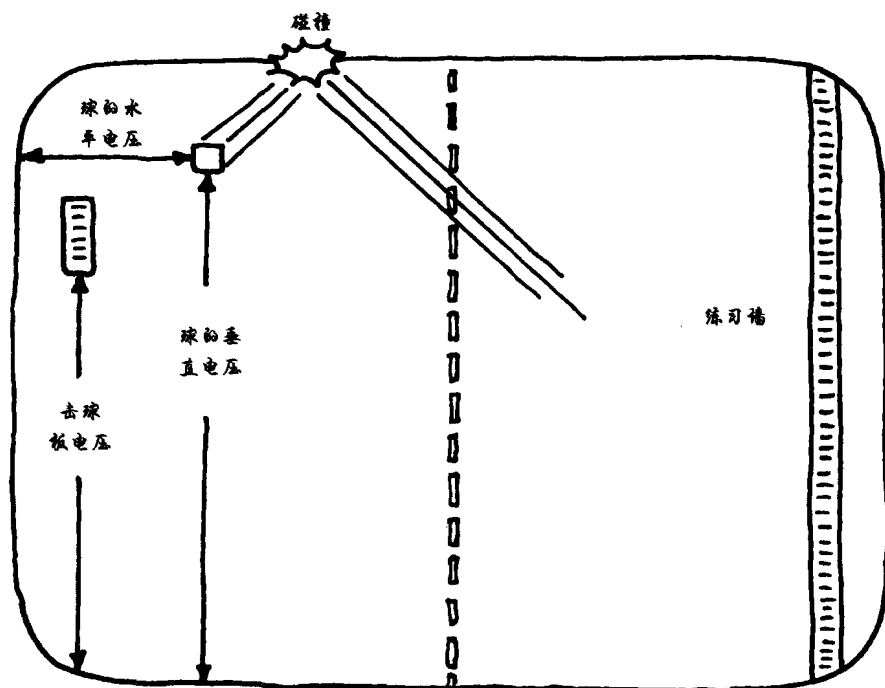


图4-1 电视网球游戏

我意识到，如果把击球板的电压连接到控制球上下位置的电压，而不是连接到手动操纵杆，那么击球板就能跟随球一起移动。这样，当球被弹回来时，击球板就能找准高度把球击回去。我按照这种思路把电路连好，击球板打得非常好！游戏终于可以自己玩起来了，我可以集中精力观察示波器，很快就发现并解决了问题。■

即使是在深夜，我也能很容易在示波器上看到球在反弹回来时是如何发生错误的。关键是在发生失败的时候要看到它。这是很多调试的典型问题——你看不到问题是如何发生的，因为你方便观察的时候，它并没有发生。这并不是说它只发生在深夜（即使你最终是在深夜时发现了它），它可能是每7次中只发生1次，或者是Charlie测试它的时候，碰巧发生了一次。

如果Charlie现在正在我的公司工作，你问他：“当你发现一个故障时该怎么办？”他会

回答说：“试着让它再次发生。”(Charlie是一位训练有素的调试人员)。这样做有3个原因。

- **可以观察它。**要观察错误(下一节将更详细地讨论这个问题),就必须使它发生。我们必须尽可能有规律地制造失败。在前面讲的电视游戏的例子中,当问题发生时我可以集中注意力观察示波器(虽然当时我已经很疲倦)。
- **可以专心查找原因。**准确地知道问题在什么条件下会发生,有助于集中精力查找原因(但是请注意,有时这会产生误导,例如,“烤箱只有在你把面包放进去的时候才会把面包烤焦,因此问题就出在面包上。”这个问题后文也会详细讨论。)
- **可以判断是否已修复问题。**当你认为已经修复了问题时,如何才能确信它确实已被修复呢?那就是明确知道问题是如何发生的。当问题没有修复时,如果你执行X操作,失败率为100%;在修复问题后,再执行X操作,如果失败率为0,那么你知道bug确实已被修复。(我这么说并不多余。很多时候,开发人员在修复bug时会修改软件,然后在一个与当初发现bug的不同条件下测试新软件。软件当然能运行,即使他在代码中输入一行打油诗,而他也高兴地回家了。然而,几星期后,在测试过程中,或者更糟,在客户现场,软件再次失败。后文将讨论更多这方面的内容。)



案例故事 我最近买了一辆四轮驱动的新车,整个夏天我开着它都没发现什么问题。当天气开始变凉时(在新罕布什尔州,一般到9月天气才开始凉爽),我注意到,如果时速变为25至30英里/小时之间,头几分钟内,车子后部会发出“嗞嗞”的噪声。加速或减速,噪声会消失。10分钟后,噪声也会消失。如果温度高于25°F^①,也不会发出噪声。

我把车开到经销商那里进行常规保养,我告诉他们在早上天气较凉的时候先把我的车子开出去,听听声音。他们却没有照我说的去做,直到上午11点才开始看我的车,这时温度已经达到37°F了,车当然没有发出噪声。他们卸下轮胎,检

① 华氏温度 T_1 与摄氏温度 T_2 的换算关系为 $T_2 = \frac{5}{9}(T_1 - 32)$ 。——编者注

查了刹车，没有发现问题（当然不会发现）。他们没有制造失败，因此无法找到问题。（我一直想等一个较冷的天气再次把车开到那里，但我们遇到了历史上最暖的一个冬天。这就是墨菲定律。我想在车子过保修期之前，总会有一个冷天气吧。）■

4.1 制造失败

但如何才能让它失败呢？一种简单的方法是进行一次内部预演，还有一种同样有效的方法是演示给未来的投资者。如果碰巧没有客户或投资者在现场，那么你就必须设法正常使用，并观察它是如何出错的。当然，测试本来就应该是这样的，但这里的重要之处是在错误第一次出现之后，能够使它再现。通常，认真记录测试过程可以作为补充，但你必须认识到仅有一次错误是不够的。当一个3岁的孩子看到她的父亲从梯子上摔下来，失手打翻了油漆桶，弄得满头满脸都是油漆时，她会拍手大叫：“再来一次！”我们应该向这个孩子学习。

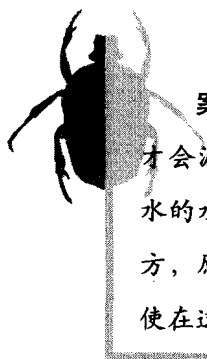
仔细观察你做了什么，然后再做一次，并且记下你做的每个步骤。然后，按照你自己所写的步骤去做，确定这样做确实导致了错误。（把油漆洒到头上的父亲就不要再这样做了。实际上，在有些情况下，令设备发生错误具有一定的破坏性，或者代价很大，这时每次都使设备发生同样的错误就不是一种好办法。为了控制损失，必须改变一些地方，但我们应该尽量少改动原来的系统和顺序。）

4.2 从头开始

通常，所需的步骤很短，也很少。例如，单击这个图标，就会出现拼写错误的消息。有时，虽然步骤很简单，但需要进行很多设置。例如，重启计算机，运行这5个程序，然后单击这个图标，出现了拼写错误的消息。由于bug可能仅仅在机器的某个复杂状态下才会出现，因此必须仔细注意机器在执行这些步骤时的状态。（如果你告诉修车工每当在寒冷的天气里开车时，车窗就会被冻住而打不开，他应该会猜出你每天早上都洗车。）试着从一个已知的状态开始，例如刚刚重启的计算机，或者是你一早步入车库时汽车的状态。

4.3 引发失败

在调试故障的时候，如果需要手工执行很多步骤，那么使这个过程自动化会很有帮助。前面的电视游戏就是这种情况，我需要同时玩游戏和调试，自动的击球板替我完成了玩游戏的任务。（很遗憾我不能把调试工作自动化，而由我来玩游戏。）在很多情况下，只有在重复很多次后，错误才会出现，因此我们希望在夜间运行自动测试工具。软件很愿意整夜工作，你连比萨饼都不用为它买一块。



案例故事 我房屋的一扇窗户漏雨，但只是在倾盆大雨而且刮东南风的时候才会漏。我想赶在下一次暴风雨到来之前把它修好，于是我架了把梯子，拿着喷水的水管向窗户喷水，看看它到底是怎么漏雨的。这使我准确地看到了漏雨的地方，原来是墙缝那里有一道空隙，我把这个空隙塞满，并再次喷水检查，发现即使在这么高的水压下，它也不再漏水了。■

过敏症专科医师会用各种已知状态的过敏原对病人皮肤进行实验，看看哪些会引起反应。牙医会在病人口腔中喷冷空气，以便发现对冷物敏感的牙齿。（另外，牙医这么做也会觉得很有趣。）州警在检查醉酒驾车时会让司机走直线，身体向后仰，让司机用手摸自己的鼻子，以及倒背字母表等事情，以确定他是否酒后驾车。（这比让司机开一段车，看看他是否能够在高速公路的正确一侧行驶要安全得多。）

如果你的Web浏览器有时会打开错误的网页，就应该通过设置浏览器来让它自动请求页面，以便查找错误。如果网络软件在高流量下发生错误，应该运行一个网络加载工具来模拟负载，这样就可以引发错误。

4.4 不要模拟失败

引发失败（正确）和模拟失败（错误）这二者之间存在着非常大的差别。在前面的例子

中，我建议模拟网络负载，而不是模拟失败机理本身。正确的方法是模拟那些导致失败发生的条件。但是，不要试图模拟失败机理本身。

你可能会问：“为什么要这么做呢？”如果有一个间歇性的bug，你可能猜测它是由某个底层机制引起的，于是构建一个配置来模拟该底层机制，以为这样就可以反复观察到bug。或者，你可能在发生bug的现场之外来模拟它，方法就是在你自己的实验室中建立一套等价的系统。以上两种方法都是在试图模拟失败，即重新制造它，只是采用了另一种不同的方式或系统。

如果你猜测失败机理，模拟往往不会成功。原因通常有两个，要么你的猜测是错误的，要么测试改变了条件，模拟的系统可以正确工作，或者更糟，发生新的错误，因而分散了你对正在查找的问题的注意力。举个例子，如果你的字处理程序（这是你要拿来与Microsoft 媲美的程序）在保存文件的时候总是删除图片，你可能猜测这个问题是在写文件时发生的。于是你编写了一个测试程序，它不断地向磁盘写文件，最后操作系统死机了。这样，你得到的结论就是Windows的速度太慢了，因此你的解决办法就是开发一个用于淘汰Microsoft Windows的操作系统。

查找现有bug已经够我们忙的了，不要再制造新的bug。利用工具来观察发生了什么错误（参见规则3），但不要改变机理，因为正是这个机理导致了错误。在字处理器的例子中，不要改变文件写入磁盘的方式，而应该自动生成按键，然后观察什么被写入了磁盘。

在类似的系统上再现bug是一种较为有用的方法，但有一些限制条件。如果一个bug可以在多个系统上再现，那么我们就可以认为它是设计bug，因为它并不是在一种系统上以某种特定的方式出现。如果在某些配置下能够再现它，而在另一些配置下无法再现，那么这就帮助我们缩小了查找范围。但是，如果无法快速地再现它，那么不要为了使它出现而改变你的模拟环境。这样会产生新的配置，而不是原来发生错误的那个配置了。无论一个系统在哪种常规配置下发生故障（即使是间歇性故障），都要在该系统上使用该配置来查找问题。

一种典型的情况是在客户现场的某种综合条件下发生问题——软件在某台机器上驱动某个特定周边设备时失败。通过在你自己的现场建立一个相同的配置，或许可以模拟失败。

但如果没有相同的设备或条件，因而无法模拟失败，那么你可能会试图模拟设备或发明新的测试程序。不要这样做，而应该克服困难，要么把设备运到你自己的场地，让工程师来调试，要么派工程师（用出租车载上插装工具）到客户现场进行调试。如果客户现场位于阿鲁巴岛^①，那么运输设备是不太现实的。顺便问一下，贵公司最近是否雇到了既擅长写作又具有丰富调试经验的好手^②？

注意，不要用一个看似完全相同（而实际上不同）的环境来代替并希望看到相同的错误。当我修理漏雨的窗户时，如果我假设窗户的设计有问题，那么我可能会用另一扇“完全相同”的窗户来做实验。这样就不会发现墙边的缝隙了，因为这条缝隙是那扇漏雨的窗户才有的。

问：有多少工程师参与修复电灯泡？

答：一个也没有。他们都说：“我不能让失败再现，因为我办公室的灯泡很正常。”

记住，这并不意味着不能用自动化过程来引发失败，也不意味着在这个过程中不能采用一些起到放大效果的措施。自动测试能够使间歇性的问题更快发生，例如电视游戏的例子。放大效果可以使得细微的问题更明显，例如在修窗户的例子中，我可以用喷水管来找到漏雨的窗户，而不用等待偶尔才有的暴风雨来检查。这两种技术都有助于引发失败，而不是模拟失败的机理。所做的改变应该是一些高层次的改变，只影响错误发生的频率，而不影响错误的发生方式。

此外，还要注意不要画蛇添足，引发新的问题。不要因为假设芯片的问题是由于热量引起的，就用热风枪来给芯片加热以模拟错误，这样只会把芯片烧化，然后你会误认为bug完全就是电路板上那堆被烧化的塑料。如果我用消防用的水龙来检查漏雨问题，可能会断定问题显然就是出在被击碎的窗子上。

4.5 如何处理间歇性 bug

当故障只是偶尔发生时，用“制造失败”这种方法来调试就困难得多。很多棘手的问题

① 阿鲁巴岛，Aruba，安的列斯群岛中的一个岛。

② 意思是：“我就是这样的一个调试高手，不如雇我吧。”

都是间歇性的，这就是不能总是应用这条规则的原因——它很难应用。你可能已经制造出了一次失败，但是当你用同样的方式再次尝试时，问题仍然间歇性出现，可能5次、10次甚至几百次中才会出现一次。

关键问题在于你并没有完全弄清楚失败是如何发生的。你知道你做了什么，但并不知道完整的、准确的条件。还有其他你没注意到或无法控制的因素，例如初始条件、输入数据、时序、外部过程、电子噪声、温度、振动、网络流量、月相（phase of the moon）以及测试者是否清醒，等等。如果你能够控制所有这些条件，那么就可以一直使错误发生。当然，有时你无法控制这些条件，我们将在下一节讨论这个问题。

那么，如何控制这些条件呢？首先，查明它们。在软件中，查找未初始化的数据（它们总是带来麻烦）、随机数据输入、时序误差、多线程同步和外部设备（例如电话网络，或者看看是不是有6 000个孩子在同时点击你的站点）。在硬件中，查找噪声、振动、温度、时序和部件误差（类型或供应商）。在我那部四驱车的例子中，如果我没有注意到温度和车速，那么问题看起来就是间歇性的。



案例故事 一家老式的大型计算机中心间歇性地在下午发生崩溃，虽然它几乎是在同一时间崩溃，但并不总是在程序的同一个位置崩溃。人们最后发现崩溃时间与下午三点人们喝咖啡短暂休息的时间吻合。这时自助餐厅中的所有自动售货机都在同时操作，导致硬件的电力供应不足。■

一旦想到了有哪些条件可能影响你的系统，必须大量尝试与这些条件相符的各种形式。初始化这些条件，并按照一种已知模式把这些条件作为你的问题软件的输入。尝试控制时序，然后改变它，看看系统在某个特殊设置下是否会失败。对有问题的电路板进行多种测试，例如振动、加热、制冷、注入噪声以及改变时钟速度和电压，直到失败频率出现变化。

有时，你会发现当你控制某个条件的时候，问题消失了。这时你就发现了是什么（随机

产生的)条件导致了失败。当然,如果发生这种情况,你需要尝试该条件下的每个可能的值,直到找到导致系统失败的那个值。如果一个随机的数据输入模式间歇性地导致系统失败,而固定的数据模式不会导致失败,那么就要尝试每个可能的数据输入模式。

有时,你会发现有些条件是无法控制的,但可以增加它的随机性。例如,振动一块电路板或注入噪声。如果故障是由某个低概率的事件(例如噪声峰值)引起的,那么问题就是间歇性的,这时可以通过增加条件(噪声)的随机性,来提高这些事件发生的频率。这样,错误就会更频繁地发生。这可能是我们所能采取的最好办法了,它能提供很大的帮助,可以告诉我们失败是由什么条件引起的,也能使我们更容易看到失败。但有一点要注意,在对条件进行放大操作的时候,不要引起新的错误。如果一块电路板有一个对温度很敏感的错误,而你却决定振动它,以至于所有芯片都松动了,那么将会有更多错误,而这些错误与原来的错误毫无关系。

有时,你的所有尝试都不会有任何区别,你又回到起点,问题仍然间歇性地发生。

4.6 如果做了所有尝试之后问题仍然间歇性发生

记住,我们之所以制造失败,是出于3个目的:一是观察错误,二是查找线索,三是确认是否已修复。下面就讨论一下当问题看起来“有它自己的思维”时,应该如何完成这3个目标。记住,问题是没有自己的思维的,失败肯定有原因,你一定能够找到它。它只是“巧妙地”隐藏在你尚未发现的大量随机因素背后。

4.6.1 仔细观察失败

你必须能够看到失败。如果它不是每次都发生,那么就必须忽略掉不发生的时候,而在它每次发生时观察它。关键是在每次运行的时候捕捉相关信息,以便在发生失败之后查看这些数据。方法就是让系统在运行的时候尽可能多地输出信息,并把它们记录到“调试日志”文件中。

通过查看捕获到的信息,很容易把正常运行和错误运行放在一起进行比较(参见规则5)。

如果你捕获到了正确的信息，就能够看到正常运行与错误运行之间的区别。仔细观察只在错误运行中才发生的那些事情。这是实际开始调试时需要注意的地方。

尽管失败是间歇性的，但这样就能识别并捕获发生错误的条件，然后对其进行分析，就像它们每一次都发生一样。



案例故事 我们的视频会议系统发生了一个间歇性的错误，在我们的部门呼叫另一家供应商的部门的时候。大约每5次呼叫中，就会有一次呼叫发生错误，导致对方的系统关闭视频，只留下音频电话呼叫。

我们无法调查对方的系统，但可以记录自己的调试日志。我们捕获了两次连续呼叫的数据，前一次是正确的呼叫，后一次则发生了错误。在失败的呼叫日志中，有一条消息显示我们发送了一条异常的命令。我们检查了前一次正常的呼叫以及其他正确的呼叫，发现所有正确呼叫中都没有这条消息。我们记录了更多日志，直到再次发生一个错误的呼叫，发现日志中又出现了那条异常命令的消息，这使我们非常确信错误就发生在这里。

后来我们查明问题出在内存缓冲器上，它里面装满了前一次呼叫的命令，而在新呼叫开始发送命令之前可能没有清空它们。如果缓冲器正确清空，则一切就会正常。如果没有清空，我们发现的那条异常命令就会在呼叫开始的时候被发送出去，对方的机器就会错误地解释它并进入只有音频的工作模式。■

我们之所以能够看到这个系统错误，就是因为跟踪了每次呼叫，获得了足够多的信息。虽然错误是间歇性的，但日志显示了它每次发生时的情况。

4.6.2 不要盲目相信统计数据

制造失败的第二个目的是获得问题发生的线索。当发生一个间歇性问题时，你可以注意