Atitit.软件调试原理与概论attilax总结

第1章预防药

第2章调试技巧

第3章理解问题

第4章调试器内部机制

第5章优化：内存占用

第6章优化：CPU周期

第7章最后的赠言

[软件驱魔：调试和优化遗](http://product.dangdang.com/23471929.html" \l "ddclick_reco_product_alsobuy" \o "软件驱魔：调试和优化遗留代码的艺术（遗留代码调试和优化领域经典之作，系统讲解有效的调试技术和实用的优化策略）  " \t "http://product.dangdang.com/_blank)

(美)Bill Blunden　著 等

  

当当自营

商品详情

商品评论(49条)

　　本书是遗留代码调试和优化领域的代表性著作，是作者10多年来在软件bug中“驱魔”经验的结晶，Amazon五星评论。不仅从实用性角度深入、系统地讲解了调试和优化遗留代码的方法、技术和最佳实践，而且从源头上阐述如何避免掉进维护遗留代码的泥潭，编写出易于维护，甚至不需要维护的高质量代码。  
　　本书共分7章：第1章介绍用于构建易于修改和调试的软件的相关技术；第2章讲解如何定位和消除软件bug；第3章提供理解问题以及程序的有用方法；第4章讲述调试器在计算机内部的工作原理；第5章深入探讨优化内存的相关技术和方法；第6章则探讨如何从CPU运行效率的角度优化代码的经验和技术；第7章结合社会和环境因素给出一些宝贵的建议。

作者简介

　　BillBlunden拥有康奈尔大学物理学学士学位和凯斯西储大学运筹学硕士学位。资深系统维护专家，从事相关工作10余年，对系统软件的维护、遗留代码的调试和优化有非常深入的研究。活跃于计算机安全类社区，常与计算机安全领域多名世界级安全专家交流探讨。

目录

译者序  
前言  
第1章预防药  
1.1?核心问题  
1.1.1上市时间的压力  
1.1.2不确定的规格说明  
1.1.3以功能特性换时间  
1.1.4写下来  
1.1.5复杂性  
1.2防御性编程  
1.2.1内聚和耦合  
1.2.2错误输入检查  
1.2.3数据范围  
1.2.4日志  
1.2.5文档  
1.2.6为改变而设计  
1.2.7增量精炼  
1.3单元测试  
1.3.1自动化测试的动机  
1.3.2实现框架的步骤  
1.3.3框架扩展  
1.4工具的配置  
1.4.1使用编译器警告  
1.4.2发行版本的设置  
1.5机器相关性  
1.5.1字节序  
1.5.2内存对齐  
1.5.3数据类型大小  
1.5.4虚拟机的好处  
1.6小结  
1.6.1底线：为什么出现bug  
1.6.2改进清单：bug主动预防  
第2章调试技巧  
2.1初始步骤  
2.1.1复现问题  
2.1.2无法复现的问题  
2.1.3验证bug是真实存在的  
2.2消除缺陷：快速修复  
2.2.1检查近期改动  
2.2.2使用跟踪信息  
2.2.3似曾相识  
2.2.4明确何时放弃  
2.3消除缺陷：科学方法论  
2.3.1一般步骤  
2.3.2定位问题：增量集成法  
2.3.3定位问题：二分法  
2.3.4理解问题  
2.3.5防范失误  
2.3.6诊断工具  
2.3.7基础调试操作  
2.4保留记录  
2.4.1个人记录  
2.4.2协同开发下的记录  
2.5小结  
第3章理解问题  
3.1知识是如何丢失的  
3.1.1竞争  
3.1.2人员流失  
3.1.3升职  
3.2难懂的代码  
3.2.1设计问题  
3.2.2混淆  
3.2.3误导性的代码  
3.3反向工程  
3.3.1通用策略  
3.3.2对策  
3.3.3建立知识库  
3.4小结  
第4章调试器内部机制  
4.1调试器的种类  
4.1.1机器调试器与符号调试器  
4.1.2调试基础：自定义构建  
4.1.3调试基础：系统调用  
4.1.4调试基础：解释器  
4.1.5内核调试器  
4.1.6界面：命令行与图形用户界面  
4.2符号调试器扩展  
4.2.1动态断点  
4.2.2单步执行  
4.3应对策略  
4.3.1系统调用  
4.3.2移除调试信息  
4.3.3代码盐  
4.3.4混合内存模型  
4.4小结  
第5章优化：内存占用  
5.1被遗忘的历史  
5.2程序的内存布局  
5.2.1场景：单段程序  
5.2.2场景：仅代码段和数据段  
5.2.3场景：所有4种段类型  
5.3代码段  
5.3.1剪切粘贴式编程  
5.3.2宏  
5.3.3僵尸代码  
5.4数据段  
5.4.1双重用途的数据结构  
5.4.2位域  
5.4.3压缩算法  
5.5栈  
5.5.1活动记录  
5.5.2函数参数  
5.5.3局部变量  
5.6堆  
5.6.1内存池  
5.6.2回收  
5.6.3延迟实例化  
5.6.4跟踪内存使用情况  
5.7小结  
第6章优化：CPU周期  
6.1程序控制跳转  
6.1.1标签与GOTO  
6.1.2函数参数  
6.1.3带可变参数的函数  
6.1.4系统调用  
6.1.5递归  
6.2程序控制分支  
6.2.1查找表  
6.2.2switch与if-else  
6.2.3常见情况放在前，罕见情况放在后  
6.3程序控制循环  
6.3.1循环不变量  
6.3.2函数调用  
6.3.3数组引用  
6.3.4分解复合布尔表达式  
6.3.5循环展开  
6.3.6循环干涉  
6.3.7提取程序分支语句  
6.4内存管理  
6.4.1处理开销  
6.4.2引用局部性  
6.5输入/输出  
6.5.1缓存  
6.5.2缓冲  
6.5.3先进的技术  
6.6异常  
6.6.1动态注册模型  
6.6.2静态表模型  
6.6.3处理开销  
6.6.4滥用异常  
6.7昂贵的操作  
6.7.1消除常见的子表达式  
6.7.2浮点运算神话  
6.7.3强度折减  
6.7.4同步  
6.7.5简写操作符的神话  
6.8快速修复  
6.8.1更好的硬件  
6.8.2约束问题  
6.8.3编译器设置  
6.9小结  
6.10信息汇总  
第7章最后的赠言  
7.1对于源代码完整性的其他威胁  
7.1.1时髦技术：一个案例研究  
7.1.2洗脑101  
7.1.3真正的问题  
7.2保持书面记录  
7.2.1悄悄记录  
7.2.2隐私的神话  
7.3历史总是重演

[显示部分信息](http://product.dangdang.com/23471929.html" \l "catalog)

前言

　　据记载，在典型的软件项目生命周期中，维护阶段的成本占总成本的60%以上。研发需要什么样的排场，什么样的环境，一般都会在设计阶段确定，一旦完成第一个版本的部署，接下来的大部分资源都将用于修复bug，增加新的功能，并依次修复这些新的功能所产生的bug。多数的软件应用程序在发布后的开发周期如图1所示。  
　　图1软件应用程序在发布后的开发周期  
　　产品发布后的开发周期始于新增一个功能，或者提交一份bug报告。在这两种情况下，单元测试和系统测试常用来确定补丁代码是否工作正常。一旦程序的执行已经足够稳定，就可以通过剖析应用程序来定位性能瓶颈。然后实施各种各样的优化技术，以提高执行速度和减少内存占用。随着客户要求加入新特性和修复bug，这种周期会周而复始地发生，应用程序也正是这样不停地向前演进。  
　　关于软件工程的大多数书籍更关注一个应用程序从立项到发布的各个开发阶段（例如，需求、分析、用例、实现等）。大家都知道，从头写一个程序会是一件更容易、更开

[显示全部信息](http://product.dangdang.com/23471929.html" \l "preface)

在线试读部分章节

　　第1章  
　　预防药  
　　概要  
　　老实说，相比那些在事后马虎的人而言，对那些在一开始就马马虎虎的人我会毫不犹豫地将其淘汰掉。这听起来相当冷酷无情，这的确冷酷无情。但是，这并不是那种“如果你无法忍受炙热，那就远离厨房”的提醒，它更深刻。我不愿与那些马马虎虎的人共事。这是软件开发的进化论。  
　　——LinusTorvalds在Linux内核邮件列表里关于内核调试器的讨论中如是说  
　　维护工程师的角色是驱逐那些潜藏在遗留软件里的邪恶幽灵，他们夜以继日维护着那些软件，上下交困，怨声载道，就像在穿过公司的那条小径上孤独前行。每天，他们都面临诸如：  
　　修复bug  
　　性能提升  
　　这样的挑战。本书前面4章会重点关注于前者。最后两章会关注后者。  
　　在1896年出版的一本名为《Hawkin’sNewCatechismofElectricity》的电气手册中，首次提出bug这个术语，“在有限范围内，用于指明与电气设备相关的或电气设备工作中出现的任何缺陷故障”。这个术语在RearAdmiralGraceMurrayHopper（COBOL语言的发明者)的帮助下逐步进化成用于表示计算机程序的故障。  
　　与bug打交道是程序员最讨厌的工作任务之一。基于此，处理软件bug最好的时机就是在软件的实现阶段。换句话说就是，在问题即将成为软件发行版本中的一部分之前,这些bug还没有“潜伏”到软件中，就修复它，这样，维护工程师在后续的工作中，就不用再陷入处理这些问题的烦恼中。包括LinusTorvalds在内的很多专业人士，都认同这套思路。  
　　《孙子兵法》有云：“上兵伐谋，其次伐交，其次伐兵，其下攻城。攻城之法，为不得已。”解决大问题固然重要，杜绝小问题同样不可小视；解决大问题彰显才华卓越，杜绝小问题更显先见之明。作为软件工程师，要有远见，尽早地实现缺陷的隔离和修复。这样，你就不用再担心凌晨3点钟还会接到来自愤怒的系统管理员的电话。  
　　本章讨论用于减少代码受到bug影响的预防性测量。如果你是正在参与构建一个全新软件的工程师，或者你正在新增软件特性，那么，你需要认真地阅读本章。此外，如果你已经接手了你从未写过的一个应用（并且这种情况应该很常见），那么，我推荐你先跳过本章，从下一章开始。一旦你处理完那些讨厌的bug，你可以再回到本章，使用本章介绍的技术来帮助你清理补丁程序