第33卷 第12期 计 算 机 工 程 2007年6月
Vol.33 No.12 Computer Engineering June 2007
·软件技术与数据库· 文章编号：1000—3428(2007)12—0082—02 文献标识码：A 中图分类号：TP311.53

基于代码检测的软件故障定位方法

易昭湘，慕晓冬，赵 鹏，张雄美
(第二炮兵工程学院，西安 710025)

摘 要：针对现有软件故障定位方法的缺陷，提出了一种基于代码检测的软件故障定位方法，用嵌入式模块获取软件发生故障时的模块运行序列，分析出软件故障可疑模块集及其故障系数，在此基础上对故障模块进行代码的分类检测，根据上述过程中得到的结果进行综合分析运算，得出软件故障的可疑代码集和故障系数，采用代码分析辅助工具进行排查，定位故障。该方法已成功应用于软件密集型系统的故障诊断，能快速有效地实现软件故障定位。

关键词：软件故障；故障定位；代码检测；软件密集型系统

Software Fault Location Based on Checking Codes

YI Zhaoxiang, MU Xiaodong, ZHAO Peng, ZHANG Xiongmei
(The Second Artillery Engineering College, Xi'an 710025)

【Abstract】 Against the limitations of the current software fault location, a new method based on checking codes is presented. The embedded modules are utilized to generate the implement sequence of software modules, and obtain a set of suspicious modules and their fault coefficients. Then, the codes of suspicious modules are checked by classification. Based on previous results, the set of suspicious codes and their fault coefficients are determined by the synthetical algorithm. The fault codes are located by eliminating the suspicious codes with the assistant tool. The method has been applied to the fault diagnosis of SIS, and proved to be effective for software fault location.

【Key words】 Software fault; Fault location; Checking codes; Software intensive system(SIS)

随着计算机技术的发展和软件应用领域的扩展，由软件故障而引发的软件维护等一系列问题逐步引起人们的重视。美国从20世纪80年代开始就对软件维护问题行了系统、深入的理论研究和实践，取得了大量的研究成果[1]，2005年的国际软件维护会议(ICSM'05)提出了SSME[2](software support, management, and evolution)，即集软件保障、管理和演化为一体的综合软件维护技术，标志了软件维护新时代的来临。相对国外，国内在软件维护方面处于起步阶段，针对软件故障进行综合分析的理论和方法都在探索中。本文将重点讨论一种基于代码检测的软件故障定位方法。

1 软件故障定位

1.1 软件故障

软件故障是指软件在运行过程中出现的一种不正常的内部状态[3]。软件故障通常是在软件的设计过程中引入的，随着软件系统应用的深入，软件的复杂程度会不断增加，因此任何软件设计方法、编程语言和工具，以及软件开发人员都不能保证在软件设计和程序编码过程中不引入故障。

1.2 软件故障定位方法

当软件发生故障时，需要进行故障定位，深入代码的内部找出故障模块和故障代码。目前常用的故障定位方法有原始法、回溯法和排除法等[4]。原始法是通过计算机在出现故障的程序范围内找错，该方法复杂且效率低。回溯法是从出现故障征兆处沿程序的控制流程往回追踪，直到发现出错的根源，该方法缺陷具有一定的局限性，当程序范围较大时，由于回溯路线急剧增加，无法实现完全回溯。排除法基于归纳和演绎推理，根据故障数据，假设错误原因，通过测试逐一排除，该方法要求对软件总体设计和代码结构十分了解。

由于软件故障问题的复杂性，现有的故障定位方法存在过程复杂、依赖性强和通用性不高的缺陷，都不能满足软件故障诊断的要求。为此，本文提出一种基于代码检测的软件故障定位方法。

2 基于代码检测的软件故障定位方法

2.1 总体设计思想

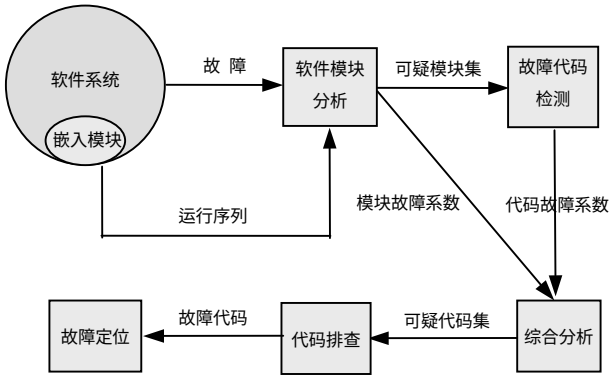

图1 基于代码检测的软件故障定位方法

基于代码检测的软件故障定位方法总体设计思想是：先通过嵌入式模块分析方法获取软件发生故障时的模块运行序

基金项目：武器装备预研项目
作者简介：易昭湘(1980－)，男，博士生，主研方向：故障诊断，软硬件故障分离；慕晓冬，教授；赵 鹏，博士生；张雄美，硕士生
收稿日期：2006-06-30 **E-mail**：yzx8848@163.com

列，从中分析出软件故障可疑模块集；然后采用故障代码检测的方法对可疑模块集中的软件代码进行检测；再根据模块分析和代码检测的结果，采用综合分析算法进行代码过滤和软件故障分析，得出可能的故障代码集和故障系数；最后再对错误代码集进行排查，定位发生故障的代码。本方法的结构如图 1 所示。

2.2 软件模块分析

软件故障与软件的运行有紧密的联系，因此要定位软件故障首先需要对软件的运行过程进行分析[1]。为此，在软件系统中嵌入特定的模块，监控软件的运行状态，实时记录软件内部模块运行状态和数据。在软件故障发生时，依据记录的数据可以分析出软件系统中模块的运行序列、数据输入和数据输出。不妨设该软件系统由n个软件模块$m_1,m_2,\ldots,m_n$组成，即

$$M=\{m_1,m_2,\ldots,m_n\}\quad n\in N$$

发生故障时的软件模块运行序列为

$$m_i\rightarrow m_{i+1}\rightarrow m_{i+2}\ldots\rightarrow m_{i+k}\quad i,k\in N$$

用u_i表示软件运行序列中第i个模块的发生故障的概率，则得到模块序列故障概率集

$$U=\{u_i,u_{i+1},\ldots,u_{i+k}\}\quad i,k\in N$$

其中，u_{i+k}的值随k增加而增大，越靠近软件故障点，所运行模块的故障率越大。同时，由于软件的各个模块之间可能存在数据交换和共享，软件故障也可能由多个模块传递的数据而引起，因此建立相关矩阵E表示模块之间的相关程度。

$$E=\begin{bmatrix} e_{11} & e_{12} & \cdots & e_{1n} \\ e_{21} & e_{22} & \cdots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} & e_{n2} & \cdots & e_{nn} \end{bmatrix}$$

其中，e_{ij}表示在m_i后运行m_j的相关系数，该系数取决于模块之间传递的数据和模块对数据的处理。

通过上面的分析，得到的故障可疑模块集为

$$S=\{m_1,m_2,\ldots,m_i\}\quad i\le n$$

设m_i运行次数为l，用w_i表示S中第i个模块m_i的故障系数，则

$$w_i=\sum_{i=1}^{l}u_i e_{ji}$$

2.3 软件故障代码的检测

上面讨论的可疑模块集 S 包含了所有运行的代码，为了有效定位故障，需要进一步深入代码的内部进行故障代码检测。

由于采用不同的语言编写的源代码具有各自的特点与风格，首先必须建立包含多种语言的语言故障数据库，存储每种语言的故障科目类别和代码特征。在此基础上进行如图 2 所示的分析过程。

先分析出可疑模块集 S 所在的源代码文件中的具体位置，然后逐个调入可疑模块的代码，分别进行变量、运算符、数组、指针、保留字和系统函数检测，同时对复杂代码行进行二次关联检测，最后将分析的结果输出。

设c_{ik}表示的是m_i的第k行代码，用$\Theta_k(k=1,2,3,4,5,6,7)$分别代表变量、运算符、数组、指针、保留字、系统函数和二次关联检测的系数，n_j表示c_{ik}在第j种类型检测所中出现的频率。则c_{ik}的故障系数v_{ik}可表示为

$$v_{ik}=\sum_{j=1}^{7}n_j\Theta_j$$

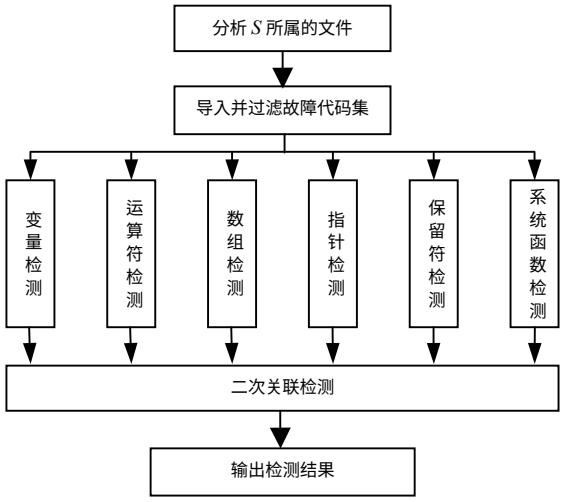

图 2 软件故障代码检测过程

2.4 故障代码综合分析及算法

前面的分析已得出可疑模块m_i、模块故障系数w_i、可疑源代码c_{ik}及其故障系数v_{ik}，通过$p_{ik}=w_i\times v_{ik}$可以得到可疑代码发生故障的故障系数p_{ik}。综合分析的算法如下所示。

(1)定义整数 i、j，和二元数组 $A(x,y)$，其中 x 保存故障代码编号，y 保存故障系数；

(2)输入可疑模块集 S 和源代码，设 $sum1$ 为 S 中模块的总数，令 $i=sum1$；

(3)调入可疑模块m_i源代码，设$sum2$ 为模块m_i中源代码的行数，令$j=1$；

(4)分析模块m_i中的第j行代码，如果$v_{ij}>0$，用公式$p_{ij}=w_i\times v_{ij}$计算出故障系数p_{ij}，将该行代码的编号和p_{ij}存入二元数组$A(x,y)$；

(5)令 $j=j+1$，如果 $j\le sum2$，跳转到(4)；

(6)令 $i=i-1$，如果 $i>0$，跳转到(3)；

(7)输出 $A(x,y)$，其保存的是故障可疑代码集。

2.5 软件故障代码的定位

前面的分析已得到可疑代码集及故障系数，相对于可疑模块集 S 而言，其将故障锁定在一个很小的范围内，但是由于源代码的逻辑复杂性，定位故障还需要在此基础上利用代码分析辅助工具进行排查和分析，找出故障的原因和位置。

3 应用实例

利用上述方法对某软件密集型系统的指控软件(采用 VC++编程)进行故障定位，首先通过模块运行序列分析得到可疑模块集(图 3)，然后进行代码检测(图 4)，再进行综合分析得到软件可疑代码集(图 5)。从运行结果可以看出，基于代码检测的软件故障定位方法与现有的方法相比，不仅能实现快速高效的故障定位，而且通用性强，不受编程语言和代码风格的限制。

模块名	文件名	参数名	参数值
OnActive	TargetRvmt.cpp		
OnInit	TargetPzpr.cpp		
OnActive	TargetPzpr.cpp		
SetCap	TargetSht.cpp		
OnActive	TargetPzpr.cpp		
OnInit	TargetPozp.cpp		
OnActive	TargetPozp.cpp		
SetCap	TargetSht.cpp		
OnActive	TargetPozp.cpp		
OnActive	TargetPzpr.cpp		
SetCap	TargetSht.cpp		
OnActive	TargetPzpr.cpp		
OnTarget	TargetRvmt.cpp		
OnTarget	TargetRvmt.cpp	X	1455656.00
OnTarget	TargetRvmt.cpp	Y	50657676.0
OnTarget	TargetRvmt.cpp	n_h	23
OnTarget	TargetRvmt.cpp	n_m	55
OnTarget	TargetRvmt.cpp	n_s	6

图 3 模块运行序列

(下转第 89 页)