一个异常传播分析工具的设计与实现*)

姜淑娟[1] 徐宝文[2] 姜元鹏[3]

(中国矿业大学计算机科学与技术学院 徐州 221116)[1] (东南大学计算机科学与工程学院 南京 210096)[2]

(中国矿业大学图书馆 徐州 221116)[3]

摘 要 异常处理是一种用来检测异常并对其进行处理的技术,异常传播改变程序原来的执行路线,从而可能改变程序中的数据流、控制流和各种成分的依赖关系。在进行程序分析时,如果不考虑异常传播对其造成的影响,则得到的信息将是不准确的。本文设计并实现了一个分析C++程序中异常传播分析工具CETool。该工具可以有效地分析C++程序的异常传播,它既可以获得异常处理结构的局部信息,也可以获得异常处理结构的全局信息,这对于有效地分析异常的传播,分析异常传播路径,以及改进异常处理结构都有很大的帮助。

关键词 异常处理,程序分析,异常传播,设计与实现

Design and Implementation of a Exception Propagation Analysis Tool

JIANG Shu-juan[1] XU Bao-wen[2] JIANG Yuan-peng[3]

(School of Computer Science & Technology, China University of Mining & Technology, Xuzhou 221116, China)[1]

(School of Computer Science & Engineering, Southeast University, Nanjing 210096, China)[2]

(Library, China University of Mining & Technology, Xuzhou 221116, China)[3]

Abstract Exception handling is a technology that tests and handles exception. Exception propagation induces a control flow other than the main control flow, so it changes the data flows, control flows of programs and the dependence relationshi Pbetween the structure elements of programs. For the analysis of C++ programs to be correct and precise, the flows induced by exception propagation must be properly analyzed. The paper describes the design and implementation of the CETool, an exception propagation analysis tool we have developed to provide exception propagation information for C++ systems. The CETool can effectively analyze the exception propagation of C++ program. It can provide the local information of the exception propagation of an exception, and also the whole information of exception propagation for the program. The information is helpful to analyze exception propagation, analyze exception propagation path, and support the improvements to the exception handling structure of a system.

Keywords Exception handling, Program analysis, Exception propagation, Design and implementation

1 引言

大多数应用领域对系统的稳定性、健壮性和可用性都有较高的要求。在其设计与实现中如何保证系统在出现故障时维持正常的服务,即如何进行系统容错是一个至关重要的问题。异常处理是目前用于软件容错技术中较常用的方法之一。自从John Goodenough于1975年首次提出异常处理的概念以来[1],人们进行了比较深入的研究,使得它在软件的开发与应用等方面都有了广泛的应用,异常处理设施已经成为高级程序设计语言中不可缺少的部分(如Ada,C++,Java等)。如果异常处理机制使用得当,确实能提高软件的健壮性[2]。但由于异常传播改变了程序原来的执行路线,从而改变了程序中数据流、控制流和程序各元素之间的依赖关系[3-5]。在进行程序分析时,如果不考虑异常传播对其造成的影响,则得到的信息将是不准确的,把这些不准确的信息用在结构测试、回归测试、程序切片等软件工程的任务中时,可能导致严重的后果[3]。

我们曾对异常传播分析做过比较深入的研究,取得了一定的成果[6-8]。本文是在以前工作的基础上,总结和扩展相关的工作,实现部分异常传播分析工作的自动化处理。

2 系统设计

我们在对C++语言的异常处理机制充分研究的基础上,经过仔细地分析和比较国内外先进的软件分析和维护工具[5,9,10],并结合以前研究工作的实际经验,针对C++语言的异常处理机制的特点,设计并实现了一个C++程序异常传播分析工具CETool。在此基础上,进一步进行有关程序理解与分析、异常传播分析、异常处理机制的改进等方面的理论与实现技术的研究。

本工具主要根据源程序,对程序中异常处理结构进行分析。分析程序中异常抛出的位置与异常类型、异常被捕获的位置与异常类型、异常的传播路径,以及与异常处理相关的各种统计信息。在此基础上,构建程序的改进的控制流图,然后对异常传播对程序的控制流、数据流、控制依赖和数据依赖的影响进行分析。CETool系统的总体框架如图1所示。CETool系统以源代码为输入,通过词法分析、语法分析,抽取程序的内部表示——中间信息库。利用信息库中记录的信息,进行异常传播分析、异常传播路径分析以及异常传播对程序

*)本文得到国家自然科学基金(90412003,60503033),江苏省高技术研究项目(BG2005032)中国矿业大学校科学研究基金资助。**姜淑娟** 教授,博士,主要从事程序设计语言、软件工程等方面的教学与研究工作;**徐宝文** 教授,博士生导师,主要从事程序设计语言、软件工程、并行程序设计等方面的教学与科研工作;**姜元鹏** 工程师,主要从事网络信息处理、程序分析与测试工作。

的控制流、数据流、控制依赖和数据依赖的影响分析等，并对异常传播路径进行可视化显示。异常传播分析工具 CETool 可以分析出 C++源程序中的异常处理结构信息，并用图示的方式显示出来。主要功能如下：

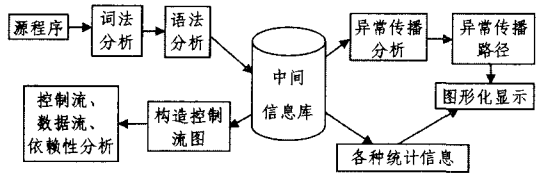

图 1　系统结构图

(1) CETool 可以计算出一定范围内所传播出的异常，并显示出所有可能显式引发异常的引发位置、异常的类型，可以显示从 try 块中传播出异常的类型、异常引发的位置以及传播异常的函数名等。

(2) 计算 catch 语句所捕获的异常的类型，检测不可达的 catch 语句，检测 catch 语句所捕获的异常类型与引发的异常类型是完全相同？还是通过包含的关系捕获的？

(3) 对选定的异常用图示的方式显示其传播路径。

(4) 根据中间信息库可以构造出控制流图，在此基本上可以进行异常传播对程序的控制流、数据流、控制依赖和数据依赖的影响分析。具体的分析方法见参考文献[7,8]。

3　系统主要数据结构

主要介绍系统中与异常处理相关的数据结构的信息。分析数据中包括：该异常处理结构的起始行数、异常处理结构所在的源文件的绝对路径名、异常处理结构所在的函数名以及该函数所在的类名、异常处理结构中所保护的程序语句数、throw 语句所在的位置(是否在 try 块内)、throw 语句所抛出异常的异常类型、catch 语句块的个数以及每个 catch 中所能处理的异常类型、在 try 语句块中调用了那些含有异常处理结构的其他函数、以及与统计相关的信息，如总 throw 数、总 catch 块数、catch 块中包含的总语句行数等等。

本系统中有一个统计的功能，用来显示程序中有关异常处理的统计信息，其结构如下：

```
struct SanalyzingResult{
    int m_nFileNum;
    int m_nLineNum;
    int m_nTryNum;
    int m_nTryLineNum;
    int m_nCatchLineNum;
    int m_nCatchNum;
    int m_nThrowNum;
}
```

其中，各个字段的含义如下：

m_nFileNum 用来表示总文件的个数；

m_nLineNum 用来表示程序的总行数；

m_nTryNum 用来表示所分析程序中的 try 块的总个数；

m_nTryLineNum 用来表示所分析程序中所有 try 块中语句的总行数；

m_nCatchLineNum 用来表示所分析程序中的所有 catch 块中语句的总行数；

m_nCatchNum 用来表示所分析程序中 catch 块的总个数；

m_nThrowNum 用来表示所分析程序中的 throw 语句的总个数。

在分析过程中需要记录每个 try 块和 throw 语句的相关信息，其结构如下：

```
struct STryBlockInfo{
    BOOL m_IsTryBlock;
    int m_nTryLine;
    int m_nTryLineNum;
    int m_nCatchNum;
    int m_nTryEndLineNum;
    char m_szFilePathName[488];
    char m_szClassName[64];
    char m_szFunctionName[64];
    char m_szExceptionName[384];
    char m_szCallFuncName[512]; }
```

其中，各个字段的含义如下：

m_IsTryBlock 用一个布尔量来表示本结构记录的是 try 块还是 throw 块；

m_nTryLine 用来表示 try 或 throw 语句所在的行数，根据 m_IsTryBlock 的值来判断该变量所表示的是 try 语句所在的行数还是 throw 语句所在的行数；

m_nTryLineNum 用来表示 try 块中所包含的行数；

m_nCatchNum 用来表示与该 try 块匹配的 catch 块中的 catch 语句的个数；

m_nTryEndLineNum 用来表示该 try 块是最后一行的行数；

m_szFilePathName 用来表示该 try 块或 throw 语句所在的文件名；

m_szClassName 用来表示该 try 块或 throw 语句所在的函数的所属类名；

m_szFunctionName 用来表示该 try 块或 throw 语句所在的函数名；

m_szExceptionName 用来表示在 try 块中处理的异常或 throw 块抛出的异常的名字；

m_szCallFuncName 用来表示 try 块中所调用的其它函数的函数名。

在程序分析的过程中，要随时记录当前分析的元素的状态，其结构如下：

```
enum ANALYZING_STATE{
    NORMAL,
    ENTER_TRY,
    IN_TRY,
    OUT_TRY,
    ENTER_CATCH,
    IN_CATCH,
};
```

其中，各个字段的含义如下：

NORMAL 表示当前分析的元素所在的状态是普通状态，即不在 try 块内，也不在 catch 块内；

ENTER_TRY 表示当前分析的元素进入 try 块；

IN_TRY 表示当前分析的元素在 try 块中；

OUT_TRY 表示当前分析的元素在 try 块的末尾；

ENTER_CATCH 表示当前分析的元素进入 catch 块；

IN_CATCH 表示当前分析的元素在 catch 块中；

下面给出几个主要的分析程序的简单说明：

```
void Analyze(PSTR lpszFilePathName)
    // 分析指定的 VC 工程文件
SAnalyzingResult GetAnalyzingResult() const { return m_AnalyzingResult; }
    // 返回分析结果结构
STryBlockInfo * GetTryBlockInfo(int nTryBlockNo)
    // 得到某个 try 块的分析结果
int GetTryBlockNum() const { return m_nCurInfoPos; }
    // 返回分析结果中的 try 块和 throw 语句的相关信息
CString GetAnalyzeFilePathName() const { return m_szAnalyzePathName; }
    // 得到分析文件的路径名
CString GetAnalyzeFileName() const { return m_szAnalyzeFileName; }
    //得到分析文件名
void AnalyzeFile(PSTR lpszFilePathName);
    //分析指定的 C++程序文件
void AnalyzeProject(PSTR lpszFilePathName);
    //提取出指定的 VC 工程中的 C/C++程序文件路径
void AnalyzeException();
```

```
        //分析 try 块和 throw 块所在的类和函数名,以及 catch 所能处理的异常的
        //异常类型,以及 throw 块抛出的异常的异常类型
    void AnalyzeTry();
        //分析 try 块中是否调用了其它含有 try 块或 throw 语句的函数
    int FindWord(const char * str1, const char * str2, BOOL bIsWord = TRUE, int nLeftWord = -1);
    //在给定的字符串中,查找是否有某字符串,主要用于 catch 中的
        //异常类型与所引发的异常类型的匹配
    void NoteTryBlock(STryBlockInfo TryBlockInfo);
        //记录 try 块信息
    void ShowExceptionPropagationPath(Char * str);
        //显示选定的异常的传播路径
```

4 系统主要界面

为了更好地将异常传播分析结果显示给用户,将 CETool 的主界面分为四个界面区域,为了叙述方便,我们用 A,B,C,D 来标注这四个界面区域。

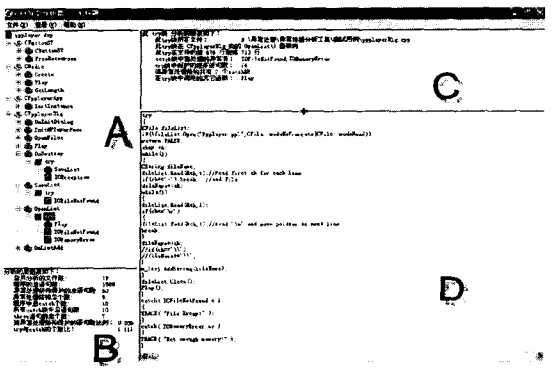

图 2 异常传播分析工具 CETool

A 区为一个树状视图,这个树状视图能够很好地表现出程序中异常结构的层次(如图 2 所示),整个树分为五层。树的根节点为程序的工程名,代表着整个程序;根结点的所有子节点表示着程序中的类;每个类节点的子节点则表示着该类下面的成员函数;每个成员函数的子节点则是该成员函数所包含的所有异常处理结构块(try—catch)和 throw 语句;如果是异常处理结构块节点,并且在该异常处理结构中包含有函数调用语句,当被调用函数中含有异常处理结构或是 throw 语句时,该异常处理结构节点下会有被调用函数名节点。以这种方式来显示异常处理结构在程序中的层次关系,可以很容易地了解整个程序中的异常处理结构。从异常处理结构块节点以及在该异常处理结构的 try 块中调用的其他函数,也能够在一定程度上表现出程序的异常传播路径,但这种表现不够直观,例如,如果在一个异常处理结构块 P 中调用了某个包含异常处理结构 Q 的函数,而在这个异常处理结构 Q 中又调用了另一个包含一个 throw 语句的函数,那么当 throw 语句抛出一个异常时,这个异常如果没有被中间这个函数的异常处理结构 Q 处理的话,就会传播到异常处理结构 P 来处理,而在树状图中,则会表现在两个树的分支上,因此,使用树状图来查看异常传播路径是比较繁琐的。为了解决这个问题,我们在 CETool 中还添加了一个专门用来显示异常传播路径的功能。当用户想看在某个异常处理结构中某个异常的传播情况时,只要在 A 区的树状图上的该异常的 throw 节点上双击,就可以弹出一个新的窗口来显示该异常的传播路径。

在图 2 中的 B 区是一个显示总体分析结果的列表框,在这个列表框中显示程序的总语句数、总异常处理结构块数、catch 语句数等统计信息,还会显示各种比例信息以及文件的各种与异常处理相关的信息。C 区和 A 区是关联在一起的,当在 A 区的树状结构中选择了第 4 层的一个节点(也就是选择了一个异常处理结构节点或是 throw 语句节点),在 C 区就会显示这个异常处理结构或是 throw 语句的相关信息,比如它在程序中的位置、它所在文件的绝对路径等。而当在 A 区的树状结构中选择的是一个异常处理结构时,在 D 区中会显示这个异常处理结构的源程序,这样可以使程序开发人员进一步地仔细察看异常处理结构代码,方便程序开发人员根据分析信息来调整不合理的异常处理结构。图 3 是在 A 区的树状图上的某个异常的 throw 节点上双击,弹出一个新的窗口来显示该异常的传播路径。

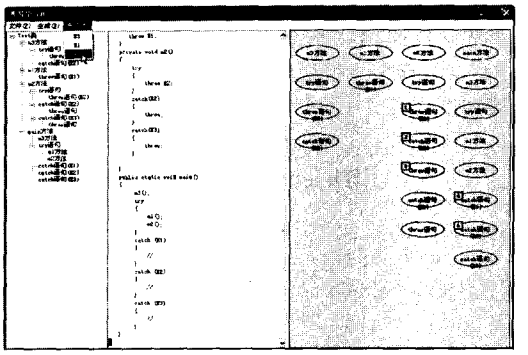

图 3 异常传播路径

结束语 本文主要讨论了异常传播分析工具 CETool 的系统总体框架和与异常传播分析相关的主要数据结构。介绍了其主要的功能、主要的界面和主要的数据结构,限于篇幅,并没有对其作更深入的讨论分析。该工具目前还只能给出显式引发的异常的有关信息,对隐式引发的异常的有关信息还没有实现,这也是我们下一步研究的方向。我们拟在此基础之上针对 C++异常处理机制的特点,特别针对异常传播对程序中的控制流、数据流、控制依赖和数据依赖的影响,在程序分析、程序维护等方面作进一步的研究,希望在理论及实现上能做更深入的探讨。

参 考 文 献

[1] Goodenough J B. Exception handling: issues and a proposed notation. Communications of the ACM, 1975, 18(12): 683-696

[2] Jiang SJ, Xu BW. Exception handling-an approach to improving software robustness. Computer Science, 2003, 30(9): 169-172

[3] Sinha S, Harrold M J. Analysis and testing of programs with exception-handling constructs. IEEE Transactions on Software Engineering, 2000, 26(9): 849-871

[4] Sinha S, Harrold M J. Criteria for testing exception - handling constructs in Java programs // Proceedings of the International Conference on Software Maintenance. IEEE Computer Society Press, Los Alamitos, CA, 1999: 265-274

[5] Sinha S, Orso A, Harrold MJ. Automated support for development, maintenance, and testing in the presence of implicit control flow // Proceedings of the 26th International Conference on Software Engineering (ICSE 2004). IEEE Computer Society, 2004: 336-345

[6] Jiang shujuan, Xu Baowen, Shi liang. An approach to Analyzing Exception Propagation // The 8th IASTED International Conference On Software Engineering And Applications. MIT, Cambridge, USA: 300-305

[7] 姜淑娟,徐宝文,史亮. 一种基于异常传播分析的数据流分析方法. 软件学报, 2007, 18(1): 74-84

[8] 姜淑娟,徐宝文,史亮,等. 一种基于异常传播分析的依赖性分析方法. 软件学报 2007, 18(4), 832-841

[9] Robillard M P, Murphy G C. Static analysis to support the evolution of exception structure in object-oriented systems. ACM Trans. on Software Engineering and Methodology, 2003, 12(2): 191-221

[10] Fu C, Ryder B G, Wonnacott DG. Robustness testing of Java server applications. IEEE Trans. on Software Engineering, 2005, 31(4): 292-311