采用Clang/LLVM的C++源代码覆盖率分析插装方法

李树芳 安金霞 刘 洋 陈 良
(中国酒泉卫星发射中心 酒泉 732750)

摘 要 近年来,越来越多的安全关键软件系统运行在国产Linux操作系统上,其中大多数采用C++开发,而C++正在扩展新版语言规范,已有的覆盖率统计插装工具不能满足要求。给出一种基于Clang/LLVM的C++源代码覆盖率统计插装方法,利用开源社区Clang/LLVM库提供的C++源代码解析和操作功能,构建面向C++源代码的语句、分支和MC/DC覆盖率统计框架,在运行时采集覆盖率信息并进行统计分析,输出覆盖率分析报告。实际案例表明,该方法简易实用,能够满足真实工程软件的覆盖率测试分析需求。

关键词 C++,Clang,代码插装,覆盖率分析
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.029

Approach to C++ Code Instrumentation for Coverage Analysis with Clang/LLVM

LI Shu-fang AN Jin-xia LIU Yang CHEN Liang
(Jiuquan Satellite Launch Center of China, Jiuquan 732750, China)

Abstract In recent years, many safety-critical software system have been running under domestic Linux operating systems, most of which are developed with C++. As C++ is expanding to new revision of the C++ ISO/IEC standards, the existing source code instrumentation tools for coverage analysis cannot satisfy user's requirements. In this paper, an approach to C++ source code instrumentation for coverage analysis with Clang/LLVM was proposed. Employing the Clang/LLVM library in open source community, which provides C++ source code parsing and manipulation functions, a framework of C++ coverage analysis for statement coverage, branch coverage and MC/DC coverage was constructed. Coverage information was collected in real-time, and was analyzed and exported to reports. Case study shows that the proposed approach is very convenient and practical in real world software testing activities.

Keywords C++, Clang, Code instrumentation, Coverage analysis

1 引言

C++语言是最受欢迎的主流程序设计语言之一。近年来,越来越多的安全关键软件系统运行在以银河麒麟为代表的国产Linux操作系统上,其中大多采用C++开发。C++11和C++14等新版规范相继发布,并得到GCC,Clang/LLVM,Visual Studio等主流编译器的全面支持。例如,GCC6默认采用gnu++14规范(即C++14的GNU扩展),QT5代码遵循C++11规范,Clang/LLVM自身代码使用C++11规范,许多开源软件、实际工程项目的软件系统也开始采用新版C++语言规范。C++语言从C++11开始增加了nullptr,constexpr和static_assert等关键字和lambda函数与表达式等新的语言特性,已有的主流C++源代码覆盖率统计插装工具一般仅支持C++98和C++03语言规范,无法用于遵循新版C++语言规范的软件源代码。

我国载人航天工程要求安全关键等级为A级和B级的软件单元测试的语句、分支、修正条件判定MC/DC覆盖率均应达到100%,对于由于测试条件限制而覆盖不到的语句、分支,应进行逐一分析和确认,并做出分析说明。因此,必须利用测试工具支持并收集被测软件的代码覆盖率,才能有效分析C++源代码的覆盖率。除了对已有测试工具进行升级之外,开源社区还提供了新的选择。

前期工作中,已提出了一种面向大型实时软件系统的测试覆盖率快速分析方法[5,7-8],其借助自主开发的C++代码覆盖率插装与分析工具,实现批量化的代码插装,实时采集和统计分析被测软件的语句、分支、条件、MC/DC和函数等覆盖率信息,并输出测试覆盖率分析报告。通过一系列脚本和工具链支持,实现全过程的自动化,以满足航天领域大型实时软件系统的特殊需求。开发了RtCppTest工具平台,并将开源PUMA库[4]作为C++源代码插装处理的前端。由于PUMA库只能部分支持C++11规范,不支持C++14规范,且其自身的API不够健壮,在扫描特定C/C++语法的源代码时会发生指针访问错误,造成崩溃,因此PUMA库所属Aspect C++项目[3]从2016年的2.0版本开始将其默认前

到稿日期:2016-10-10 返修日期:2016-12-13
李树芳(1978-),男,硕士,高级工程师,主要研究方向为软件工程,E-mail:shufang@ustc.edu;安金霞(1973-),女,博士,高级工程师,主要研究方向为大数据分析;刘 洋(1983-),硕士,工程师,主要研究方向为软件测试;陈 良(1987-)男,硕士,工程师,主要研究方向为网络安全。

端改为 Clang，以充分利用优秀的开源成果，降低开发维护成本。

本文给出了一种基于 Clang 的 C++源代码覆盖率统计插装方法，该方法利用开源社区 Clang 库提供的强大的C++源代码解析和操作功能，可快速构建面向 C++源代码的语句、分支和 MC/DC 覆盖率统计框架，并将其作为 C++源代码插装处理的前端嵌入 RtCppTest 工具平台，在运行时采集覆盖率信息并进行记录和统计分析，输出覆盖率统计分析报告。

2 Clang/LLVM 的 C++源代码处理框架

Clang 是美国 Apple 公司开发的一种 C/C++/ Obj-C 编译器前端，并于 2007 年开源。Clang 构建在 LLVM(Low Level Virtual Machine，一个模块化和可重复使用的编译器和工具技术的集合，提供了与编译器相关的支持，可以用作多种语言编译器的后台)的基础上，与 LLVM 一起构成完整的编译工具链，支持多种操作系统和硬件架构，可替代 GCC 编译系统。其由于采用模块化的库架构设计，很容易嵌入到各种应用程序中使用。

Clang 提供了 C++源代码分析的强大支持库，从 3.3 版本开始全面支持 C++11，3.4 版本已全部支持 C++14，甚至部分支持 C++1z 规范。目前，Clang 已是 Apple OS X 和 FreeBSD 10.x 操作系统的默认编译系统，达到了工业级产品质量标准。

基于 Clang/LLVM 的 C++源代码抽象语法树遍历和代码插装操作框架类图如图 1 所示，利用 Clang/LLVM 的 LibTooling 库可实现独立工具。

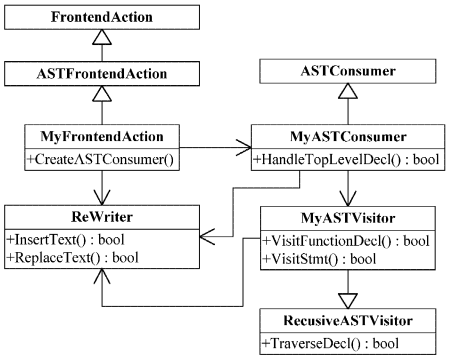

图 1 Clang 的 C++源代码插装处理框架类图

(1)类 ClangTool 是工具入口，读入命令行配置选项并输入源文件清单进行初始化，自动搜索系统已安装的 C/C++编译系统，针对每个输入源文件自动生成内部编译命令，调用 ASTFrontendAction 实例完成处理过程。

(2)从类 ASTFrontendAction 派生出子类 MyFrontendAction，针对每个输入源文件，调用 CreateASTConsumer 方法创建 ASTConsumer 实例，用于遍历抽象语法树；还创建类 ReWriter 的实例，用于在指定位置上修改 C++源代码内容，如插入文本(InsertText)、替换文本(ReplaceText)、删除文本(RemoveText)等；处理完每个输入源文件后，负责将 ReWriter 实例中缓存的包含插装语句的 C/C++源代码内容写回到指定输出文件。

(3)类 ASTConsumer 是用于访问抽象语法树的通用接口，从类 ASTConsumer 派生出子类 MyASTConsumer，并实现 HandleTopLevelDecl 接口函数；对于每个输入源文件的每个函数体，调用 RecursiveASTVisitor::TraverseDecl 方法，执行抽象语法树遍历操作；在实现上，应限制为访问且仅访问一次待插装 C++源代码的函数体。

(4)类 RecursiveASTVisitor 采用 Visitor 设计模式，能够以深度优先方式遍历访问特定类型的抽象语法树节点；从类 RecursiveASTVisitor 派生出子类 MyASTVisitor，通过重载 VisitStmt(Stmt*)方法实现对特定类型语句代码的访问和插装处理。

图 2 给出了 Clang 库的 Stmt 类图。对于源代码覆盖率统计插装，重点关注类 Stmt(语句)及其子类 IfStmt(IF 语句)和 CompoundStmt(复合语句)。Clang 使用 C++面向对象设计实现，可读性好，使得工具和代码库具有良好的可扩展性，能够快速增加新的语言特性。

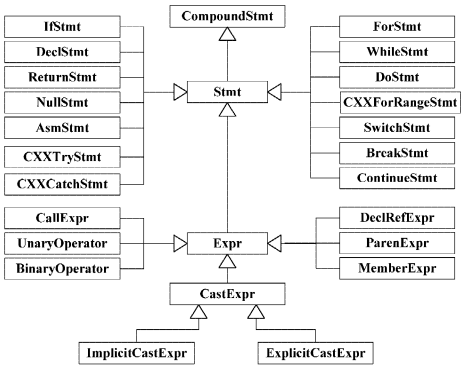

图 2 Clang 库的 Stmt 类图

3 C++源代码覆盖率统计插装方法

3.1 语句覆盖统计代码插装

由于 Clang 库将类 Expr(表达式)作为类 Stmt(语句)的子类(见图 2)，不容易区分 Expr 是独立语句还是位于表达式之中，如 CallExpr，UnaryOperator，BinaryOperator 等类型，因此不能直接增加语句覆盖统计的插装代码，而需要进行变通，分以下两种情况分别进行插装处理。

Case I Stmt 是单行语句的情况

当 IfStmt 语句的 Then 和 Else 分支以及 ForStmt，DoStmt，WhileStmt，CXXForRange-Stmt 的循环体只有单行语句且没有使用大括号{}时，直接使用 instru_stmt_single_line 函数(见图 3)在这个单行语句前后(对于 return 语句，必须在其前面)插入 STMT_INSTRU(index)，其中 index 是插装点序号，并使用大括号{}包围。当程序运行到插装点时，index 对应的计数值增加。

```
void MyASTVisitor::instru_stmt_single_line(Stmt* stmt)
{
    if (!isa<CompoundStmt>(stmt))
      调用 TheRewriter. InsertText()，在 stmt 语句后插入新的代码：
STMT_INSTRU( index );
    end if
}
```

图 3 Stmt 是单行语句的情况

Case II　Stmt 是复合语句的情况

在重载的 MyASTVisitor::VisitStmt(Stmt* s)函数(见图 4)中，对于 CompoundStmt(复合语句)，即用大括号{}包围的代码片段，在每个 stmt 语句前后(对于 return 语句，必须在其前面)插入 STMT_INSTRU(index)。

```
bool MyASTVisitor::VisitStmt(Stmt* s)
{
  if (isa<CompoundStmt>(s))
    for (Stmt* stmt:s 所属的语句序列)
      if (stmt 不是[CaseStmt,DefaultStmt,ContinueStmt,BreakStmt])
        调用 TheRewriter. InsertText(),在 stmt 语句后插入新的代码:
        STMT_INSTRU(index);
      end if
    end for
  end if
}
```

图 4　Stmt 是复合语句的情况

3.2　分支覆盖统计代码插装

分支覆盖又称判定覆盖，是指 if 或 while 语句的真分支和假分支至少执行一次；对于 switch 语句，则要求执行全部 case 和 default 语句。本文仅介绍 if 语句的分支覆盖插装方法，对于 while 和 switch 语句，可采取类似方法进行处理。

将 IfStmt 语句 if(condition)中的 condition 替换为：IF_STMT_INSTRU(index_true, index_false, condition)(见图 5)。当程序运行到分支覆盖插装点时，若 condition 值为 true，则 index_true 对应的计数值增加，否则 index_false 对应的计数值增加。当 IfStmt 包含两个以上条件且需要进行 MC/DC 覆盖率统计插装时，按 3.3 节的方法进行处理。

```
bool MyASTVisitor::VisitStmt(Stmt* s)
{
if (isa<IfStmt>(s))
    Stmt* Then=(cast<IfStmt>(s))->getThen();
    调用 TheRewriter. ReplaceText(),将 s 语句的"if"替换为:if (IF_ST-
    MT_INSTRU(index_true,index_false,(
    调用 TheRewriter. InsertText(),在 Then 语句之前,插入:)))
  end if
}
```

图 5　分支覆盖代码插装

3.3　MC/DC 覆盖统计代码插装

MC/DC 覆盖率要求判定中每一个条件的所有可能结果至少出现 1 次，每一个判定本身的所有结果也至少出现 1 次，每个入口与出口点至少被唤醒 1 次，并且每个条件都能独立影响判定的结果，即在其他条件不变的情况下，改变这个条件的值，能使得判定结果改变。MC/DC 线性地增加测试用例数量，对于包含 N 个简单条件的判定语句，至少需要 $N+1$ 个组合才能满足 MC/DC 覆盖测试要求。

MC/DC 覆盖统计插装方法如图 6 所示，在 MyASTVisitor::VisitStmt(Stmt * s)函数中实现，当遇到 IfStmt 类型的语句节点时进行处理。插装的 IF_CONDITION_MCDC_INSTRU(index,mcdc_condition)用来检测是否满足 MC/DC 组合测试条件 mcdc_condition。当程序运行到 MC/DC 覆盖插装点时，若 mcdc_condition 值为 true，则 index 对应的计数值增加，即指定的测试用例条件为真。

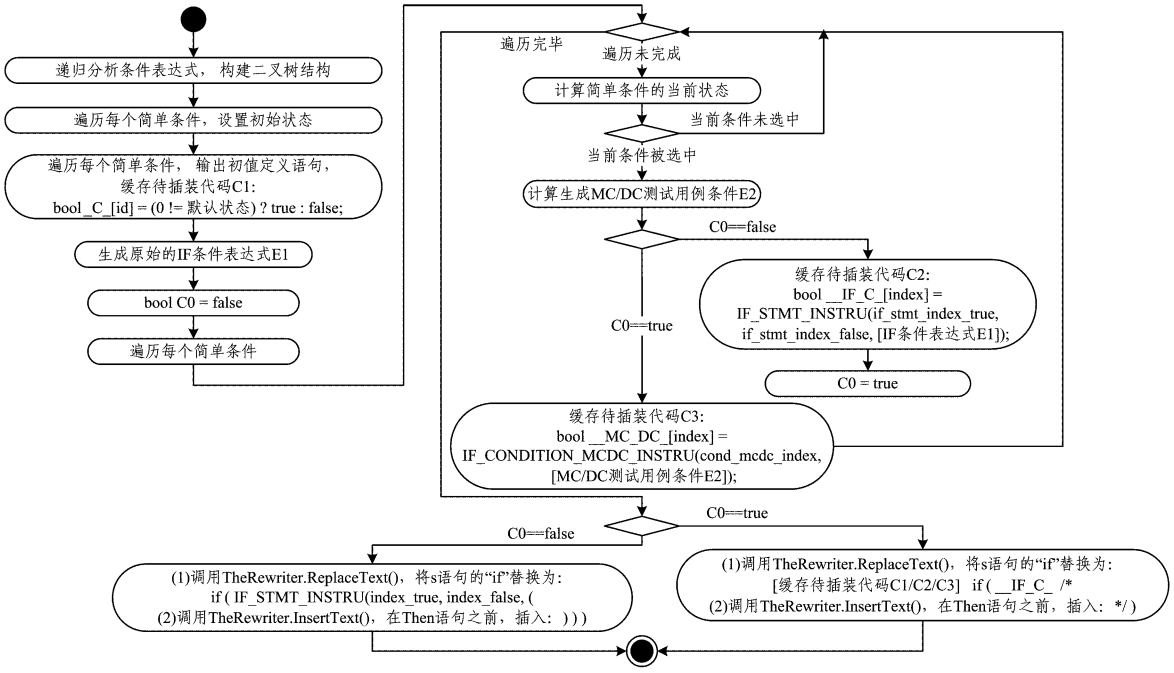

图 6　MC/DC 覆盖代码插装

考虑到测试用例的设计一般是按照最小测试用例集进行选取的，对于 MC/DC 覆盖插装和统计，当满足最小测试用例集时，认为达到 MC/DC 覆盖。本文采用文献[6]给出的方法来计算并生成符合最小测试用例集的 MC/DC 覆盖插装代码。

由于 C/C++条件表达式具有短路计算特性，当简单条件仍为函数调用或运算语句时，如果插装代码多次调用这些简单条件，则可能引发副作用，例如，if (++i>0 && --j>0)，其中简单条件(++i>0)和(--j>0)不能被多次调用。在代码插装过程中，将简单条件定义为临时变

量,在语句中将其替换为引用临时变量可以有效避免该问题。

4 原型工具与案例分析

在原有 RtCppTest 框架下,前端实现由 PUMA 库改为 Clang/LLVM 库,运行环境为 Kylin3.2/GCC 5.3.0/Clang 3.8.0,对两个大型工程软件代码进行插装处理(两个软件的源代码规模都是 27KLOC,插装点数量分别为 22023 和 23660,插装处理耗时分别为 186.581s 和 142.200s),输出的插装代码能够正确编译和运行,未发生运行崩溃现象(原来采用 PUMA 前端的工具可能崩溃)。图 7 所示为利用 Clang 前端生成的覆盖率统计插装代码片段截图,实现了语句、分支和 MC/DC 覆盖率插装。图 8 给出了 RtCppTest 自动生成的 C++源代码覆盖率详情报告。

```
STMT_INSTRU( 82443); //$ $[MC/DC]++ ( m_bUse_X_BufferPool && ! m_isXServer && ( 0 != getIBufferPoolHelper () ) )
bool __C_82449 = true; //$ $[C]-- (m_bUse_X_BufferPool )
bool __C_82450 = true; //$ $[C]-- (!m_isXServer )
bool __C_82451 = true; //$ $[C]-- (0 != getIBufferPoolHelper () )
bool __IF_C_82458 = IF_STMT_INSTRU(82445, 82446, (((__C_82449 = IF_CONDITION_INSTRU(82452, 82453, m_bUse_X_BufferPool)) &&
        (__C_82450 = IF_CONDITION_INSTRU(82454, 82455, ! m_isXServer))) &&
        (__C_82451 = IF_CONDITION_INSTRU(82456, 82457, 0 ! = getIBufferPoolHelper ()))));
bool __MC_DC_82459 = IF_CONDITION_MCDC_INSTRU(82459, (((__C_82449) && (__C_82450)) && (__C_82451)));
bool __MC_DC_82460 = IF_CONDITION_MCDC_INSTRU(82460, (((! __C_82449) && (__C_82450)) && (__C_82451)));
bool __MC_DC_82461 = IF_CONDITION_MCDC_INSTRU(82461, (((__C_82449) && (! __C_82450)) && (__C_82451)));
bool __MC_DC_82462 = IF_CONDITION_MCDC_INSTRU(82462, (((__C_82449) && (__C_82450)) && (! __C_82451)));
if (__IF_C_82458 /* m_bUse_X_BufferPool && ! m_isXServer && (0 ! = getIBufferPoolHelper()) */ )
```

图 7 C++覆盖率统计插装代码截图

行号	源代码行(80 字节/行)	插装序号	覆盖类型	覆盖统计
797	if (m_bUse_X_BufferPool && ! m_isXServer && (0 != getIBufferPoolHelper()))	82443	语句	7756
797	^^^ m_bUse_X_BufferPool && ! m_isXServer && (0 != getIBufferPoolHelper ())	82445	分支	2215
797	^^^ !(m_bUse_X_BufferPool && ! m_isXServer && (0 != getIBufferPoolHelper ()))	82446	分支	5541
797	^^^ __MC_DC_82459 : (((__C_82449) && (__C_82450)) && (__C_82451))	82459	MCDC	2215
797	^^^ __MC_DC_82460 : (((!__C_82449) && (__C_82450)) && (__C_82451))	82460	MCDC	4432
797	^^^ __MC_DC_82461 : (((__C_82449) && (! __C_82450)) && (__C_82451))	82461	MCDC	1109
797	^^^ __MC_DC_82462 : (((__C_82449) && (__C_82450)) && (!__C_82451))	82462	MCDC	0

图 8 C++源代码覆盖率详情截图

结束语 开源社区提供了丰富的高品质软件代码库,为科学研究、工程应用提供了强大的基础平台,可以来快速构建功能强大的实用工具。本文采用 Clang/LLVM 库构建了一种 C++源代码覆盖率统计插装方法,将其集成到原有覆盖率统计分析框架内,并直接应用于实际的工程软件,取得了较好的效果。

参 考 文 献

[1] AN J X,ZHU J. Software Reliability Modeling with Integrated Test Coverage [C]// Proceedings of the 4th IEEE International Conference on Secure Software Integration and Reliability Improvement. Singapore,2012:106-112.

[2] YANG Q,LI J,WEISS D,et al. A Survey of Coverage-Based Testing Tools[J]. The Computer Journal,2009,52(5):589-597.

[3] MAHRENHOLZ D,SPINCZYK O,SCHRÖDER-PREIKSCHAT, et al. Program Instrumentation for Debugging and Monitoring with AspectC++ [C]// Proceedings of Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Washington DC,2002:249-256.

[4] URBAN M,LOHMANN D,SPINCZYK O. The Aspect- Oriented Design of the PUMA C/C++ Parser Framework [C]// Proceedings of the 9th International Conference on Aspect-Oriented Software Development. New York,NY,USA,2010: 217-221.

[5] AN J X,WANG G Q,LI S F,et al. Dynamic Evaluation Method Based Multi-Dimensional Test Coverage for Software Testing [J]. Journal of Software,2010,21(9):2135-2147. (in Chinese)
安金霞,王国庆,李树芳,等. 基于多维度覆盖率的软件测试动态评价方法[J]. 软件学报,2010,21(9):2135-2147.

[6] DUAN F L,WU X,ZHANG F,et al. Rapidly Generating Algorithm for Minimum Test Case Set on MC/DC[J]. Computer Engineering,2009,35(17):40-42. (in Chinese)
段飞雷,吴晓,张凡,等. MC/DC 最小测试用例集快速生成算法[J]. 计算机工程,2009,35(17):40-42.

[7] LI S F,AN J X,ZHENG P F,et al. An Approach to Rapid Test Coverage Analysis for Large-scale Real-time Software[J]. Journal of Southwest University of Science and Technology(Natural Science Edition),2013,28(3):89-94. (in Chinese)
李树芳,安金霞,郑鹏飞,等. 面向大型实时软件的测试覆盖率快速分析方法[J]. 西南科技大学学报(自然科学版),2013,28(3): 89-94.

[8] LI S F,CHEN X,AN J X,et al. Real-Time Software Memory Error Analysis with C++ Code Instrumentation[J]. Journal of Frontiers of Computer Science and Technology,2014,8(6):704-711. (in Chinese)
李树芳,陈霞,安金霞,等. 采用 C++代码插装的实时软件内存错误分析[J]. 计算机科学与探索,2014,8(6):704-711.