源代码分析

宫云战　邢颖　肖庆等　著

科学出版社

源代码分析

宫云战　邢　颖　肖　庆　等　著

科学出版社
北京

内 容 简 介

目前，源代码分析是软件工程领域的必备方法之一，有着强烈的工程需求和实用价值，已成为国际学术界和工业界的一个热点。本书从源代码分析的基本概念开始，将其中所涉及的重要的技术和应用——抽象解释、符号计算、区间运算、路径敏感分析、抽象内存建模、上下文分析、程序切片、路径计算和约束求解等，结合大量的实例进行由浅入深的介绍和讲解；同时，在本书的最后专门介绍应用源代码分析技术所研发的一些常用测试工具，并重点介绍两款静态分析工具——DTS、CTS。

本书是软件工程领域的专业书籍，可供从事软件工程领域相关工作的研究人员学习和参考。

图书在版编目(CIP)数据

源代码分析/宫云战等著. —北京：科学出版社，2018.1

ISBN 978-7-03-055188-7

Ⅰ.①源… Ⅱ.①宫… Ⅲ.①源代码-分析 Ⅳ.①TP311.52

中国版本图书馆 CIP 数据核字(2017)第 270213 号

责任编辑：张艳芬 朱英彪 / 责任校对：桂伟利

责任印制：张 伟 / 封面设计：蓝正设计

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

北京中石油彩色印刷有限责任公司印刷

科学出版社发行 各地新华书店经销

*

2018 年 1 月第 一 版 开本：720×1000 B5

2018 年 1 月第一次印刷 印张：17 3/4

字数：358 000

定价：98.00 元

(如有印装质量问题，我社负责调换)

前　言

一般软件工程师所开发的软件，如不经过任何测试，其缺陷密度可达每千行代码 100 个缺陷，而经过个人软件工程（PSP）、团队软件工程（TSP）训练的软件工程师所开发的软件大约是每千行代码 50 个缺陷。软件重用、软件过程控制与管理、软件测试、缺陷管理等技术是减少和发现软件缺陷的主要手段。

软件测试是软件开发过程很重要的一步，目前已有数十种软件测试方法，它们从不同的角度验证软件的某些功能或非功能属性，或发现不同的软件缺陷。对任何一种测试技术而言，测得准、测得快、测得多是其研究领域永恒的主题。通过众多的软件测试，逐步降低软件的缺陷密度，直至满足其需求。

软件测试是需要大量经费的。统计表明，美国的软件开发，有 53％的费用投入在软件测试上，美国国家航空航天局甚至高达 90％以上，而我国的软件开发在软件测试上的投入一般不超过 20％。这也从一个方面说明，为什么我国每年会有十几万到几十万个软件产品推出却没有一个在国际上知名的软件，为什么我国基础软件的产值会这么少。原因就是对软件测试的认识不够，对软件测试的投入太少。目前，制约软件测试技术发展的重要因素之一是测试的自动化问题，手工或半自动的测试需要太多的人力和时间，致使很多软件没有经过严格的测试就进入市场了。

代码分析是软件测试的基本方法，就是对代码进行各种计算，以期发现其中的缺陷、安全漏洞等。代码分析的理论研究已有 50 多年，2000 年以后，该技术得到快速发展。以该技术为基础，产生了上百个软件测试工具，现在每年的产值达到上百亿美元。

代码分析技术涉及的内容较多，包括抽象解释技术、符号执行技术、路径敏感技术、约束求解技术、上下文分析技术、抽象内存建模技术、区间计算技术、路径计算技术、程序切片技术、循环建模技术和缺陷模式库技术等，通过这些技术的综合使用可以开发出单元覆盖测试工具、集成测试工具和代码扫描工具等。本书对上述技术一一作了论述。

作者及团队在过去的 20 年中一直致力于软件测试方法的研究和软件测试工具的研发，在代码分析领域发表了 200 多篇论文，拥有 50 多项专利，所研发的两款代码分析工具——缺陷检测系统 DTS、代码测试系统 CTS 已在国内拥有数百个用户。本书是作者多年研究成果的总结，可为这两款工具的使用提供理论基础。

本书由宫云战教授提出写作方案，并进行统稿。邢颖博士具体负责了本书的

出版工作，从内容组织到人员分工做了大量工作。中国科学院计算技术研究所李炼研究员、北京化工大学赵瑞莲教授以及在北京邮电大学学习过的邢颖博士、肖庆博士、金大海博士、王雅文博士、张大林博士、杨朝红博士和张旭舟博士参与了本书的撰写。

限于作者水平，书中难免存在不足之处，敬请读者批评指正。

作　者

2017 年 4 月

目　录

第 1 章　源代码分析概要

源代码分析的应用从起初的编译领域已经渗透到软件工程的各个领域。目前，源代码分析主要用于缺陷检测、安全漏洞检测、恶意代码检测、源代码相似性检测、代码质量审查、程序理解、性能分析、故障定位、软件调试、逆向工程和中间件等[1]。本章从源代码分析的基本概念开始，通过叙述常用的图、树、表、运算及各种分析方法，为后续介绍奠定基础。

1.1　基本概念

1.1.1　源代码

源代码也称源程序，是指未编译的、按照一定的程序设计语言规范书写的文本文件，是一系列人类可读的计算机语言指令。因此，源代码是静态的、文本化的、可读的（人能够理解的）计算机程序，该程序可进一步被编译为可执行文件[2]。

由于大量的商业软件获取其开发环境和源代码比较困难，实际的代码分析对象可能是中间代码（intermediate code）、目标代码或可执行代码。中间代码是源程序的一种内部表示，不依赖目标机的结构，易于机械生成目标代码的中间表示。中间代码是可移植的（与具体目标程序无关），且易于实现目标代码优化。目标代码是源代码经过编译程序产生的能被 CPU 识别的二进制代码。可执行代码是将目标代码链接后形成的可执行文件。源代码编译过程如图 1-1 所示，以 C/C++为例，经过预处理过程形成中间文件；经过编译汇编后形成的文件为目标代码；经过链接后形成的代码为可执行代码。

图 1-1　源代码编译过程

1.1.2　源代码分析

源代码分析是从源代码或源代码生成的相关构件（如位文件等）中抽取程序信

息并根据需要进行计算的过程。

代码分析技术可以分为静态分析和动态分析。如果一种分析技术可在代码编译阶段发挥作用，那么这种技术就是静态分析技术；如果一种分析技术需要实际运行程序才能得到结果，那么就是动态分析技术。无论哪一种分析技术，所提取的相关信息都应与语言的实际语义保持一致，并有助于人们理解源代码的含义。

1.1.3 分析过程

无论使用何种分析技术，所有针对软件源代码分析的工具，其工作方式大致相同。源代码分析的一般过程如图 1-2 所示，它们都接受源代码作为输入，为待分析程序构建可计算的模型，结合大量软件相关属性知识来对这个模型进行分析，并最终向用户提交其分析结果[1,3]。

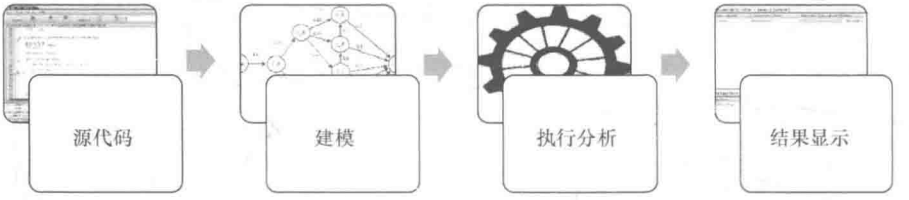

图 1-2 源代码分析的一般过程

1.1.4 源代码建模

代码分析技术所提取的信息通常可通过模型来表示，将其转换为一种程序模型，即一组代表此代码的数据结构。因此，源代码分析可视为状态空间中的模型实例提取过程。源代码分析所提取的信息往往是更高层次的抽象语义信息。在源代码分析中，除了宏观上的语法树、控制流图、缺陷模式等模型和数据结构，常见的微观领域的建模还有内存建模、函数建模和循环语句处理建模等[4]。

1. 内存建模

针对 C 语言的源代码分析，内存建模一般包括指针、结构体、数组和字符串等的建模，这些模型用来对被分析的源代码的数据类型进行抽象表征，进而支持相对精确的分析。

指针分析的精度直接影响后续的程序分析。域敏感性用来描述指针分析是否需要区分结构体对象的不同域成员。完整的指针建模本质上是建立一套非标准类型系统，该类型系统由类型和类型规则组成。其中，类型用于描述存储对象的指向情况，存储对象包括程序中出现的标量、数组、结构体和堆对象；类型规则规定了指针分析应该具有的性质及其计算规则，同时保证指针分析的结果可以保守地描述

程序实际运行时的动态存储构造。

数组对象的分析十分复杂，尤其是在 C/C++语言中存在着大量的别名关系、变量类型之间的层次关系等关联关系，如果不能有效、完整地表示这些复杂关系，那么分析的精度将会下降。针对数组的内存建模，其难点主要表现在数组的抽象内存表示、数组元素计算、数组与指针的结合、数组与字符串数组的结合等方面。

字符串是通用编程语言中的一种基本数据类型，是一种特殊的字符数组，应用相当普遍。指针的使用使得针对字符串的内存建模更加困难。操作字符串变量的程序经常使用一组库函数，如字符串比较函数、字符串查找函数和字符串复制函数等。目前，常用的字符串处理方法有数组模型方法和约束求解方法。

(1) 数组模型方法是使用数组模型对库函数进行模拟操作，把字符串中的每一个字符元素当做取值范围为 0～255 的整型变量来处理，并且用一组由赋值语句组成的逻辑公式来模拟字符串函数的操作。

(2) 约束求解法是构建字符串变量约束条件，用一组约束条件来刻画字符串变量的属性。给定一个字符串变量的一组约束条件，会输出满足所有约束条件的字符串，或者报告该约束条件是不可满足的。

2. 函数摘要建模

函数间分析主要关注函数调用引起的上下文变化，其分析结果可以提高调用函数(caller)内的分析精度。函数调用对调用上下文的影响可以归纳为如下三种：

(1) 函数调用对调用上下文的数据流更新情况的影响，包括传引用式形参及全局变量的数据流更新。

(2) 函数返回值、函数调用引起的其他数据流及控制流的变化。

(3) 调用点处必须满足的数据约束条件。

因此，可以根据上述函数调用的影响和实际分析需求对函数摘要进行建模。

3. 循环语句处理建模

在源代码分析过程中，循环语句分析往往难以确定循环具体的执行次数，如果循环每迭代一次都当做一条新的路径，那么将会引起路径爆炸问题。因此，结合分析给出一个有效的循环语句处理模型是源代码分析的必然选择。传统的 0-1 模型和 0-K 模型并不能满足实际计算的需要，主要原因是这种静态的模型往往和实际不符，而本书提出并采用的动静 $K+1$ 模型是一个可行的方法。

1.2 语法与语义分析

1.2.1 语法分析

语法分析(syntax analysis)是最基本的代码分析,是代码分析的一个基本阶段。语法分析的任务是在词法分析的基础上将单词序列组合成各类语法单元,进而判断源程序在结构上是否正确,按照相应源代码的语法规则进行语法检查。完成语法分析任务的程序称为语法分析器,或语法分析程序[5]。

语法分析就是按照文法的产生式,识别输入符号串是否为一个句子。对一个给定的文法及输入串,要判断是否能从文法的开始符号出发推导出这个字符串;或者从概念上讲,就是要建立一棵与输入串相匹配的语法分析树。

语法分析的方法可分为自下而上和自上而下两种。自下而上分析法的基本思想是从输入串开始,逐步进行"归约",直至归约到文法的开始符号;或者说,从语法树的末端开始,步步向上"归约",直到根节点。所谓归约,就是根据文法的产生式规则,把产生式的右部替换成左部符号。自上而下分析法是从文法的开始符号出发,根据文法的产生式规则正向推导出给定句子的一种方法;或者说是从树根节点开始,往下构造语法树,直到建立每个叶子节点的分析方法。

1.2.2 抽象语法树

抽象语法树(abstract syntax tree,AST)在更多情况下是先创建的程序抽象模型,是源代码分析的一个很好的起点。抽象语法树的节点对应于解析树上的源代码。抽象语法树是创建更复杂的图结构(模型)的基础,如控制流图(control flow graph,CFG)等,这些图可用于不同类型或不同复杂程度的分析算法。

程序的语法树是一种树型数据结构,该数据结构充分地说明按照语法怎样看待程序的各个部分,语法树可以通过语法分析得到。完全符合文法规格要求的语法树(或分析树)对于进一步处理并不是最适合的,实际应用时经常会根据需要对语法树作适当的简化和修改,这种修改后的形式称为抽象语法树。为了强调它们之间的区别,把语法树称为具体语法树,相应文法称为该语言的具体语法。

图 1-3 是表达式 b * b—4 * a * c 对应的分析树,表达式所用的文法与 Java 语言相似:

```
expression→expression '+' term | expression '-' term | term
term→term '*' factor | term '/'factor | factor
factor→identifier | contant | '('expression ')'
```

图 1-4 以 AST 形式表示了同样的表达式。

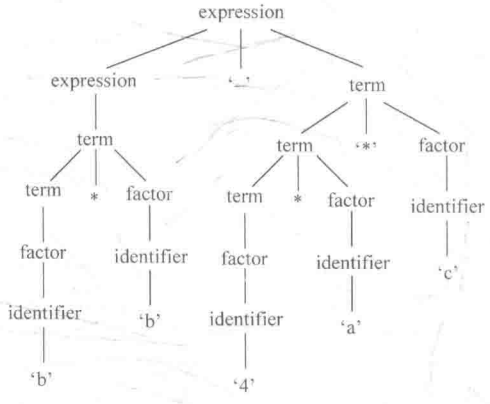

图 1-3　表达式 b * b—4 * a * c 的分析树

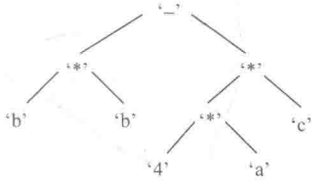

图 1-4　表达式 b * b—4 * a * c 的抽象分析树

出于后期建立控制流图、数据流图和调用图(call graph)等的需要，本章所涉及的抽象语法树忠于 Java 语言语法的巴克斯范式(Backus normal form，BNF)原始表达式，仅简化掉一些表达式中的单枝树，抽象程度介于图 1-3 和图 1-4 之间，更接近于图 1-3。下面给出一个简单的例子。

源程序为 Welcome. java，代码如下：

```
public class Welcome {
    public static void main(String args[]) {
        b=b*b-4*a*c;
    }
}
```

图 1-5 是上述源代码的抽象语法树，该抽象语法树生成单元通过 Java CC 工具辅助生成。

大多数程序设计语言使用 for、do、if 等固定的字符串作为语法符号来表示某种构造，这些字符串统称为关键字。除了关键字，变量名、数组名和函数名等字符串统称为标识符，因此关键字也称为保留字。

抽象语法树上每个标识符的出现都是独立的，互相之间没有任何联系。例如，对于一个表达式 $i=1$，从语法树上很难了解变量 i 是什么类型，和前面某个 i 是否为同一个变量，这些信息一般都需要符号表来提供。

1.2.3　符号表

符号表被用来记录标识符(名字)的作用域、声明使用信息，它将每个标识符与

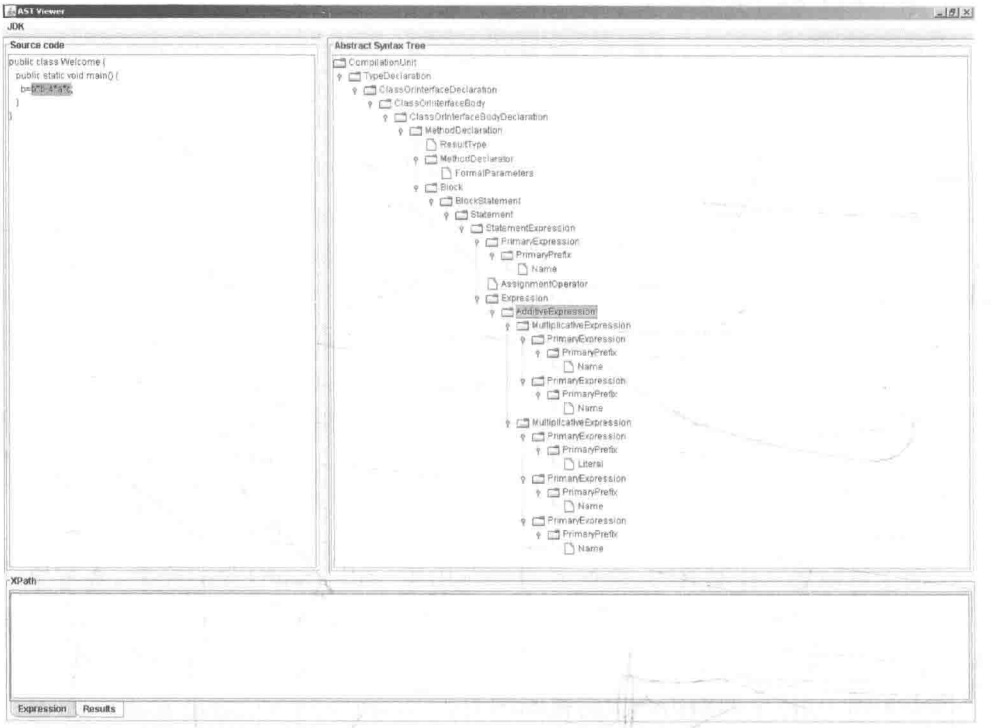

图 1-5 一个抽象语法树例子

其声明位置进行映射。符号表中的每一个记录代表一个标识符。在 Java 语言中，标识符可以用来表示类(包括接口和枚举类型等)的名称、方法名称和变量名称。另外，Java 程序中每个标识符都有一个在其内可见的作用域，不同的作用域中允许出现相同名称的标识符。作用域之间是包含的关系，形成一个树状数据结构，例如：

```
package XQ;                          SourceFileScope(XQ):(classes:Test,TestCase)
public class Test{                   ClassScope (Test):(methods:f)(variables:y,x)
   public int x=0;                   MethodScope(f):(variables:y,x)
   public float y=0;                 LocalScope:(variables:b,c,a)
   public void f(int x,int y){       LocalScope:(variables:f,d,e)
     int a,b,c;                      LocalScope:(variables:f,d,e)
     {int d,e,f;}                    ClassScope (TestCase):(methods:f,g)(varia-
                                          bles:a)
     {int d,e,f;}                    MethodScope(g):(variables:y,x)
   }                                 LocalScope:(variables:)
}                                    MethodScope(f):(variables:)
```