分析测试工具结题报告

1. 姓名：谭丽娟，学号：MF1833067
2. 工具名称及功能描述
3. 工具名称

Soot

1. 功能描述

通过它可以进行过程内和过程间的分析优化，以及程序流图的生成，还能通过图形化的方式输出，让用户对程序有个直观的了解。尤其是做单元测试的时候，可以很方便的通过这个生成控制流图然后进行测试用例的覆盖，显著提高效率

1. 需求分析
2. Java类的源代码

public class Triangle {

public String check(int a, int b, int c){

String result = "";

if(a == b && b == c){

result = "该三角形为等边三角形";

}else if(a == b || a == c || b ==c){

result = "该三角形为等腰三角形";

}else if(a + b > c && a + c > b && b + c > a){

result = "该三角形为不等边三角形";

}else{

result = "这不是一个三角形";

}

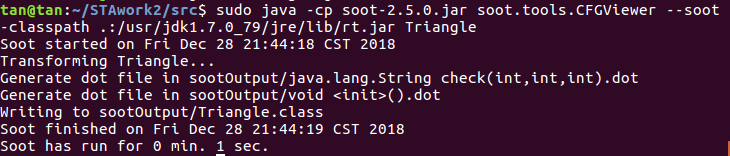
return result;

}

}

1. 输入

java -cp soot-2.5.0.jar soot.tools.CFGViewer --soot-classpath .:/usr/jdk1.7.0\_79/jre/lib/rt.jar Triangle



1. 输出

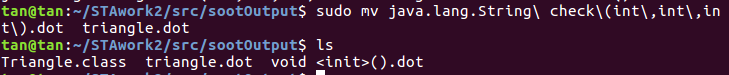
输出一个sootOutput文件



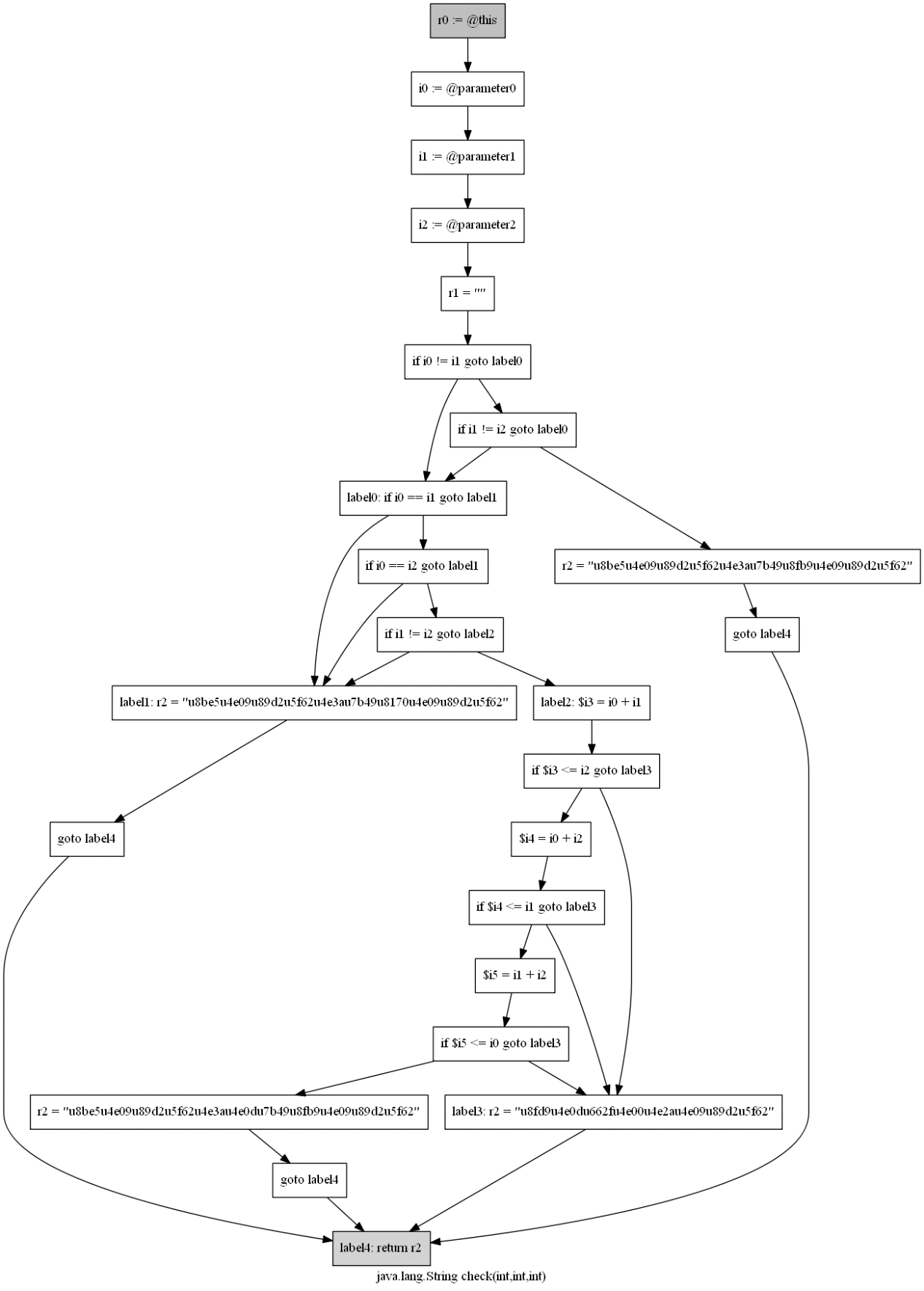
C:\Users\TANLIJ~1\AppData\Local\Temp\1546004893(1).png

1. 方法

根据输出的.dot文件，如果系统中没有安装graphviz，先安装一下graphviz，然后将java.lang.String check(int,int,int).dot文件改名为triangle.dot文件。



然后，在sootOutput文件夹下输入命令dot -Tpng triangle.dot -o Triangle.png 将.dot文件转化为可视化文件

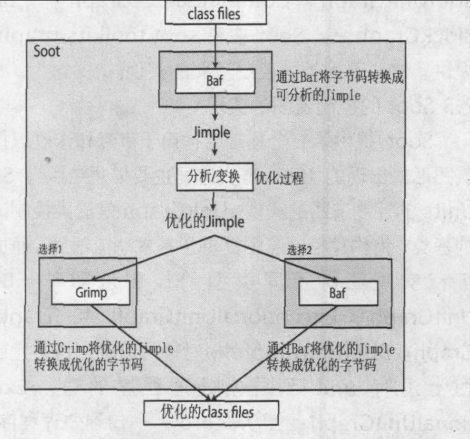


1. 工具名称及功能描述
2. 技术路线

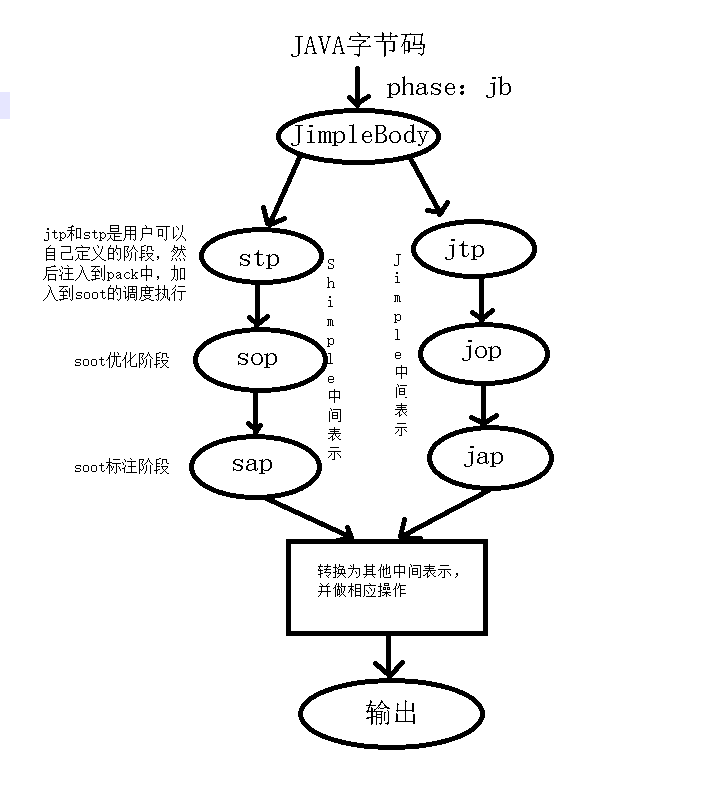
Soot的输入时多源的，可以是java的字节码。Soot提供了四个中间表示法，通过将源文件转换为中间表示，基于这些中间表示传入不同的变换类来进行分析，优化或者再变换，另外还直接提供一组直接用于优化Java字节码的API。Soot的扩展机制以Pack为中心，一个Pack包括若干个变换用户可以自行设计新的变换，将其加入到soot的调度执行过程中以实现特定的功能，如输出为dot文件格式。

1. 详细设计
2. 流程图

Soot读入Java字节码，然后通过Baf将字节码转换为可分析的Jimple，然后再进入分析变换过程，接着可以再转换为Grimp形式或者Baf形式来输出或者再转换为java字节码输出优化的Java字节码，这就是Soot在数据层面抽象的处理过程。



Soot的执行过程被分成了好几大步，每一大步被称为一个pack。第一步是把输入的bytecode (.class)或者.java 文件或者.jimple 翻译成Jimple code。再把生成的Jimple作为剩下packs的输入。"函数中分析(intra-procedure analysis)"执行流程示意如下：



1. 各模块核心功能，算法

有四个核心模块：Baf，Grimp，Jimple和Shimple

1. Baf - 基于栈的bytecode

传统的JVM bytebode是基于栈操作的指令集（Dalvik 基于寄存器操作），与之对应的Baf同样如此。那Baf抽象了什么呢？两个，忽略了constant pool(常量池)和bytecode指令中的type依赖。在bytecode中对不同保留类型，如int和float，的同一操作（如add），有不同的指令。这是因为在计算机中整形和浮点型的表达方式是不一样的，在底层实现时无法让两个操作符分属于这两种不同类型，也就是需要不同的指令对应不同的数据类型的操作。我们做分析时不用在意它到底调用的什么类型的指令，不对int还是float做细致区分，只要知道它是个数且知道是对这数的什么样的操作就行了。Baf因此用于在bytecode层面上的分析。

1. Jimple - 紧凑、无栈、类型化的三地址代码中间表示法

Jimple是Soot的核心，是四种IR中最重要的。Soot能直接创建Jimple码，也可由Java sourcecode或者bytecode转化翻译而来。bytecode会被翻译成untyped Jimple，再通过type inference 方法对局部变量加上类型。翻译的重要一步是对表达式作线性化使得每个statement只能最多refernce 3个局部变量或者常量（没懂。。）。相对于bytecode的200多种指令，Jimple只有15条，分别对应着核心指令的 NopStmt, IdentityStmt, AssignStmt；函数内控制流指令的IfStmt, GotoStt, TableSwitchStmt和LookUpSwitchStmt，函数间控制流的InvoeStmt, ReturnStmt, ReturnVoidStmt, 监视器指令EnterMonitorStmt和ExitMonitorStmt，最后处理异常ThrowStmt和退出的RetStmt。

1. Grimp -- Jimple聚合版

可以构造树形结构，更接近于java源代码，适用于反汇编并便于阅读

1. Shimple -- Static Single Assignment 版的Jimple