第34卷 第2期 计 算 机 工 程 2008年1月
Vol.34 No.2 Computer Engineering January 2008
·软件技术与数据库· 文章编号：1000—3428(2008)02—0069—03 文献标识码：A 中图分类号：TP311

一种Java字节码优化框架

张 昱，刘玉宇
(中国科学技术大学计算机科学技术系，合肥 230027)

摘 要：Soot是一种用于优化、审查Java字节码的独立工具，也是一个在字节码上开发优化和变换的框架。为指导用户在Soot上快速开展工作，该文总结Soot的主要数据结构、流程及优化框架的组织管理，并通过实例剖析Soot的工作流程和实现机制，简述了Soot的扩展方法及已有应用。

关键词：优化；包装；变换器

Java Bytecode Optimization Framework

ZHANG Yu, LIU Yu-yu
(Department of Computer Science, University of Science & Technology of China, Hefei 230027)

【Abstract】 Soot can be used as an independent tool to optimize or inspect Java bytecode, as well as a framework to develop optimizations or transformations on bytecode. In order to guide users to work with Soot fleetly, the main data structures and flow of Soot and the methodologies of the optimization framework are summed up. An example is given to anatomize the implementation mechanisms of Soot. Extension steps of Soot and related application are stated.

【Key words】 optimization; pack; transformer

Java是一种先将源程序编译成字节码，再在虚拟机上解释执行字节码的面向对象语言。Java的平台独立、类型安全及垃圾回收等特性使得其被广泛应用。但与C/C++相比Java程序的运行要慢得多，为此许多机构已经或正在开展对Java程序优化的研究。为简化Java字节码优化工具的开发，McGill大学的Sable研究组研制了名为Soot的工具[1-3]。它为字节码的分析、优化、反编译、注解等提供一个可扩展的框架；它本身实现了不少优化变换算法，故也可以单独使用。最新的Soot版本是2.2.3，2.0版以上的Soot还包含一个Eclipse插件。本文通过剖析Soot 2.2.3，总结其特色、流程和扩展方法，期望能指导用户使用Soot开展自身的字节码优化工作。

1 Soot概览

Soot的总体结构如图1所示。

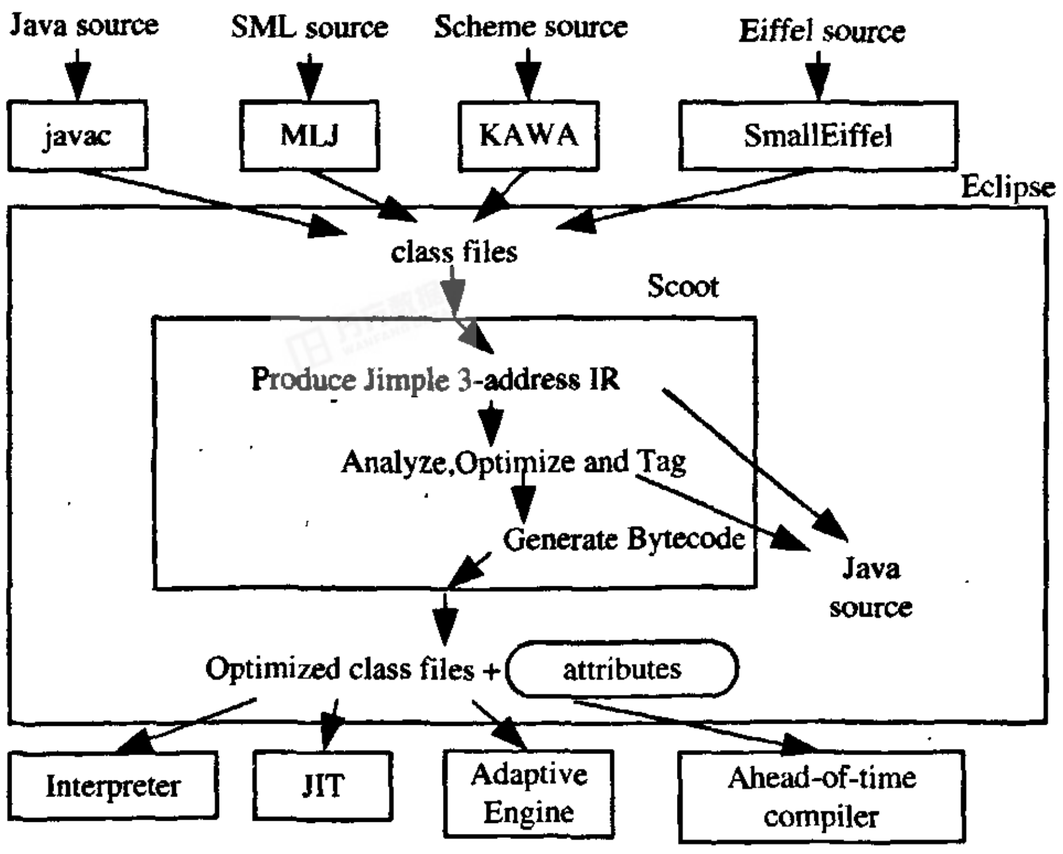

图1 Soot的总体结构

Soot以Java源程序或字节码(class文件)等为输入，以优化的class文件为输出；其分析和变换建立在内部的中间表示(IR)之上，它们可以是局部的(仅对某方法体)，也可以是全局的(对整个程序)。早期Soot调用javac将源程序编译成字节码再处理；近来则用Polyglot编译Java源程序(见JavaToJimple类)，再构造Jimple IR。另外，Soot本身不直接生成字节码，而是先生成Jasmin汇编码，再调用Jasmin汇编器生成字节码(见JasminOutputStream类)。

Jimple是Soot的核心IR，是一种类型化的三地址无栈表示。此外，Soot还提供4种IR用于分析和变换：(1)Baf是基于栈的字节码的简洁表示；(2)Grimp是Jimple的聚合版本，适于反编译和代码审查；(3)Shimple是Jimple的SSA(static single assignment)变种；(4)Dava是适用于反编译的结构化表示。这些IR之间可以相互转化。

要在Soot上开展优化变换，必需进一步分析Soot源码，以了解其主要的数据结构、优化框架的组织和已有的变换及其实现机制。

2 Soot剖析

Soot 2.2.3有近2 000个java文件，按其是否为自动生成分别存放在generated和src目录中，文件组织繁杂。所有类的包路径以soot为根，soot.Main是主类。soot包中定义有为各种IR共享的基本类，并含有13个子包：baf, dava, grimp, jimple, shimple包分别包含对应IR的表示及变换类；javaToJimple包实现Java到Jimple的转化；options包提供对

基金项目：Intel公司研究基金资助项目
作者简介：张 昱(1972－)，女，副教授，主研方向：程序设计语言理论和实现技术，XML数据管理；刘玉宇，硕士研究生
收稿日期：2007-02-05 **E-mail**：yuzhang@ustc.edu.cn

命令行参数的分析；tagkit 包支持对字节码的注解；coffi 包来自 Coffi 工具；toolkits 包提供如控制流图 CFG、流分析等工具包；tools 包存放 Soot 扩展，它们可以取代 soot.Main 被执行；util, xml 包提供其他有用的类。

类 soot.G 收集 Soot 的全局变量，大部分为类 Singletons(由 make_singletons 自动生成)收集的 singleton 对象，可通过相应类中的 v 方法取得。

2.1 主要数据结构

类 Scene, SootClass, SootMethod, SootField 和 Body 是 Soot 的主要数据结构。Scene 表示整个分析变换的环境，其主要的域如表 1。

表 1 Scene 类的主要域

域名	含义
classes	所有类的链
mainClass	待分析程序的主类
applicationClasses	用于分析的应用类形成的链
libraryClasses	库中类形成的链
nameToClasses	类名到类的映射
sootClassPath	命令行参数或环境变量中指定的 classpath
activeHierarchy	类层次图，Hierarchy 类型
activeFastHierarchy	类层次图，FastHierarchy 类型
activeCallGraph	方法调用图
activePointsToAnalysis	PAG 图(Pointer Assignment Graph)
activeSideEffectAnalysis	side-effect 信息
entryPoints	程序入口点

SootClass 表示装入 Soot 或由 Soot 创建的单个类。SootMethod 表示类中的单个方法。SootField 表示类中的一个域。Body 表示一个方法体，不同的 IR 由它派生不同的子类，见图 2。

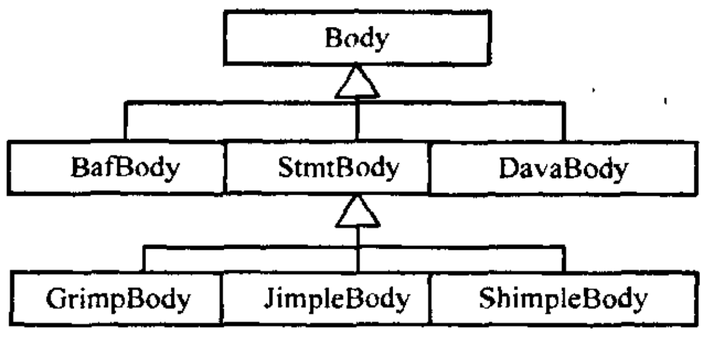

图 2 Body 及其子类

Body 类包含域 localChain, trapChain 和 unitChain，分别代表局部变量、Trap 和 Unit 3 个链。Trap 是描述异常捕获的接口。Unit 是描述代码片段的接口，子接口 Inst 和 Stmt 分别表示 Baf IR 上的指令和其它 IR 上的语句。许多类由 Unit 的抽象子类 AbstractUnit 派生并实现 Inst 或 Stmt 接口，表示具体的指令或语句类。

接口 ValueBox 和 UnitBox 分别提供对值引用和语句引用的封装。值的公共接口是 Value。通过 Unit 可以取得语句中定义或使用的值；取得跳到该 unit 的各 unit，以及由该 unit 跳到的各 unit；还可以查询分支行为。

2.2 优化框架组织

为便于扩展，Soot 采用以 Pack 为中心的优化框架。Pack 是一套优化变换的包装类，其 opts 域收集各变换对象(为 Transform 实例)，方法 apply 调用 internalApply 执行所包装的各种变换。Pack 有 BodyPack 和 ScenePack 两个子类，继承前者的类包装方法体内的变换，继承后者的类则包装全局变换。各 Pack 实现类需重写 internalApply 方法。

类 Transform 维护二元组<阶段名，变换器>。以 Transformer 为基类的变换器是实现从字节码到 IR 以及 IR 间的转化，或基于 IR 的分析、优化、反编译、标注等的类。和 Pack 类似，它有 BodyTransformer 和 SceneTransformer 两个子类。变换由类中的 transform 方法调用 internalTransform 完成，各 Transformer 实现类需重写 internalTransform 方法。不论是全局还是局部变换，一般都是对方法体中的语句逐条分析变换的，故操作的核心是 Unit 实例。

各种 Pack 由类 PackManager 管理，其 init 方法负责创建各 Pack 实例对象，并为之添加变换器。表 2 列举了 Soot 中的部分 Pack。

表 2 部分 Pack

Pack 名	所属的 Pack 类	说明
jb	JimpleBodyPack (BodyPack 的子类)	创建 Jimple 体
jj	JavaToJimpleBodyPack (BodyPack 的子类)	实现 Java 到 Jimple 的转换
cg	CallGraphPack (由 ScenePack 派生)	调用图生成、指针分析、类层次分析(CHA)
wstp	ScenePack	全局 Shimple 变换包
wsop	ScenePack	全局 Shimple 优化包
wjtp	ScenePack	全局 Jimple 变换包
wjop	ScenePack	全局 Jimple 优化包
wjap	ScenePack	全局 Jimple 注解包
jtp	BodyPack	Jimple 变换包
jop	BodyPack	Jimple 优化包
jap	BodyPack	Jimple 注释包
tag	BodyPack	代码属性 tag 聚集包

2.3 总体流程

soot.Main.run(args)反映 Soot 的主要流程，步骤如下：

(1)调用 Options.v().parse(args)解析参数并设置全局选项对象。

(2)调用 Scene.v().loadNecessaryClasses()装载必需的类，包括基本类、库中的类和待分析应用中的类，并设置全局选项对象。

(3)调用 PackManager.v().runPacks()执行各 Pack 变换，步骤为：

1)由选项决定是否执行行号变换(LineNumberAdder)；

2)由选项决定是否调用 runWholeProgramPacks 进行全局变换，它除执行名为 cg 的 Pack 外，还执行与当前 IR 有关的全局 Pack；

3)调用 retrieveAllBodyes 确认方法体全部载入；

4)由选项决定是否预处理 DAVA 类和设置交互模式；

5)执行各方法体内的变换 runBodyPacks()；

6)处理内部类。

(4)调用 PackManager.v().writeOutput()输出结果。

3 一个例子：基于 Spark 的 side-effect 分析

3.1 Spark 分析

Spark(Soot pointer analysis research kit)[2]是由 Spark Transformer(SceneTransformer 的子类)实现的一种指针分析。在名为 cg 的 Pack 中包装有 SparkTransformer 及对应的阶段名 cg.spark。cg 中还有两个阶段：cg.cha 为类层次分析；cg.paddle 在 Soot 中只对应 PaddleHook 接口，需安装 paddle 插件[3]才能实现上下文相关的指针分析。

Spark 通过构建 PAG 图，运用传播算法计算图中每个结点的 P2Set(Points-to Set)，来分析程序中的指针在运行时所指向的对象。

SparkTransformer 的 internalTransformer 方法给出指针分析的实现，主要步骤有：

(1)创建 ContextInsensitiveBuilder 对象 b。

(2)pag=(PAG) b.setup(opts)，其步骤为：

1)创建 PAG 对象 pag，初始化 pag.nodeToTag 和 pag.typeManager，调用 typeManager 的方法 setFastHierarchy 做快速类层次分析建立类层次图，再将结果保存在 Scene 对象的 activeFastHierarchy 域中。

2)由选项 opts 决定 P2Set 的表示方式：Hash 集、有序数组集、位集或 Hybrid 集等，并依此实例化 setFactory 域。

3)由选项创建 b 的方法调用图域 cgb(CallGraph 对象)或 ofcg(OnFlyCallGraph 对象)。

(3)b.build()，其步骤为：

1)生成调用图，填充 b.ofcg 或 b.cfg，并设置到 Scene 对象的 activeCallGraph 域中；

2)填充 pag 中的结点与边，但结点的 P2Set 域仍为空。

(4)对 pag 进行类型屏蔽处理。

(5)由选项决定是否输出 pag 到文件。

(6)由选项选择传播器并执行传播。这一步计算出 pag 中各结点的 P2Set。Soot 提供多个传播器类(基类为 Propagator)实现不同的传播算法。

(7)将 pag 存到 Scene 对象的 activePointsToAnalysis 域中。

由上述分析可知，Spark 分析的结果包括类层次图、调用图以及 PAG 图，它们均保存在 Scene 对象中以便外界使用。

3.2 Side-effect 分析

Side-effect 通过分析程序中每一语句或方法的读写集进行如无用代码删除、循环代码外提等优化。为获得读写集，必须先掌握各变量指向哪些对象，即指针分析。

SideEffectTagger (继承自 BodyTransformer)为 Jimple IR 做 Side-effect 分析，它包装在名为 jap 的 Pack 中。其 internalTransformer 方法首先利用 Scene 对象中的调用图和指针分析信息创建 SideEffectAnalysis 对象 sea；再调用 sea.findNTRWSets 获得读写集；之后利用读写集建立依赖图；最后对分析结果编码和注解。

Soot 对 Side-effect 分析的默认实现是基于域的类层次分析的，这里则用 Spark 分析代替之。由第 2.3 节、第 3.1 节知，方法 runPacks 先执行全局变换，其中，cg.spark 阶段的变换结果保存在 Scene 对象中；之后进行局部变换，其中，side-effect 分析器将利用 Spark 分析的结果进行变换。

4 Soot 的扩展方法与应用

在 Soot 上可按如下方法扩展：

(1)分析要扩展的优化和变换，思考其工作流程和涉及的数据。

(2)确定变换的性质(全局还是局部?在哪种 IR 上操作?变换、优化还是注解等?)。

(3)根据步骤(1)、步骤(2)，定义由现有变换器类派生的新变换器，重写 internalTransformer 方法实现变换。

(4)将新变换器加入到现有或新建的 Pack 对象中，为之指定阶段名并决定它在 Pack 内各阶段中的位置。

(5)若新建一个 Pack 对象，则在 PackManager 类的 init 方法中添加相应的 addPack(…)语句；若新 Pack 对象所属的类不存在，则为之写一个由现有 Pack 类派生的类。

(6)若新 Pack 对象为全局变换，则在 soot.Main 中 runWholeProgramPacks 方法体的适当位置添加 getPack(<Pack 名>).apply()语句及相应的条件判断；若为局部变换，则在 soot.Main 中 runBodyPacks 方法体的适当位置添加 getPack(<Pack 名>).apply ()语句及相应的条件判断。

(7)对新变换器或 Pack 对象扩展命令行参数并在源码中进行相应的处理。

目前有许多基于 Soot 的项目，如加州伯克利分校的 Ptolemy、斯坦福大学的 chord、IBM 的 Canvas 等；还有一些基于 Soot 的大学研究生课程，如 McGill 大学的“优化编译器”和华盛顿大学的“程序设计语言的实现”等。

5 结束语

虽然 Soot 已实现许多分析、优化和注解技术，但是逃逸分析、Java 程序并发性分析等更多的功能有待实现。笔者期望通过本文的介绍能对从事相关工作的人员予以帮助，推动这方面研发工作的开展。

参考文献

[1] Vallee-Rai R, Co P, Gagnon E, et al. Soot: a Java Bytecode Optimization Framework[EB/OL]. (1999-06-24). http://www.sable.mcgill.

[2] Lhoták O. SPARK: A Flexible Points-to Analysis Framework for Java[D]. Montreal: McGill University, 2003.

[3] Lhoták O. Program Analysis Using Binary Decision Diagrams[D]. Montreal: McGill University, 2006.

(上接第 68 页)

[2] 王金全, 王 侃, 郑宇军. CBD 技术在装备保障信息管理系统中的应用[J]. 装甲兵工程学院学报, 2004, 18(4): 58-61, 70.

[3] Hudak P. Modular Domain Specific Languages and Tools[C]//Proc. of 5th Int'l Conf. on Software Reuse. Victoria, Canada: [s. n.], 1998: 134-142.

[4] Ledeczi A, Bakay A, Maroti M, et al. Composing Domain-specific Design Environments[J]. IEEE Computer, 2001, 34(11): 44-51.

[5] Weiss D M, Lai C T R. Software Product-line Engineering[M]. Massachusetts: Addison-Wesley, 1999.

[6] Batory D, Johnson C, MacDonald B, et al. Achieving Extensibility Through Product-lines and Domain-specific Languages: A Case Study[J]. ACM Trans. on Software Engineering and Methodology, 2002, 11(2): 191-214.

[7] Zheng Y J. Object-oriented Specification Composition and Refinement via Category Theoretic Computations[C]//Proc. of the 3rd Int'l Conf. on Theory and Applications of Models of Computation. Heidelberg: Springer, 2006: 601-610.

[8] Zheng Y J, Xue J Y. MISCE: A Semi-automatic Development Environment for Logistic Information Systems[C]//Proc. of the 1st IEEE Int'l Conf. on Service Operation, Logistics, and Informatics. Beijing: [s. n.], 2005: 1020-1025.

[9] 郑宇军, 石海鹤, 薛锦云. Spec#语言中的形式化特性[J]. 计算机科学, 2005, 32(增刊): 165-168.

[10] Xue J Y. A Unified Approach for Developing Efficient Algorithmic Programs[J]. Journal of Computer Sci. & Tech., 1997, 12(4): 103-118.