Slithice: 一个基于系统依赖图的 Java 程序切片工具

钱巨+，陶彬贤

(南京航空航天大学 计算机科学与技术学院，南京 中国 210016)

摘 要 当前程序切片的相关理论已经较为成熟，但针对 Java 程序的静态切片工具却非常少见。为便于展开切片应用研究，本文设计并实现了一个基于系统依赖图的 Eclipse 切片插件—Slithice。该插件支持不同粒度的底层分析和系统依赖图构建，从而可以使切片算法能够在精度和性能之间进行权衡，适应各种规模程序的分析需要。

关键词 Java，程序切片，静态

中图法分类号 TP311 DOI 号：

Slithice: A System Dependence Graph Based Program Slicing Tool for Java

QIAN Ju+, TAO Bin-Xian

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract Nowadays, the theory of program slicing is highly developed, but open tools for slicing Java programs are still rarely seen. To support the researches on the application of program slicing techniques, we design and implement an Eclipse slicing plugin named *Slithice*. The tool can perform system dependence graph based program slicing. It provides many options for the basis analyses and the system dependence graph construction. This can let the users make better trade-offs between efficiency and precision and hence make the tool more suitable for analyzing programs of various scales.

Key words Java, program slicing, static

1 引言

当前程序切片[1][2]的相关理论已经较为成熟，但在工具实现方面，主要的工作关注 C 程序[3][4]，以及 Java 程序的动态切片[5][6]，针对 Java 的静态切片工具依然非常少见。造成此种现象的主要原因包括多个方面：首先，程序切片依赖于许多底层的程序分析技术，因此实现起来较为困难；另外，代码规模的不断扩大使得许多切片算法必须经过一定的优化才能适应实际程序分析的需要。打造一款实际可用的切片工具存在许多挑战。目前，公开可获得的 Java 程序静态切片工具主要是 Indus[7][8]。Indus 具有强大的功能，但该工具不是基于依赖图实现的。对于经典的基于系统依赖图的程序切片[9]，尚未见到可公开获取的工具。

我们认为基于系统依赖图的程序切片应用广泛，其底层的依赖图可用来实现代码重构、克隆代码分析等多种不同的软件工程目标，在该依赖图上也可灵活配置不同的切片算法，因此此类切片方法值得进一步研究。为便于展开程序切片应用研究，我们设计并实现了一个 Eclipse 下的程序切片插件—Slithice[10]。Slithice 可对 Eclipse 中的项目作基于系统依赖图的静态切片。为保证伸缩性，该工具支持不同精度和性能的指针分析，支持采用不同的粒度来对动态堆空间进行抽象，可允许创建不同粒度的系统依赖图参数节点，允许进行限定深度的库模块分析，从而可以使算法能够在精度和性能之间进行更好的权衡，适应不同规模程序的分析需要。

在获得切片数据的基础上，我们设计并实现了了一个切片结果的呈现系统。它可用柱状图展现切片结果在不同文件中的分布。对于每个文件，可用缩略图形式，显示切片覆盖的代码在文件中的分

本课题得到国家自然科学基金(No.60903026, 61100034)资助. 钱巨，男，1981年生，博士，E-mail: jqian@nuaa.edu.cn，副教授，主要研究领域为软件分析与测试. 陶彬贤，男，1988年生，硕士研究生，E-mail: taobinxian@nuaa.edu.cn，主要研究领域为软件分析与测试.

布，也可在编辑器中以高亮方式展现切片结果。对于每个语句行，Slithice 可显示该行依赖的语句，并支持在互相依赖的语句之间进行跳转。在后续章节中，本文将详细介绍 Slithice 切片工具的设计思想以及使用方法。

2 系统总体结构

Slithice 切片系统的总体结构如图 1 所示。该系统的分析过程分为多个层次。底层是待分析程序的源代码以及字节码。在底层程序的基础上，分析分为两路。一路基于 Soot 字节码分析平台[11]，构建与字节码相对应的 Jimple 程序中间表示，并在该表示上构建依赖图，进行中间代码层的切片；另一路在源代码层基于 Eclipse JDT 工具[1]构建程序语法树。最后，通过行号信息和程序实体名称信息，可将 Jimple 中间表示层的切片结果反映到 JDT 语法树中，进而借助 JDT 在 Eclipse 视图中进行显示，并在源代码编辑器中高亮标注。

在 Jimple 中间表示层的分析方面，Slithice 首先进行指针分析，获得每个指针变量的指向信息。在指针分析的基础上，我们将结合类型继承信息等，构建程序调用图。以指向信息、调用图等为基础，可进行控制流分析和数据流分析。通过控制流分析，可采用经典的必经节点分析法，获得语句间的控制依赖[12]。在数据流分析方面，Slithice 第一步先通过副作用分析获得每个语句或方法可能读写的外部可见存储空间集，然后通过可达定义迭代获得各个方法中的定义—使用对，通过定义—使用对，可得到语句之间的数据依赖。结合控制依赖和数据依赖即可构建 Jimple 中间表示层的系统依赖图，在该依赖图上可进行过程间程序切片。为进行中间表示层的切片，我们先将源代码层设定的切片准则翻译为中间代码层的切片准则，如此即可实现从源代码层切片到字节码层切片的转换，最终的切片结果可通过行号和名字映射再次返回到源代码层。

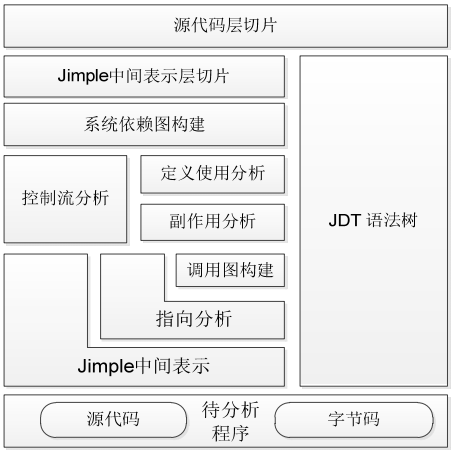

图 1 Slithice 总体结构

[1] Eclipse Java development tools (JDT). http://www.eclipse.org/jdt/

3 指针和副作用分析

指针和副作用分析是程序切片的重要基础，Slithice 支持不同粒度的指针和副作用分析（如图 2 所示）。在指针分析方面，该工具首先支持 Soot 中提供的 Spark 流和上下文不明感的指针分析方法。Spark 方法提供了较好的精度，但对大型程序仍然分析较慢。为提高效率，Slithice 还支持一种基于类型的指针分析方法（Type-Based），和一种素朴的指针分析方法（Naïve）。基于类型的指针分析以类型作为运行时对象的静态抽象，用一个指针变量运行时允许绑定的动态类型的集合作为该指针变量的指向集描述。研究表明，对一些应用，基于类型的指针分析也可以取得不错的分析效果[13]。Naïve 分析方法不对堆空间进行详细区分，它以一个普通对象和一个数组来抽象所有运行时对象，根据指针变量的类型判定其指向集是包含所有这两个抽象对象或是其中某一个。该方法具有最佳的性能，但精度稍差，适合用来分析规模非常大的程序。

在指针分析的基础上，Slithice 支持四种不同粒度的副作用分析。Field-Sensitive 分析能够在一定程度上区分针对不同实例对象属性域的读和写；Field-Based 分析不区分对象，但能够区分对不同属性域的读写；Type-Based 分析可区分对不同类型对象的读写；Naïve 分析可识别对数组和一般对象的读写，但不对具体对象作详细区分。在 Spark 指针分析下，采用 Field-Sensitive 的副作用分析对于许多语句

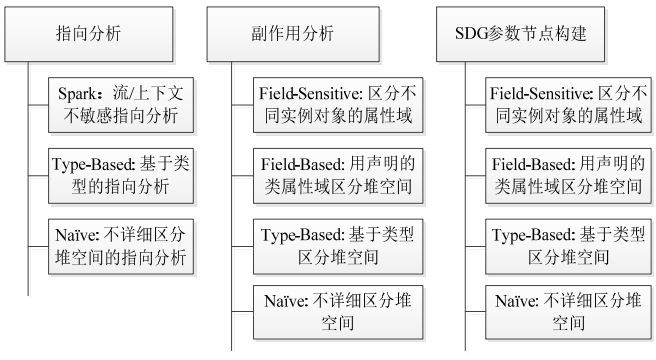

图 2 不同粒度的分析算法

会得到数以万计的可能读写空间，这将对后继分析的性能产生较大负面影响。支持不同粒度的副作用分析可允许用户在精度和性能之间进行更多选择。

4 系统依赖图构建

与已有的许多面向对象程序切片方法[1]类似，Slithice 将方法的接收对象当作一个隐藏的参数，将多态调用解析为到最终目标方法的调用，来将 Java 程序的切片问题转换为一般的过程间切片问题，构建系统依赖图完成程序切片。

在构造系统依赖图的过程中，一个关键的问题是对各个方法创建对应外部可见存储空间读写的形参节点和实参节点。被读写的外部可见空间可通过副作用分析获得。由于静态分析的保守性，一个方法的副作用集常常非常巨大。这使得据此而构建出来的参数节点有时数量过于庞大，影响了切片分析的效率。为减少系统依赖图中形参和实参节点的个数，Slithice 提供了 Field-Sensitive、Field-Based、Type-Based 和 Naïve 多种不同的粒度来构建参数节点。这些粒度与副作用分析中的堆空间抽象粒度含义类似，通过将不同的参数节点抽象归结为同一个节点，可降低系统依赖图的规模，提高分析性能。在上述各个粒度中，Field-Based 粒度上构建的系统依赖图与 Larsen 等[14]的依赖图接近，而 Field-Sensitive 粒度上构建的依赖图与 Liang 等[15]的依赖图接近。通过控制参数节点的抽象粒度，可支持现有的多种面向对象程序切片算法。

在 Java 程序切片中，庞大的库程序是造成切片效率低的一个重要原因。我们的实验表明，即使分析一个简单的输出语句，也会涉及到 5000 个以上的库方法。为避免在库方法的分析上投入过多的时间，Slithice 支持限定深度的库程序分析。比如，若设定库程序的分析深度为 2，则两层调用深度以内的库方法，将被当作一般方法来分析，而对于调用深度多于两层的库方法，我们对其进行一般的指针分析和副作用分析，但不计算方法体内的定义—使用关系，不分析从输入参数到输出参数之间的数据传递，而是假定每个输出参数依赖于所有输入参数。如此虽然降低了分析精度，但精度损失主要在库程序方面，对用户代码上分析的影响相对较小。

构建完系统依赖图后，通过经典的依赖图上的两遍遍历算法[9]就能获得一个切片准则所对应的过程间程序切片。

5 程序切片的基本步骤

使用 Slithice 工具进行切片包括以下三个基本步骤。首先，设置依赖图和切片的相关配置，如图 3 所示。这一步将设置底层采用的指针分析、副作用分析方法，以及依赖图中参数节点的构建粒度等。

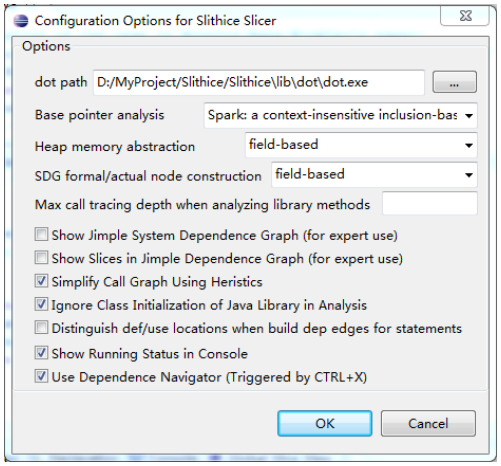

图 3 系统依赖图和切片配置

在此基础上，下一步需要设置程序的入口，也即指出程序 main 入口方法所在的类。图 4 给出了具体的设置界面，在该图中，点击“…”按钮，工具将列出所有带 main 方法的类，用户只需从中选择一个即可。

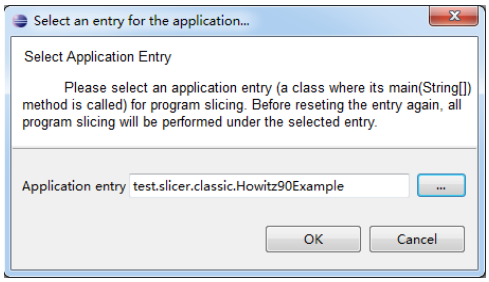

图 4 设置程序入口

选定程序入口后，Slithice 将自动在后台构建完整的系统依赖图，完成后将提示用户可以开始进行程序切片。用户只需在 Java 编辑器中选中一行，点击右键菜单中的“New Slicing Criterion”项即可开始配置切片准则。配置界面如图 5 所示。

配置好切片准则后，系统将在后台获得其对应的切片。切片将通过全局分布视图和单个文件中的代码高亮呈现给用户。图 6 给出了呈现界面，下方的树表是全局分布视图，柱状条表明了切片在不同文件中的分布情况。柱状条中的竖线是相关文件内切片覆盖语句分布的缩略图展示。上方的编辑器中

除了代码高亮，还支持依赖向导。用户在选择某个语句后，可通过快捷键调出依赖向导窗口，该窗口列出了当前语句所依赖的其它语句，点击窗口中的语句，可直接跳转到所依赖的语句上。

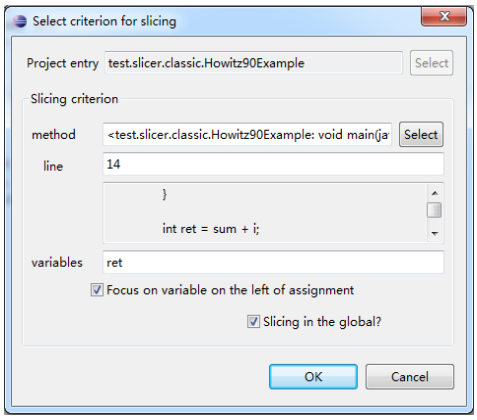

图 5 配置切片准则

除了显示最终的切片外，系统还支持显示选定方法所对应的过程内依赖图、控制流图、调用图等。以便于对程序的依赖结构做更深层次的分析。图 7 显示了调用相关功能的编辑器右键菜单。

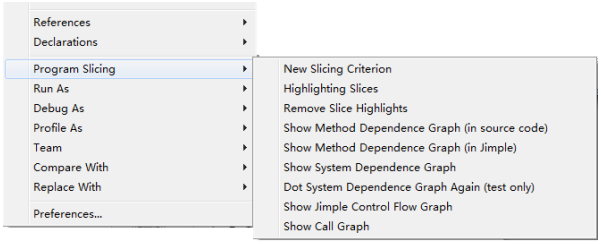

图 7 右键菜单以获得当前位置相关的依赖图、调用图等

6 小结

本文介绍了一个 Eclipse 下的 Java 程序静态切片插件的设计和实现。该切片插件支持多种不同粒度的底层分析，支持不同粒度的依赖图构建，从而可以使它能够适应不同规模程序分析的需要。

参 考 文 献

[1] Xu Baowen, Qian Ju, Zhang Xiaofang, Wu Zhongqiang, Chen Lin. A brief survey of program slicing, ACM SIGSOFT Software Engineering Notes, 2005, 30(2):1-36

[2] Abadi A., Ettinger R., Feldman Y. A. Fine Slicing: Theory and Applications for Computation Extraction // Proceedings of the International Conference on Fundamental Approaches to Software Engineering, 2012, pp.471-485.

[3] CodeSurfer. http://www.grammatech.com/products/codesurfer

[4] 徐晓晶，戚晓芳．并发程序切片原型系统的设计与实现．计算机科学与探索, 2012, 6(3): 257-266.

[5] JSlice. http://jslice.sourceforge.net/

[6] JavaSlicer. http://www.st.cs.uni-saarland.de/javaslicer/

[7] Jayaraman G., Ranganath V. P., Hatcliff J. Kaveri: Delivering the Indus Java Program Slicer to Eclipse // Proceedings of the International Conference on Fundamental Approaches to Software Engineering, 2005, pp. 269-272.

[8] Ranganath V. P. Scalable and accurate approaches for program dependence analysis, slicing, and verification of concurrent object oriented programs. PhD Thesis, Kansas State University, 2006.

[9] Horwitz S., Reps T., Binkley D. Interprocedural slicing using dependency graphs. ACM Transaction on Programming Languages and Systems, 1990, 22(1): 26-60

[10] Slithice. https://github.com/juqian/Slithice

[11] Vallée-Rai R., Hendren L., Sundaresan V., et al. Soot – a java optimization framework // Proceedings of the IBM Centre for Advanced Studies Conference (CASCON), 1999

[12] Ferrante J., Ottenstein K. J., Warren J. D. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319-349

[13] Diwan A., McKinley K.S., Eliot J., Moss B. Type-based alias analysis //Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 1998, pp. 106–117

[14] Larsen L., Harrold M. J. Slicing object-oriented software //Proceedings of the 18th International Conference on Software Engineering (ICSE), Berlin, Germany, 1996, pp. 495-505

[15] Liang D., Harrold M. J. Slicing objects using system dependence graphs //Proceedings of the International Conference on Software Maintenance (ICSM), Bethesda, MD, 1998, pp. 358-367

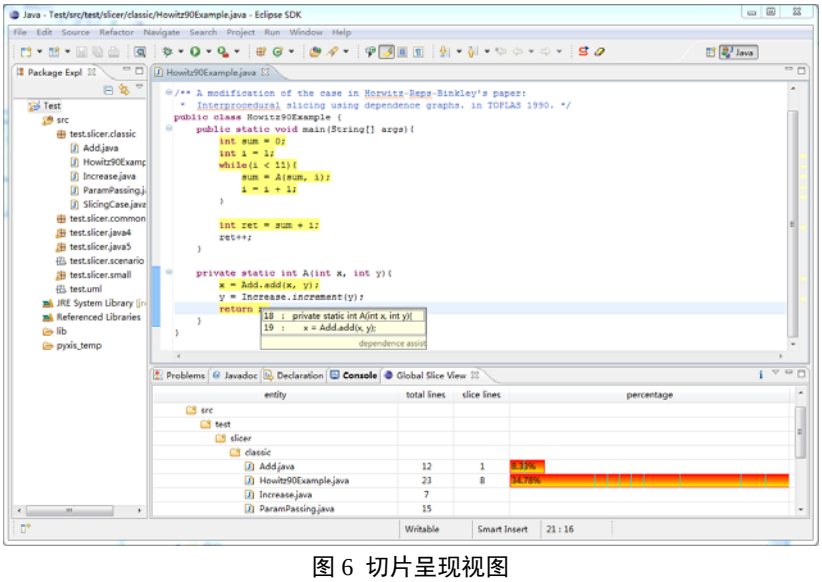

图 6 切片呈现视图