行政院國家科學委員會專題研究計畫成果報告

Java 反編譯器

Java Decompiler

計畫編號：NSC 90-2213-E-009-142
執行期限：90 年 8 月 1 日至 91 年 7 月 31 日

主持人：楊武　交通大學資訊科學學系
計畫參與人員：陳建財 黃經緯 王正峰 文忠民　交通大學資訊科學學系

一、中文摘要

早自 1960 年代開始，反編譯器就被認為是一項很有用的軟體工具，可用來做程式維護和解決程式安全問題。編譯器的工作是把原始程式變成機器碼，反編譯器的工作則是恰恰相反，其目的是把機器碼還原成原始程式。

Java 語言自 1995 年問世以來，由於其純物件導向、跨平台、及符合網路使用的特性，深受各界歡迎，許多企業為了節省應用軟體的維護成本，紛紛改採用 Java 語言開發新的系統，因此 Java 語言益顯重要。

我們提出本項計畫的主要目的，是為了配合去年所提出的計畫「Java 程式原始碼最佳化工具」，因為由僅僅分析程式設計師所寫的原始程式無法得知會由 callback 方式呼叫的 methods，必須建立包含函式庫的完整程序呼叫圖才能正確分析。

我們希望能建立自己的 Java 反編譯器，除了支援去年的計畫，使其更完整之外，並希望能藉此計畫深入瞭解 Java Virtual Machine (JVM)及 Java class 檔案格式，尋找其它對 Java 程式最佳化的可能性。此外，在反向工程日益顯得重要的今日，此計畫亦能成為幫助我們進入反向工程的墊腳石。

關鍵詞：Java、反編譯器、JVM、虛擬機器

Abstract

Decompilers have been considered useful software tool since 1960. The applications of decompilers include program maintainence and security. Compilers translate source code to binaries while decompilers do the things reversely.

Because Java has the features of pure object-oriented, cross-platform, and internet-featured, it has become more and more popular since 1995. Now, many enterprises use Java to develop new application for reducing the cost of software maintianence. Therefore, Java will become more important in the future.

The main purpose of this project is to support the project “Java program source code optimizer” that we proposed last year. We cannot know which methods will be called with the callback mechanism by analyzing the source written by the programmer. For getting the correct analysis result, a complete call graph that includes the methods in the libraries must be constructed.

Building a Java decompiler has many benefits for us. First, it can support and complete our previous project. Second, We will have the experience and knowledge of constructing a decompiler and study the JVM and class file structure thoroughly. It may help us to find other optimization methods for Java. As reverse engineering becoming more and more important today, this project will help us to get into this research area.

Keywords: Java, decompilation, decompiler, JVM, virtual machine

二、緣由與目的

編譯器的工作是把原始程式變成機器碼，反編譯器的工作則是恰恰相反，其目的是把機器碼還原成原始程式。雖然使用反編譯器會有侵犯版權的問題，但還是有

其正面的用途，其用途主要分為二類：程式維護和安全問題。前者包括取得已遺失的程式碼、維護舊的應用程式、把舊的未結構化程式改成結構式程式、移植應用程式到新的硬體平台、及針對有錯卻無法取得原始程式碼的程式做除錯工作。而在安全應用方面，在一些極機密重要的系統裡，因為也無法絕對信任編譯器，反編譯器可用來驗證編譯器所產生的程式碼是否安全，以及查驗是否存在有病毒之類懷有惡意的程式碼。

早自 1960 年代開始，反編譯器就被認為是一項很有用的軟體工具。在當時，反編譯器是用來幫助把程式從二代電腦移植到三代電腦上，如此一來，就不必花費太多的人力來改寫所有的程式。到了 1970 和 1980 年代，反編譯器是用來增加程式的移植性、除錯、重建遺失的原始碼、及修改現有的二進位碼。到了 1990 年代，反編譯器已經成為一項反向工程(reverse engineering)的工具，可用來檢查軟體中是否隱藏著不合法的程式碼、檢驗編譯器是否產生正確的碼、以及把二進位程式移植到不同的硬體平台上。

Java 語言自 1995 年問世以來，由於其純物件導向、跨平台、及符合網路使用的特性，深受各界歡迎，許多企業為了節省應用軟體的維護成本，紛紛改採用 Java 語言開發新的系統，因此 Java 語言益顯重要。

Java 反編譯器以 Java class 檔案為輸入資料，其輸出結果為 Java 原始程式碼。實際上，要從 bytecode 重建原始程式並不是一件容易的事，因此，Java 反編譯器並不一定可以產生原來的程式，或者會產生不容易看懂的程式，甚至可能會產生錯誤的程式。

目前已經有許多 Java 反編譯器問世，例如：

- Mocha, the original Java decompiler by **Hanpeter van Vliet**, beta (summer 1996). Hanpeter deceased on Dec. 31, 1996;
- Jasmine a patch to Mocha by **Tang Hongbo**, v. 1.10 (Jan. 25, 1998), *SourceTec*;
- WingDis by **Kathy Ho**, v. 2.12 (Dec. 13, 1997), *Wingsoft*;
- DeJaVu the decompiler of OEW 2.0.3 (Oct. 2, 1997), v. 1.0 (1996), *Innovative Software*;
- SourceAgain by **Paul J. Martino** and **Grigori Humphreys**, v. 1.10 beta 3.a (Feb. 16, 1998), *Ahpah*;
- Jad by **Pavel Kouznetsov**, v. 1.5.3.2 (Feb. 28, 1998).

由於 Java class 保存有豐富的符號(symbolic)資訊，反編譯器可以利用這些符號資訊產生近似原來程式的結果，讓人更容易瞭解程式原來的意義。有許多 Java 程式開發者在開發完成後，為了保護其智慧財產，會利用 Obfuscator 破壞這些符號資訊，讓人無法輕易瞭解程式原來的意義。尤有甚者，有些 Obfuscators 更在 bytecode 加入一些無害的、不會影響程式執行的碼，擾亂反編譯器的分析，使其無法產生原始程式。例如 HoseMocha 這套 Obfuscator 就會在每個 method 的 return 指令之後加上 pop 指令，這並不會破壞程式的語意(semantics)，但已經可以讓許多反編譯器執行失敗。

我們提出本項計畫的主要目的，是為了配合去年所提出的計畫「Java 程式原始碼最佳化工具」。該計畫的主要構想是把整個程式裡未用到的 classes、fields、methods 捨去，縮減程式的大小，達到最佳化的目的。但在執行該計畫的過程中，發現有一問題必須借助反編譯器才能達成。此問題源自於 Java 1.1 版之後所採用的 event model — delegation model，delegation model 在 event 發生時，會 notify 已註冊的處理物件，並呼叫處理物件的 methods，這類會由 callback 方式呼叫的 methods，無法單純由僅僅分析程式設計師所寫的原始程式得知，必須分析所有呼叫到的函式庫，建立完整的呼叫圖(call graph)才能確切得知那些 methods 會因為 callback 而使用到，而不會錯誤地省略了會使用到的 methods。

原本我們可以利用現有的這些 Java 反編譯器來支援前項計畫，但我們發現這些

現有的 Java 反編譯器各有其不足之處，有些無法處理 inner classes，有些無法處理複雜的控制結構。因此，我們無法信任前面列出的這些 Java 反編譯器，必須自行解決相關的問題。

我們希望能建立自己的 Java 反編譯器，除了支援去年的計畫，使其更完整之外，並希望能藉此計畫深入瞭解 Java Virtual Machine (JVM)及 Java class 檔案格式，尋找其它對 Java 程式最佳化的可能性。此外，在反向工程日益顯得重要的今日，執行此計畫的寶貴經驗，亦能成為幫助我們進入反向工程的墊腳石。

三、結果與討論

我們利用反編譯的技術將 Java class 檔案轉成抽象剖析樹(abstract parsing tree, AST)，然後將此剖析樹連接到「Java 程式原始碼最佳化工具」所建立的剖析樹，該工具就能取得所須的全部資訊，做完整的分析。若需輸出原始程式，只要在剖析樹上做一次前序(preorder)追蹤，依序輸出每個節點的內容，就可以得到原始程式。

其詳細步驟如下：

1. 讀入 class 檔案，分析 class 檔案結構，重建其 constant pool，做為查詢各種屬性名稱的資料庫；準備好各種屬性資料，供作查詢。
2. 還原 class 的結構和屬性，包括 base class、實作的 interfaces、fileds、methods，建立此 class 的符號表(symbol table)，此符號表主要是提供給「Java 程式原始碼最佳化工具」使用。
3. 針對各個 method 的程式碼劃分 basic blocks，建立流程圖，重建其控制結構，成為較高階的控制結構，讓使用者較容易看懂。
4. 將低階的 bytecode 指令，先做一次 disassembler，成為較高階的 JVM assembler 指令，再依其 pattern 還原成更高階的 Java 指令和運算式，建立剖析樹。
5. 輸出原始程式。

第１、２項工作是根據 JVM 規格書，分析 class 檔案結構以取得所需的資訊。基本上，constant pool 就像一本字典，記錄著有關 class、fields、methods 及各種屬性的符號資訊，如 class 名稱、fields 的型別和存取旗標、methods 的參數型別和回傳型別等，好好利用這些資訊，就可以產生與原始程式非常相近的程式。constant pool 的架構像是多層連結的序列，有些資訊必須依其連結輾轉探訪至最後一層才能得到符號資訊，中間有許多節點只是記錄另一個節點的位置，因此，必須依照 class 的檔案規格，讀入 constant pool 資料之後，重建其結構，讓後面的工作能夠很輕易地查詢到所需要的符號資訊。

第３項工作則相當具有挑戰性，主要的工作是將那些條件式分支(conditional branch)令和無條件跳躍(jump)指令消除，變成較高階的控制結構，如 if、for、while 等結構。Java 語言雖然沒有 goto 指令，但它的 break 和 continue 指令後面可指定標籤(label)，跳到或跳過指定的位址。雖然看起來很像 goto，但此標籤有限制範圍(scope)，僅限於緊接其後的敘述式(statement)之內。

Ramshaw 在 1988 年就已經針對 Pascal 語言的各種 goto 情形提出消除 goto 的方法，由於 Pascal 的 goto 指令幾乎沒有任何限制，它可直接跳進迴圈或其它程序裡，因此，其轉換相當複雜。而 Java 語言的 break 和 continue 指令雖然有變化，但所產生的控制流程圖都還是 reducible，根據 Ramshaw 的証明，利用 multiexit 的指令就可以找出結構上相等(structure equivalent)的結構化程式，而 Java 的 break label 和 continue label 指令正是支援 multiexit 的指令。因此，我們可以確定 Java 的 bytecode 一定可以轉換成結構化的程式。

雖然按照直接 Ramshaw 的方法去做可以得到結構化的程式，但我們可以發現，即使是簡單的 if 敘述式也都會變成迴圈，而且程式變得很難看得懂。反編譯器的主要目的就是要讓人看得懂程式的意義，如

果產生的結果不具可讀性，那這個反編譯器就沒有什麼價值了。

我們在 Java 語言的獨特限制下，找出較簡單的還原高階控制結構的方法，包括迴圈、if 判別式等，並針對 if 判別式做到 short circuit evaluation，合併多個 if 判別式的條件，讓程式更容易讓人看得懂。

第 4 項工作必須詳細瞭解 JVM 指令的格式，JVM 本身是個堆疊機器(stack machine)，因此反編譯器也必須模擬指令所執行的堆疊動作，以 bottom-up 的方式建立剖析樹。比較麻煩的是那些控制堆疊的指令，如 dup, dup_x1, dup_x2, dup2, dup2_x1, dup2_x2, pop, pop2 等，這些指令主要是為了配合 integer, float, long, double 等型別的資料長度不同。而我們在模擬 JVM 時，進出堆疊的並不是原來的資料，而是抽象剖析樹的節點。因此，在操作堆疊時必須特別注意這些特定的指令。

此外，還必須觀察某些組成運算式的特定指令 pattern，遇到這些 pattern 時，就可以還原成較高階的運算式。由於我們無法得知 Java 編譯器產生 bytecode 的規則，因此必須觀察歸納各種基本運算式所產生的碼，建立重組的規則，以啟發式(heuristic)的方式反編譯。

第 5 項工作必須在剖析樹上做一次前序(preorder)追蹤，依序輸出每個節點的內容，就可以得到原始程式。在這裡還可以加上一些美化程式碼的工作，例如做縮排處理，或是將 full qualified class name 改成 simple class name，並加入所需的 import 敘述句，以符合一般人寫 Java 程式的習慣。

四、成果自評

在本計畫中，我們的研究人員得到研究 Java JVM 的寶貴經驗，對 Java class 檔案之結構和 JVM 之 bytecode 指令有了深入的瞭解。此外，亦獲得了實作反編譯器的方法與實務經驗。對於 Java 的推廣與開發，有許多的幫助。此外，對於將來要進行 reverse engineering 方面的研究，亦墊立了良好的基礎

五、參考文獻

[1] Cifuentes C. A Structuring Algorithm for Decompilation. Proceedings of the XIX Conferencia Latinoamericana de Informatica. Buenos Aires, Argentina, 1993; 267-276.

[2] Cifuentes C, Gough KJ. Decompilation of binary programs. Software - Practice and Experience 1995; 25(7):811-829.

[3] Cifuentes C. Structuring decompiled graphs. Proceedings of the International Conference on Compiler Construction (CC'96). Lecture Notes in Computer Science 1060. Linkoping, Sweden, 1996; 91-105.

[4] Ramshaw L. Eliminating go to's while preserving program structure. Journal of the Association for Computing Machinery 1988; 35(4):893-920.

[5] Lindholm T, Yellin F. The Java virtual machine specification (2ed). Addison-Wesley: Reading, MA, 1999.

[6] Gosling J. The Java language specification (2ed). Addison-Wesley: Boston ; London, 2000.

[7] Fischer CN, LeBlanc RJ. Crafting a compiler with C. Benjamin/Cummings: Redwood City, Calif., 1991.

[8] Muchnick SS, Jones ND. Program flow analysis : theory and applications. Prentice-Hall: Englewood Cliffs, N.J., 1981.

[9] Muchnick SS. Advanced Compiler Design and Implementation. Morgan Kaufmann: San Francisco, California, 1997.

附件：封面格式

行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※　　　　　　　　Java 反編譯器　　　　　　　　　　※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：個別型計畫
計畫編號：NSC90－2213－E－009－142－
執行期間： 90 年 8 月 1 日至 91 年 7 月 31 日

計畫主持人：楊武
共同主持人：
計畫參與人員：陳建財 黃經緯 王正峰 文忠民

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學資訊科學學系

中 華 民 國 91 年 10 月 1 日