Optimizarea fisierelor Java

Abstract

In acest paper voi descrie incercarea mea de a crea un optimizator de spatiu pentru fisierele .class ale limbajului Java.

Acest optimizator se bazeaza pe analiza statica a fisierelor pentru eliminarea metodelor nefolosite din cadrul fisierelor.

Voi analiza structura fisierelor, voi explica modul de analiza alor si voi expune modul de eliminare a metodelor.

Introducere

Java

Java este un limbaj de programare orientat pe obiecte. Acesta a fost dezvoltat de catre Sun Microsystems (acum Oracle), iar prima versiune a aparut in anul 1995.

Jaza s-a bazat pe sintaxa limbajului C, si a introdus notiunea de “scrie o data, ruleaza peste tot” (eng. “write once, run everywhere”). Spre deosebire de C si de C++, care trebuiesc compilate pentru fiecare platforma tinta, Java a avut avantajul ca trebuie compilat o singura data, si va merge garantat pe toate platformele suportate de limbaj.

Java Bytecode

Solutia limbajului Java pentru a fi independent de platforma este de transforma codul intr-o reprezentare intermediara, in loc de direct in cod binary pentru o anumita arhitectura .

Compilatorul Java (`javac`), transforma codul Java intr-un limbaj intermediar, numit Java Bytecode.

Acest limbaj este un limbaj low-level, destinat in mod exclusiv procesarii de catre masini, spre deosebire de codul Java, care este destinat oamenilor.

Dupa ce compilatorul a procesat codul Java, provenit din fisere .java in format text, acesta salveaza rezultatul in fisiere de tip clasa (.class) in format binar.

Masina Virtuala Java (JVM)

Odata generate fisierele binare, acestea sunt executate pe o masina virtuala specifica limbajului Java - numita JVM sau `The JVM` (eng. Java Virtual Machine).

Aceasta masina virtuala are rolul de a citi fisierele de clasa binare si de a le interpreta.

Masina virtuala este implementata ca o "masina cu stiva" (eng. stack machine), unde toate instructiunile limbajului bytecode interactioneaza cu datele de pe o stiva controlata de aplicatie.

Masina virtuala insusi este implementata in C/C++, si este compilata in cod binar direct, dependent de arhitectura. Dezvoltatorii limbajului Java sunt responsabili pentru corectitudinea si siguranta masinii virtuale, in timp ce dezvoltatorii de aplicatii Java au garantia ca daca codul lor Java este corect, atunci acesta va rula la fel, deterministic, pe orice platforma.

In acest regard, limbajul Java poate fi vazut ca un limbaj interpretat. Comparand cu alte limbaje populare interpretate, ca de exemplu Python, Ruby, sau Perl, ne-am astepta ca si Java sa fie la fel de incet ca acestea [1]. Totusi, Java obtine performante mult mai bune decat aceastea. Acest fapt se datoreaza compilarii tocmai-la-timp (eng. just-in-time), in care atunci cand interpretorul observa o secventa de cod care este interpretata repetitiv de foarte multe ori, va genera direct cod binary pentru aceasta.

[1] https://github.com/trizen/language-benchmarks