자바 클래스 영역 크기 예측을 위한 탐침 클래스의 사용†

양희재
경성대학교 컴퓨터공학과
Email : hjyang@star.ks.ac.kr

Use of Probe Class for Estimating Java Class Area Size

Heejae Yang
Department of Computer Engineering
Kyungsung University

Abstract

Class area is a portion of memory where the constants, fields, and codes of the classes loaded into the Java virtual machine are kept. Knowing the size of the class area is very important especially for embedded Java system with limited memory resources. This paper induces a formula which makes it possible estimate the size of the area. The formula needs some constant values specific to target JVM implementation. We also show that these values can be found using some simple probe classes. An experimental result is included in this paper to confirm the correctness of our approach.

I. 서론

자바 프로그램은 자바가상기계(JVM) 상에서 실행된다. JVM 명세에 따르면 JVM 에서 사용되는 메모리는 클래스 영역, 자바 스택 영역, 힙 영역, 그리고 네이티브 메소드 영역 등 크게 네 가지의 개념적 공간으로 나뉜다 [1].

이 공간들 중에서 클래스 영역이란 클래스에 대한 모든 정적인 정보들이 저장되는 곳이다. 클래스의 이름, 상속 관계, 접근제어값, 필드와 메소드 정보 등이 모두 이곳에 저장된다. 메소드를 이루는 바이트코드나 오퍼랜드로 사용되는 상수들도 역시 모두 이곳에 저장된다.

클래스 영역의 내용은 일단 클래스가 적재된 후에는 변함이 없다. 즉 자바 프로그램 실행 시 그 내용이나 크기가 결코 바뀌지 않는다.

반면 힙 영역은 실행시 계속해서 그 내용과 크기가 변한다. 새로운 인스턴스가 생성될 때마다 힙 메모리가 할당되며, 그 인스턴스가 더 이상 사용되지 않으면 쓰레기 수집기에 의해 기 할당된 힙 메모리는 회수되어진다.

본 논문의 관심은 JVM 의 메모리 중에서 클래스 영역의 크기를 예측하는 방법을 발견하는 것에 있다.

클래스 영역의 주요 정보는 클래스 파일에 있는 내용들이므로 클래스 파일을 분석함으로서 이 영역의 크기를 대략 예측할 수도 있다. 그러나 클래스 영역 메모리에 놓이는 정보들은 클래스 파일 자체가 아니라 변형된 반사 이미지이므로 [2] 클래스 파일의 분석만으로 이 크기를 예측하는 것은 매우 어려우며, 또한 구현된 JVM 플랫폼마다 분석이 달라질 수 있다.

본 논문에서는 클래스 영역의 크기를 예측하기 위한 새로운 방법을 제안하였다. 클래스 영역의 주요 정보는 클래스 정보(*class*), 상수 정보 (*const*), 필드 정보 (*field*), 메소드 정보 (*method*) 등 네 가지이다. 따라서 클래스 영역이 사용하는 메모리의 양 M 은 이 네 가지 정보들을 변수로 하는 함수 $M = f(class, const, field, method)$ 로 나타낼 수 있다.

본 연구에서는 함수 f 가 어떤 값을 가질지 유도하고, 각 변수들이 갖는 계수(coefficient)를 발견하기 위해 탐침 클래스를 사용하는 것에 대하여 밝히고자 한다. 이 방식을 통해 클래스 영역의 크기를 분석해 본 결과 대부분의 경우에서 메모리 사용량의 예측값과 실

† 이 논문은 2003년도 정보통신부 지원 정보통신기초기술연구지원사업에 의해 연구되었음.

제값이 3% 이내의 오차범위에서 일치함을 발견할 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 자바가상기계 명세에서 정의하는 클래스 영역 메모리에 대해 설명하며, 3장에서 클래스 영역의 크기를 나타내는 일반적 수식을 유도한다. 이 수식은 JVM 플랫폼 구현에 따라 각기 다른 값의 계수들을 가지며, 탐침 클래스를 사용하여 이 값을 밝히는 방법에 대해서도 설명하였다. 4장에서는 이 수식 및 탐침 클래스를 simpleRTJ 라고 하는 특정 JVM 상에서 적용하여 본 연구의 유효성을 증명하였으며, 5장에서 결론을 맺는다.

II. 클래스 영역

모든 클래스는 클래스 영역 메모리에 저장된다. 클래스는 원래 클래스 파일에서 비롯된 것이므로 그 구조는 클래스 파일의 그것과 매우 흡사하다. 클래스 파일 구조는 자바가상기계 명세에서 정한 바에 따라 헤더, 상수 풀 (constant pool), 클래스 정보, 필드 정보, 메소드 정보 등 모두 다섯 가지 부분으로 구성된다 [3].

헤더는 매직 번호와 버전 번호로 이루어지는 8 바이트의 고정된 영역이며, 상수 풀은 연산의 오퍼랜드로 사용되는 일반 상수 외에 클래스나 필드, 메소드 등의 이름과 형식 등을 나타내는 각종 상수들로 이루어진다. 이들 상수들은 클래스 적재 시 다른 클래스들과의 링크 목적으로 사용되며, 실행 시 형식 확인 등의 목적으로도 일부 사용된다. 클래스 정보는 이 클래스의 접근제어값, 자신 및 상위 클래스의 이름, 구현하고 있는 인터페이스의 이름 등을 나타내며, 필드 정보와 메소드 정보는 각각 이 클래스가 갖는 필드와 메소드에 대한 정보를 가지고 있다.

클래스 파일이 메모리의 클래스 영역에 놓일 때는 다른 클래스와의 연결, 즉 레졸루션(resolution)이 완료된 상태이므로 상수 풀의 내용 중 많은 부분이 생략되어진다. 헤더 부분도 확인이 끝난 후이므로 역시 생략된다. 나머지 부분은 보다 접근하기 쉬운 형태로 변형되어 메모리에 적재된다.

III. 클래스 영역 메모리 사용 분석

3.1 클래스 당 메모리 사용량

클래스 파일이 다섯 가지의 부분으로 구성되는데 비해 클래스 영역에서 실제 클래스가 차지하는 메모리는 다음과 같은 네 가지 부분으로 구성될 것임을 가정할 수 있다.

첫째는 클래스 파일 내 클래스 정보 해당 부분을 저장하는 메모리다. 이 부분은 접근제어값, 자신 및 상위 클래스의 연결 포인터 또는 인덱스 값, 구현하는 인터페이스들에 대한 포인터, 필드 테이블과 메소드 테이블 등에 대한 포인터 등으로 이루어진다. 이 값들의 크기를 각각 산출하기는 어렵다. 예를 들어 접근제어값의 크기는 플랫폼에 따라 16비트일 수도 있으며 32비트일 수도 있다. 각종 포인터나 인덱스의 크기, 그리고 테이블의 구성 등도 구현된 플랫폼마다 각기 다른 값과 형식을 가질 것이며 따라서 개별 크기를 알기 어렵다. 그래서 우리는 클래스 정보를 저장하는 메모리의 크기 M_s 를 그저 K_s 라는 값이라고 정하도록 하자. 즉 $M_s = K_s$ 이다.

둘째는 상수 정보를 저장하는 메모리다. 클래스 파일의 상수 풀 내용 중에서 레졸루션을 위한 정보들은 클래스 적재 시 이미 이용된 상태이므로 이들이 클래스 영역에 있을 필요는 없다. 즉 메모리에는 바이트코드 실행 시 실제로 오퍼랜드로 사용되는 상수들만 있는 것으로 가정할 수 있다. 이들 상수들은 값과 더불어 형식(int, char, String 등)을 나타내는 꼬리표와 길이 등이 포함되어있으므로 c 개의 상수들이 차지하는 메모리의 양 $M_c = \sum_{i=0}^{c-1}(K_c + cl_i)$ 로 나타낼 수 있다. 여기서 K_c 는 꼬리표 등의 크기이며, cl_i 는 i 번째 상수의 길이이다. 이 식을 풀면 다음과 같이 나타내어진다. $M_c = K_c\, c + \sum cl_i$..

셋째는 필드 정보를 저장하는 메모리다. 필드 정보에서 반드시 필요한 내용은 접근제어값과 필드의 형식, 각 필드의 인덱스 값 등을 들 수 있다. 필드마다 이런 내용들이 저장되어야 하므로 메모리 사용량 M_f 는 필드의 개수 f 에 정비례한다고 볼 수 있다. 즉 $M_f = K_f\, f$ 가 된다.

마지막 넷째는 메소드 정보를 저장하는 메모리이다. 메소드 정보에서 필수 내용은 접근제어값, 형식, 스택과 지역변수배열의 크기, 바이트코드의 길이 등이다. 이 전체 정보가 요구하는 메모리 양을 K_m 이라 하자. 이 값은 각 메소드의 종류에 관계없이 일정한 값을 갖는다. 다만 메소드를 이루는 바이트코드 자체의 길이는 메소드마다 각기 다르므로 i 번째 메소드의 바이트코드 길이를 ml_i 라고 하면 메소드 정보의 전체 메모리 사용량 M_m 은 $M_m = \sum_{i=0}^{m-1}(K_m + ml_i) =$

$K_m\ m + \sum ml_i$ 가 된다. 여기서 m 은 메소드의 개수를 의미한다.

따라서 한 클래스가 클래스 영역에서 사용하는 메모리의 사용량 M 은 다음 수식으로 주어진다.

$$M = M_s + M_c + M_f + M_m$$
$$= K_s + (K_c\ c + \sum cl_i) + K_f\ f + (K_m\ m + \sum ml_i) \quad \text{..... (1)}$$

3.2 탐침 클래스의 사용

JVM 내부에서 한 클래스가 사용하는 메모리의 양은 (1) 식과 같다. 이 식에서 상수, 필드, 메소드의 개수 c, f, m, 상수의 길이 cl_i , 메소드 바이트코드의 길이 ml_i 등은 클래스 파일을 분석함으로서 알 수 있다. 우리가 알 수 없는 것은 계수로 사용되는 각종 상수 K_s, K_c, K_f, K_m 이다. 이 값들은 JVM의 구현에 따라 다소 다른 값을 가지며, 구현된 JVM 의 원천 코드를 조사함으로서 알 수 있겠지만 코드를 모두 조사하는 것은 어려울 뿐 아니라 오류를 일으킬 확률도 크다.

본 논문에서는 이 값들을 알기 위해 탐침 클래스를 사용하는 방법을 제안했다. 즉 아주 단순한 4개의 클래스에 대해 메모리 사용량을 구한 후 (1) 식을 대입하면 상수를 알 수 있는 것이다. 모르는 상수가 4개이므로 4개의 탐침 클래스만 사용하면 된다.

본 논문에서 사용한 탐침 클래스는 그림 1과 같다. 클래스 C1 은 아무런 상수나 필드, 메소드를 갖지 않는 null 클래스이다. 그러나 사실은 이와 같은 null 클래스라 할지라도 생성자 메소드를 가지므로 m = 1, $\sum ml_i$ = 5 이며, $c = f = 0$ 이다.

탐침 클래스 C2 는 하나의 메소드를 갖는 클래스이지만, 실제로는 생성자 메소드를 포함하므로 m = 2, $\sum ml_i$ = 6 이다. 마찬가지로 $c = f = 0$ 이다.

탐침 클래스 C3 는 하나의 필드를 갖는 클래스이다. 즉 f = 1 이며, 생성자 메소드를 가지므로 C1 과 마찬가지로 m = 1, $\sum ml_i$ = 5 가 된다. c = 0 이다.

마지막으로 C4 는 하나의 상수를 갖는 클래스이다. 따라서 c = 1 이며, $\sum cl_i$ = 5 가 된다 ("Hello" 의 길이). 이 클래스는 한 개의 필드를 가지며 (f = 1), 생성자 메소드를 가지므로 m = 1, $\sum ml_i$ = 11 이다 (C1 과 달리 생성자 메소드의 바이트코드 길이가 길다).

각 탐침 클래스의 메모리 사용량을 각각 M_1, M_2, M_3, M_4 라고 하면

```
class C1 {   /* null class */
}
class C2 {   /* single method */
    void methd() {
    }
}
class C3 {   /* single field */
    C1 c;
}
class C4 {   /* single constant */
    String s = "Hello";
}
```

그림 1. 4가지의 탐침 클래스

$$M_1 = K_s + K_m + 5 \quad \text{..... (2)}$$
$$M_2 = K_s + 2K_m + 6 \quad \text{..... (3)}$$
$$M_3 = K_s + K_f + K_m + 5 \quad \text{..... (4)}$$
$$M_4 = K_s + (K_c + 5) + K_f + (K_m + 11) \quad \text{... (5)}$$

와 같다.

이제 실제 JVM 내에서 M_1, M_2, M_3, M_4 의 값을 측정하면 K_s, K_c, K_f, K_m 의 값을 알 수 있으며, 그 값을 (1)식에 대입하면 특정 JVM 내에서 일반 클래스가 사용하는 메모리의 양을 알 수 있게 된다.

VI. 실험 및 평가

이 장에서는 실제 실험을 통해 3장에서 설명한 탐침 클래스를 사용한 클래스 영역 메모리 사용량을 조사해 보았다. 실험에 사용한 JVM 은 RTJ computing 사의 simpleRTJ [4] 이다.

먼저 simpleRTJ 상에서 그림 1에서 보인 탐침 클래스를 적재해 본 결과 메모리 사용량 M_1, M_2, M_3, M_4 는 각각 68, 96, 72, 100 바이트로 조사되었다. 이 값을 (2)-(4) 식에 대입하면 상수 K_s, K_c, K_f, K_m 의 값은 각각 36, 17, 4, 27 이 됨을 알 수 있다. 이 값들을 (1)식에 대입하면 simpleRTJ JVM에서 임의의 클래스의 클래스 영역 메모리 사용량 M 은 다음과 같이 주어진다.

$$M = 36 + (17c + \sum cl_i) + 4f + (27m + \sum ml_i)$$

구한 식의 정확성을 알아보기 위해 simpleRTJ 의 핵심 API 클래스들에 대해 위 식을 대입한 결과값과 실제 측정값[5]을 비교해 보았다 (표 1).

이 표에서 알 수 있듯이 거의 대부분의 경우에서 오차는 2-3%에 지나지 않으며, 다만 Boolean (7%), Object (8%), Thread (11%) 에서는 다소 큰 오차를 보였다. 따라서 우리가 구한 식은 전반적인 API 클래스에 대해 매우 정확하게 클래스 영역의 메모리 사용량을 나타내는 것으로 볼 수 있었다.

V. 결론

하나의 자바 프로그램은 다수개의 클래스 파일로 이루어지며, 클래스 파일의 내용은 JVM 상의 클래스 영역에 저장되어진다. 클래스 파일의 내용만으로는 클래스 영역의 크기를 예측할 수가 없다. 본 논문에서는 클래스 영역의 크기를 나타내는 일반적 수식을 유도하였다. 이 수식은 구현 JVM 에 따라 각기 다른 값을 갖는 네 가지의 상수들을 갖는데, 이 상수값을 알기 위해서 아주 간단한 형태의 클래스, 즉 탐침 클래스를 사용하는 방법에 대해 설명하였다. 네 가지의 각기 다른 탐침 클래스를 사용함으로서 수식에 포함된 네 가지의 상수값을 알 수 있다. 실제 실험을 통해 본 방법의 유효성을 증명하였다.

클래스 영역의 크기를 예측하는 것은 내장형 자바 시스템과 같이 특히 메모리의 사용상 제한을 받는 시스템에서 더욱 중요하다. 본 논문에서 제안한 방법으로 어느 정도 정확한 수준의 메모리 사용량의 예측이 가능했다. 향후 클래스 영역의 크기 뿐 아니라 힙 메모리 등 동적 메모리의 사용량도 예측할 수 있는 연구도 추진할 예정이다.

표 1 simpleRTJ API 클래스들에 대한 실험

클래스	c	$\sum c$	f	m	$\sum m$	M	측정값
Boolean	2	9	3	8	155	462	496
Integer	6	13	3	18	490	1,139	1,160
Object	0	0	0	10	63	369	400
String	1	4	4	45	1,772	3,060	3,012
Thread	0	0	3	22	161	803	900
Throwable	1	0	1	7	41	287	292
Exception	0	0	0	2	11	101	104
Error	0	0	0	2	11	101	104
Arithmetic-Exception	0	0	0	2	11	101	104
Internal-Error	0	0	0	2	11	101	104

참고문헌

[1] 양회재, 자바가상기계, 한국학술정보, 2001년 3월, ISBN 89-5520-342-4

[2] 김성수, 김세영, 양회재, "내장형 자바 가상기계를 위한 클래스 이미지 파일의 분석과 비교," 한국정보과학회 학술발표회, 제30권 1(B)호, pp.28-30, 2003년 4월

[3] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley, 1997

[4] RTJ Computing, *simpleRTJ: A Small Footprint Java VM for Embedded and Consumer Devices*, http://www.rtjcom.com

[5] 양회재, "simpleRTJ 임베디드 자바가상기계의 ROMizer 분석 연구," 한국정보처리학회 논문지, 제10-A권 5호, 2003년 10월