전형적 자바 프로그램에서 필드의 사용 형태

양희재
경성대학교 컴퓨터공학과
hjyang@star.ks.ac.kr

Field Usage Pattern in Typical Java Programs

Heejae Yang
Department of Computer Engineering, Kyungsung University

요 약

자바 프로그램은 클래스의 집합으로 이루어지며 클래스는 필드와 메소드로 구성된다. 필드는 힙 메모리에 저장되므로 필드의 사용 형태는 힙 메모리의 사용 형태에 직접적인 영향을 미친다. 즉 자바 힙 메모리의 사용을 분석하기 위해서는 필드의 사용 형태를 분석해야 한다는 것이다. 본 논문에서는 전형적 자바 프로그램에서 필드가 어떻게 사용되어지는지를 정성적으로 분석하고 실험을 통해 확인하였다. 본 연구의 결과는 자바가상기계의 힙 메모리 구조 개발에 도움을 줄 것이며, 나아가 효율적인 자바가상기계의 개발을 가능하게 할 것으로 기대된다.

1 서 론

객체지향형 언어인 자바에서 프로그램은 클래스의 집합으로 구성된다. 잘 알려진 바와 같이 자바의 클래스는 객체의 속성을 나타내는 필드(field) 부분과 행동을 나타내는 메소드(method) 부분으로 이루어진다 [1][2].

필드는 자바 프로그램 실행에서 매우 중요한 역할을 담당한다. 일반적으로 하나의 클래스에 대해 여러 개의 객체, 즉 인스턴스(instance)들이 생성되는데, 이 인스턴스들은 필드들을 저장하기 위한 힙(heap) 메모리 공간을 각각 할당 받는다. 예를 들어 문자열 처리를 위한 `java.lang.String` 클래스는 단지 하나이지만 일반적 자바 프로그램에서 `String` 클래스의 인스턴스는 수없이 많이 생성되며 각 인스턴스들은 저마다 필드 저장을 위한 독립된 힙 메모리를 할당 받는다.

자바 프로그램 실행을 위해 힙 외에 다른 메모리도 사용되지만 그 중 힙 메모리가 가장 중요하다. 가장 많은 양의 물리적 메모리가 할당되는 곳이 힙 메모리이며, 프로그램 실행 도중 메모리 고갈로 인해 더 이상의 인스턴스 생성이 불가능해 질 경우 자바의 특징 중 하나인 자동적 쓰레기 수집(automatic garbage collection)이 이루어지는 곳도 힙 메모리다.

효율적인 자바 프로그램 실행을 위해서는 효율적 힙 메모리 관리가 무엇보다 필요하다. 힙 메모리는 앞에서 언급한 바와 같이 각 객체들의 필드 저장을 위한 목적으로 주로 사용되므로 힙 메모리 관리를 위해서는 자바 프로그램에서 필드들이 어떻게 사용되어지는지를 분석할 필요가 있다.

본 연구에서는 일반적 자바 프로그램에서 필드가 어떻게 사용되는지를 분석하였으며, 이 분석은 효율적인 힙 메모리의 설계에 도움을 줄 것이다. 본 연구에서는 필드의 읽기 및 쓰기 비율, 필드에 대한 접근 빈도, 그리고 필드 접근에 대한 시간적 분포 등에 대해 고찰해보았으며, 실제 실험을 통해 고찰 내용을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 필드가 놓이는 힙 메모리를 비롯한 자바가상기계의 메모리 구조를 설명하고, 3장에서는 전형적인 자바 프로그램에서 필드가 어떤 형태로 사용되는지에 대해 추론해본다. 4장에서는 3장에서 내린 가정을 실험을 통해 확인하며, 5장에서 결론을 맺는다.

2 자바 메모리

그림 1은 자바가상기계(JVM)의 메모리를 개념적으로 나타낸 것이다 [3]. 클래스 영역은 클래스 정보, 즉 메소드를 이루는 바이트코드 등이 포함되어있는 읽기 전용 영역이며, 오퍼랜드로 사용되는 각종 상수 값도 이곳에 위치한 상수 풀에 저장되어있다.

다음으로 힙(heap) 영역이 있는데, 이곳은 클래스들의 인스턴스들이 저장된다. 즉 각 인스턴스들이 가지고 있는 필드 값이 저장되며, 아울러 배열들도 이곳에 저장된다. 힙 영역은 새로운 인스턴스의 생성에 따라 할당되며, 그 인스턴스가 더 이상 사용되지 않게 되면 쓰레기 수집기에 의해 메모리가 회수되게 된다.

세 번째는 자바 스택 영역이다. 하나의 메소드가 호출될 때마다 자바 스택 영역에 스택 프레임이 새로 생성되며, 스택 프레임은 다시 오퍼랜드 스택과 지역변수배열로 나뉜다. 스택 프레임은 호출된 메소드가 종료되면 메모리에서 사라지게 된다.

마지막 네 번째는 네이티브 메소드 스택인데, 이곳은 C 등 네이티브 언어로 작성된 프로그램이 사용하는 스택 부분을 의

‡이 논문은 한국학술진흥재단 지역대학우수과학자 지원에 의해 연구되었음 (R05-2004-000-10967-0)

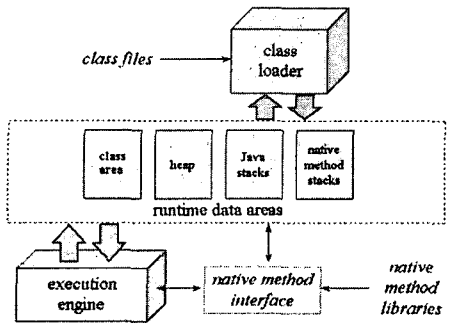

그림 1 자바 메모리

미하며, 자바 프로그램 실행과는 큰 상관이 없으므로 본 연구에서는 제외한다.

3 필드의 사용 분석

3.1 필드의 읽기/쓰기

서론에서 언급한 바와 같이 필드는 클래스의 성질 또는 속성을 나타내는 중요 부분이며, 메소드는 이 속성을 읽거나 변경시키는 목적으로 사용된다.

직관적으로 생각해 볼 때 속성값을 변경하는 일은 흔치 않다. 속성이 바뀐다는 것은 그 클래스의 성질이 크게 달라지는 일이기 때문이다.

예를들어 java.util.Date 클래스를 생각해보자. Date 클래스의 인스턴스를 만드는 시점에서 날짜가 설정되며, 이 날짜는 Date 클래스의 필드에 저장된다. Date 클래스는 날짜를 읽는 getTime() 메소드와 날짜를 변경하는 setTime() 메소드를 모두 가지고 있지만, 현실적으로 setTime() 메소드가 빈번히 호출될 확률은 매우 낮다. 한번 정해진 날짜를 자주 바꾸는 일은 실제 상황에서 매우 드물기 때문이다. 하지만 현재 날짜를 읽어야 하는 프로그램은 굉장히 많으며, 따라서 getTime() 메소드는 실제 상황에서 빈번히 사용될 것이다. 즉 Date 클래스의 필드는 주로 읽기 목적으로 사용되며, 쓰기 동작은 거의 일어나지 않을 것으로 예상된다.

java.lang.String 클래스는 더욱 그렇다. String 클래스는 애초부터 immutable 클래스이다. 즉 처음 정해진 값을 바꿀 수 없다. String 클래스의 인스턴스가 생성될 때 문자열은 필드에 저장되는데, String 클래스가 가진 메소드들, 즉 charAt(), endsWith(), indexOf(), length(), startsWith(), valueOf() 등에서 볼 수 있듯이 이들은 값을 읽기만 할 뿐 변경하지는 않는다. 즉 String 클래스의 필드는 읽기 목적으로만 사용되며, 쓰기 동작은 인스턴스 생성 당시 1회를 제외하고는 일어나지 않는다.

API 클래스들은 물론 여타 사용자 클래스도 마찬가지다. 어느 프로그램이나 객체의 속성을 빈번하게 변경하는 일은 드물며, 만일 그렇다면 그 프로그램은 잘 작성된 프로그램으로 보기 어렵다.

결론적으로 우리는 자바 프로그램에서 필드는 읽기 위주(Read-Mostly)로 사용된다고 예상할 수 있다. 즉 자바에서 힙 메모리는 쓰기보다는 읽기 목적으로 사용된다고 볼 수 있으며, 4장에서 실제 실험을 통해 이 가정을 확인하고자 한다.

3.2 필드의 접근 빈도

필드의 사용과 관련하여 예상할 수 있는 또 다른 가정은 프로그램의 실행 도중에 필드에 접근하는 횟수가 그리 많지 않을 것이라는 것이다. 이것을 다른 표현으로 나타낸다면 힙 메모리를 읽거나 쓰는 횟수가 많지 않다는 것이다.

필드는 본래 앞에서 언급한 바와 같이 클래스의 속성을 나타내는 목적으로 사용되지만, 이것은 또한 클래스 내에 포함된 모든 메소드들이 공통적으로 사용하는 전역변수와 같은 기능도 갖고 있다. C 언어 프로그래밍 등에서 알 수 있듯이 전역변수의 주요 기능은 여러 함수들이 상호간 정보를 주고받게 하는 통신 수단이다. 지역 함수의 계산 도중에 전역변수가 직접 사용될 가능성은 낮으며, 모든 계산은 지역 변수에서 일어나고 최종 결과가 산출된 후에야 비로소 전역변수에 반영되는 것이 일반적이다.

그림 2의 C 프로그램을 고찰해보자. 여기서 sum 은 전역변수임을 발견할 수 있다. sum 의 값을 변경하는 방법으로 update() 에서와 같은 방법을 사용할 수도 있겠지만, 대부분의 C 프로그래머들은 update2() 의 경우와 같은 방식을 취한다. 두 함수의 결과는 동일하지만 update() 에서는 for-loop 이 돌 때마다 전역변수 값이 달라지며, update2() 에서는 한번만 달라진다. update() 에서는 전역변수가 n 번만큼 접근되지만 update2() 에서는 2번만 접근된다. update2() 의 경우 지역변수 s 는 메모리가 아닌 레지스터에 할당될 수 있으므로 실행속도 향상도 기대된다.

```
int sum;

void update(int n) {
        int i;

        for (i=0; i<n; i++)
                sum += i;
}

void update2(int n) {
        int i, s = sum;

        for (i=0; i<n; i++)
                s += i;
        sum = s;
}
```

그림 2 전역변수를 사용하는 C 프로그램의 예

자바에서도 마찬가지다. 필드는 클래스 내 모든 메소드들이 공통적으로 사용하는 전역변수의 역할을 하므로 잘 훈련된 자바 프로그래머들은 전역변수, 즉 필드를 직접 접근하기보다는

지역변수를 사용하여 자료 처리를 할 것이다. 꼭 필요한 경우가 아니라면 전역변수인 필드보다는 지역변수를 사용한다. 따라서 필드에 대한 접근 빈도는 지역변수에 대한 접근 빈도에 비해 상대적으로 작을 것이 예상된다.

3.3 필드 접근의 시간별 분포

마지막 세째로 생각할 수 있는 자바 필드 사용 특징은 필드에 대한 접근이 시간적으로 균일하게 이루어지지 않고 특정 순간에 집중되는 형태를 보일 것이라는 것이다.

그림 2의 update2() 에서 알 수 있듯이 전역변수에 대한 접근은 함수 시작 또는 끝 부분에 잠깐 일어나며 반면 지역변수에 대한 접근은 함수의 시작부터 끝까지 균일하게 지속되고 있다. 꼭 이 예제 프로그램에서만 그런 것이 아니라 잘 짜여진 프로그램은 대부분의 계산 과정에서 지역변수를 사용하며, 최초 값을 읽거나 최종 값을 설정할 때만 전역변수를 사용할 것이다. 즉 자바 클래스에서 필드는 앞 절에서 밝힌 바와 같이 전역변수의 역할을 하므로 특정 시간에만 집중적으로 사용되고 나머지 시간에는 지역변수들 위주로 사용될 것으로 예상할 수 있다.

이것을 다른 말로 표현하면 힙 메모리에 대한 접근은 시간적으로 고르게 일어나는 것이 아니라 특정 순간에 집중되는 형태를 띤다는 것이다. 자바에서 힙 메모리가 이런 성질을 갖는다는 것은 다른 논문에서 이미 연구되었으며 [4], 본 절의 내용은 왜 그런 접근 형태를 갖는지를 정성적으로 해석해 본 것이다.

4 실 험

3장에서 고찰한 자바 필드 사용의 특징은 다음과 같이 요약된다.

- 필드는 쓰기보다는 읽기위주(Read-Mostly)로 사용된다. 즉 필드의 값을 변경하는 경우는 많지 않으며, 주로 값을 읽기만 한다.
- 필드는 지역변수배열이나 오퍼랜드 스택 등에 비해 사용빈도가 낮다. 즉 프로그램 실행 중 필드를 직접 접근하는 횟수는 많지 않다.
- 필드의 사용은 시간적으로 균일하게 일어나지 않고 특정 시점에 집중된다. 즉 평시에는 필드가 별로 사용되지 않다가 어느 순간에 갑자기 높은 빈도로 사용된다.

이상의 결론을 확인하기 위해 실제 프로그램에 대해 실험을 시행하였다. 사용한 프로그램은 임베디드 자바 시스템을 위한 벤치마크 프로그램인 CaffeineMark 3.0 이며, 원천코드가 공개되어있는 simpleRTJ 1.4.2/Win32 [5] 자바가상기계를 사용하였다. 자바 컴파일러는 J2SDK 1.4.1_04 이다.

표 1은 CaffeineMark에서 제공하는 5가지의 개별 벤치마크 프로그램 실행에 따른 메모리 접근을 보인 것이다. Locals 는 지역변수들이 놓이는 지역변수배열 메모리를 의미하며, Heap 은 자바 필드가 놓이는 힙 메모리를 의미한다.

먼저 첫 번째 결론인 힙 메모리의 읽기/쓰기 비율을 보면 전체 메모리 접근 회수 중 읽기가 26.9%, 쓰기가 1.9% 였다. 즉 읽기 회수가 쓰기 회수의 14배에 이르며, 이것은 3.1절의 결론과 동일하다. 두 번째 결론인 힙 메모리의 사용빈도를 보면 전체 메모리 접근의 71.2%가 지역변수배열 메모리에 집중되었으며, 힙 메모리는 28.8%로 나타났다. 벤치마크 프로그램에 따라 차이가 많으며, 특히 Logic, Method 에서는 이 비율이 각각 97.7:2.3, 87.5:12.5 로 힙 메모리 접근이 매우 드물었다. 세 번째 결론은 본 실험에서 시행하지 못하였으나 다른 연구 결과에서 힙 메모리의 사용을 시간대별로 분석하였으며 [4], 이 결과는 필드 사용에도 그대로 적용할 수 있으므로 3.3절의 결론도 잘 맞아짐을 알 수 있다.

표 1 CaffeineMark 메모리 접근 회수 (단위: 1,000회)

Benchmark	Locals		Heap	
	Read	Write	Read	Write
Sieve	41910	1062	21468	74
Loop	37637	6454	25890	2607
Logic	26681	20230	1125	1
String	36481	282	24623	277
Method	26313	4451	2226	2182
All	177356	33452	79569	5836
Percent(%)	59.9	11.3	26.9	1.9

5 결 론

본 논문에서는 전형적 자바 프로그램에서 필드가 어떻게 사용되는지를 정성적으로 분석하였으며 실제 실험을 통해 분석 결과를 확인하였다. 연구 결과 필드는 쓰기보다는 읽기위주로 사용되며, 접근 빈도는 커지 않고, 또한 시간적으로 불균일하게 사용된다는 것을 발견하였다. 필드는 자바가상기계의 힙 메모리에 저장되므로 필드의 사용 형태는 힙 메모리의 사용 형태에 직접적인 영향을 미친다. 본 연구의 결과는 효율적인 힙 메모리 구조 개발에 적용될 것이며, 궁극적으로 자바가상기계의 성능향상에 큰 역할을 담당할 수 있을 것으로 기대한다.

참 고 문 헌

[1] T. Lindholm and F Yellin, *The Java Virtual Machine Specification*, Second Edition, Addison-Wesley, 1999

[2] J. Engel, *Programming for the Java Virtual Machine*, Addison-Wesley, 1999

[3] 양희재, 자바가상기계, 한국학술정보(주), 2001년 3월, ISBN 89-5520-342-4

[4] 양희재, "에너지 관점에서 임베디드 자바가상기계의 메모리 접근 형태," 한국정보처리학회 논문지, 12-A권 3호, 2005. 6.

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