simpleRTJ 클래스 파일의 형식 분석

양희재

경성대학교

Analysis of the simpleRTJ Class File Format

Heejae Yang

Kyungsung University

E-mail : hjyang@star.kyungsung.ac.kr

요 약

내장형 시스템은 데스크톱 시스템과 달리 메모리 사용상 큰 제한을 받는다. 자바 프로그램 실행을 위해서는 클래스 파일들이 메모리 상에 배치되어져야 하는데, 클래스 파일은 내부에 상수풀, 클래스 정보, 필드 정보, 그리고 메소드 정보 등을 갖는다. 이 정보들 중 어떤 것들은 디버깅 등의 목적으로 사용되며 또 어떤 것들은 실제 프로그램 실행을 위해 사용되어진다.

본 연구에서는 내장형 자바 시스템을 위한 클래스 파일들의 내부 정보, 즉 형식에 대해 분석해보고, 그것이 메모리 상에 배치되었을 때 요구되어지는 메모리의 양 등을 해당 정보별로 조사해보았다. 실험은 원천코드가 공개되어져있는 상용제품인 simpleRTJ 내장형 자바 시스템에 대해 이루어졌다.

ABSTRACT

Unlike desktop systems, embedded systems usually meet a strict restriction on using memory. It is required to allocate several class files on memory to run a Java program. A Java class file contains such data including a constant pool, class overview, fields information, and methods information. Some of them are used merely for a debugging purpose, others for a program execution.

This paper analyzes the internal structure, or format of the class files used for embedded Java systems. We also investigate how much memory will be necessary for each part of the class files when the files are allocated in memory. The experiment was performed on the simpleRTJ, an open-source commercial embedded Java system.

키워드

내장형 시스템, 자바가상기계, 클래스 파일, J2ME

Ⅰ. 서 론

내장형 시스템에 자바 기술을 적용하는 시도가 급격히 증가하고 있다. 이것은 자바가 갖는 플렛폼 독립성, 안전성, 객체지향성 등이 다양한 플렛폼을 가지고 안전성을 요구하며 복잡한 프로그래밍을 필요로 하는 내장형 시스템에 잘 맞아지기 때문인 것으로 해석된다 [1].

하나의 자바 프로그램은 여러 개의 클래스 파일들로 구성된다. 클래스 파일은 특정 클래스에 대한 모든 정보를 포함하고 있으며, 상수풀(constant pool), 클래스 개요, 필드 정보, 메소드 정보 등으로 구성된다. 클래스 파일에는 프로그램 실행에 꼭 필요한 정보 외에도 디버깅 등의 목적을 위한 정보들도 포함되어있다 [2].

자바 프로그램의 실행을 위해서는 클래스 파일 내용들이 메모리 상에 배치되어야 한다. 내장형 시스템의 경우 메모리의 양에 대한 제한이 엄격하므로 클래스 파일의 내용 중 디버깅을 위한 정보, 클래스 상호 연결을 위한 정보 등은 최소화하여 메모리에 배치할 필요가 있다.

일반적으로 클래스 파일들은 두 종류로 나눌 수 있

다. 한가지는 특정 응용 프로그램을 위해 프로그래머에 의해 작성된 것이며, 또다른 한가지는 여러가지 응용 프로그램에 공통으로 사용될 수 있는 클래스들, 즉 클래스 라이브러리에 포함된 것들이다.

클래스 라이브러리(class library)는 여러 자바 개발자들에 의해 만들어지며, 대표적인 것으로는 썬 마이크로시스템사의 라이브러리를 들 수 있다. 썬 마이크로시스템사에서는 자바의 적용 분야에 따라 자바 환경을 J2EE (Enterprise Edition), J2SE (Standard Edition), 및 J2ME (Micro Edition) 으로 구분하고 있는데, 각 환경마다 저마다의 클래스 라이브러리를 갖는다. 또다른 회사들은 자신들의 고유 클래스 라이브러리를 제공하기도 한다.

본 연구에서는 특히 내장형 시스템을 위한 자바 환경에서 제공되는 클래스 라이브러리에 포함된 클래스 파일들에 대해 고찰해보고자 한다. 내장형 시스템의 특성 상 클래스 파일에도 제약이 가해지는 것이 일반적이다. 우리가 관심을 가지는 부분은 내장형 시스템을 위한 클래스 라이브러리의 클래스 파일들은 어떤 종류들이 있으며, 그들의 상속관계, 즉 클래스 계층구조는 어떠하며, 클래스 파일의 크기는 평균 몇 바이트인지 등에 대해 알아 보는 것이다. 이와 함께 그 클래스들의 내부 구조, 즉 필드와 메소드의 수효, 각 메소드를 위한 스택공간의 크기, 지역변수배열의 크기, 그리고 각 메소드의 평균 코드길이 등에 대해서도 조사해 보고자 한다.

이 연구는 내장형 시스템과 같이 메모리 자원의 제약을 받는 환경에서 각각의 클래스들이 어떤 특성을 갖는지에 대한 이해를 돕게 한다. 아울러 클래스들이 메모리에 배치되었을 때 요구되는 메모리의 양은 어느 정도인지에 대한 분석도 가능하게 한다. 또한 내장형 시스템을 위한 자바가상기계의 구현시에도 필드나 메소드의 특성을 이해하면 더욱 효과적인 자바가상기계의 설계가 가능해 질 수 있을 것이다.

많은 내장형 자바 시스템이 있지만, 본 연구에서는 특히 클래스 및 자바가상기계의 원천코드가 공개되어져 있는 상용 내장형 자바 시스템인 simpleRTJ [3] 의 경우에 대해 분석을 해 보았다. 분석 결과 simpleRTJ 클래스 라이브러리에 포함된 클래스들과 썬 마이크로시스템사의 대표적 내장형 클래스 라이브러리인 CLDC [4] 에 포함된 클래스들간에 매우 큰 유사성이 있음을 확인할 수 있었다. 즉 개발회사는 다르더라도 내장형 시스템을 위한 자바 클래스 파일은 서로 큰 유사성이 있다는 것이다. 따라서 본 연구의 결과는 단순히 simpleRTJ 라고 하는 특정 환경에서만 적용될 수 있는 것이 아니라 일반적인 내장형 자바 시스템에도 적용될 수 있으리라 기대해 볼 수 있을 것이다.

본 연구에서 우리는 다음과 같은 결론을 얻었다.

1) 클래스 라이브러리에 포함된 클래스의 절반 이상은 예외 및 오류처리를 위한 것이다.

2) 대부분의 일반 클래스들은 클래스 상속계층상 3단계 이내에 위치하며, 예외 및 오류 클래스들은 3단계부터 6단계까지에 걸쳐 놓인다.

3) 클래스 파일의 크기는 평균 902 바이트이며, 각 클래스들은 평균 1.58개의 필드와 7.28개의 메소드를 갖는다. 4) 클래스에 포함된 메소드들이 필요로 하는 스택공간의 크기는 2.25이며, 지역변수배열의 크기는 1.90 이다. 그리고 한 메소드의 평균 코드길이는 20 바이트이다.

본 논문의 구성은 다음과 같다. 2절에서는 simpleRTJ 클래스 라이브러리에 포함된 클래스들의 구분 및 상속계층에 대해 알아보며, 3절에서는 클래스 파일의 형식에 대해 분석해본다. 4절에서 결론 및 향후 연구방향에 대해 설명한다.

II. simpleRTJ 클래스 라이브러리 구성

simpleRTJ 는 내장형 시스템을 위한 자바 환경이므로 데스크톱 시스템을 위한 자바 환경과는 많은 차이점을 갖는다. 예를들어 java.awt 등 그래픽 관련 클래스 패키지들은 아애 포함되어있지 않다. 또한 내장형 시스템의 경우 파일 시스템이 없는 경우가 많고, 일반적인 입출력 장치보다는 직렬포트에 의한 입출력 등 특정 플랫폼을 위한 한정된 장치만 있는 경우가 많다. 따라서 simpleRTJ 는 java.io, java.net 등의 패키지들도 포함하지 않고 있다.

그외 simpleRTJ 는 64비트 정수형이나 배정밀도 실수형 자료를 사용하지 않고 있으며, JNI (Java Native Interface) 도 제공하지 않는다.

simpleRTJ 는 내장형 시스템 작동에 필요한 핵심 클래스들만 기본적으로 제공하고 있으며, 이들은 java.lang 및 java.util 패키지에 포함되어있다. java.lang 에는 41개의 클래스 및 2개의 인터페이스가 있고, java.util 에는 2개의 클래스와 1개의 인터페이스가 있다. 본 논문에서는 인터페이스를 제외한 총 43개의 클래스들에 대해 연구했다.

이들 클래스들을 종류별로 구분하면 다음과 같다. 먼저 System 클래스들은 Object, String, StringBuffer, System, Thread, Throwable 등 6개가 있으며, Data 클래스들로는 Boolean, Byte, Short, Character, Integer, Float, Number 등 7개가 있다. 그외 Math 클래스로 Math, Utility 클래스로 StringTokenizer 등 각각 1개씩 있다.

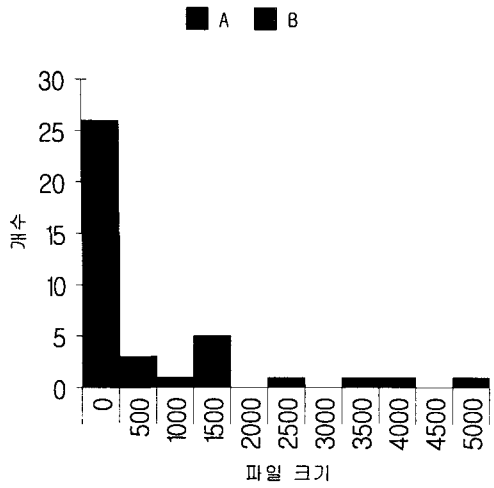

그림 1 클래스 파일 크기(바이트)

A: 일반 클래스 B: 예외/오류 클래스

이상 일반 클래스들이 총 15개이며, 나머지 28개는 모두 예외/오류 클래스들에 해당된다. 즉 클래스 라이브러리에 포함된 클래스들 중 절반 이상인 65% 가 예외/오류 클래스들인 것이다. 예외 클래스, 즉 클래스 계층구조상 Exception 이하의 클래스들은 모두 16개이며, 오류 클래스, 즉 Error 이하의 클래스들은 모두 12개이다.

이들 클래스들을 클래스 계층구조로 분류해보면, 최상위에 놓인 Object 클래스 밑에 2단계로 10개의 클래스들이 있으며, 일반 클래스들은 대부분 이 단계에 해당된다. 3단계에는 나머지 일반 클래스 4개와 Error, Exception 등 6개의 클래스들이 있으며, 4단계 이하는 예외/오류 클래스들이 차지하고 있다. 즉 4단계에 5개, 5단계 16개, 6단계에 5개의 예외/오류 클래스들이 놓인다.

객체지향 개념에 의하면 하나의 클래스가 생성될

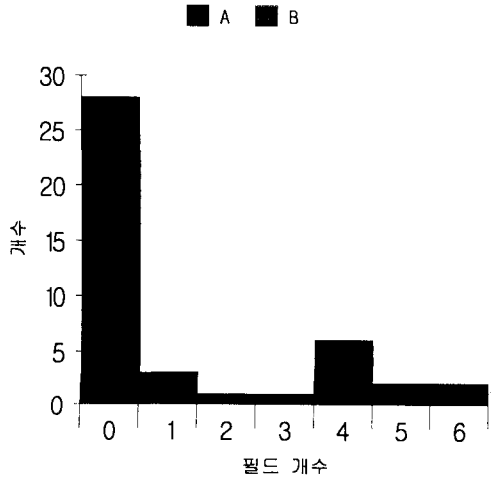

그림 2 필드 개수 분포

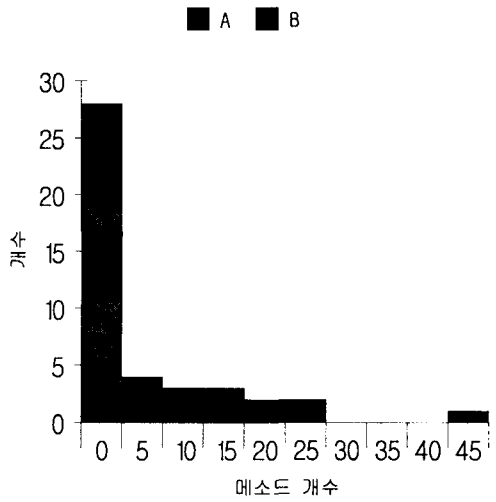

그림 3 메소드 개수 분포

때는 그것의 상위 클래스들도 모두 참조되어야 하므로 하위 계층에 놓인 클래스의 수가 많을수록 메모리의 활용상 좋지 않을 수도 있다. simpleRTJ 에서는 예외/오류 클래스들 외에는 모두 2,3단계에 배치하므로서 메모리의 활용도를 높일 수 있게 하고 있다.

III. simpleRTJ 클래스 파일의 형식 분석

3.1 클래스 파일의 크기

일반적 자바 환경에서는 필요한 클래스들이 동적으로 적재되어진다. 따라서 클래스 파일의 크기가 너무 크면 적재에 따른 시간이 오래 소요되어 응답성이 떨어지게 되므로 클래스 파일의 크기를 어느 정도 제한할 필요가 있다. 적재시간 외에도 클래스 파일의 크기는 메모리 소비량과도 밀접한 관계를 가지므로 내장형 시스템에서는 그 크기를 최소화 할 필요가 있다.

그림 1은 simpleRTJ 클래스 파일들의 크기 분포를 나타낸 것이다. 그림에서 알 수 있듯이 예외/오류 클래스들은 전부 500바이트 이하를 차지하고 있으며 (평균 335, 표준편차 67.57), 일반 클래스들의 경우 2,000바이트 이내가 다수이다 (1,959/1,469 - 평균/표준편차; 이하 동일표기). 전체적인 평균은 902바이트이며, 표준편차는 1,164이다. 이 값은 데스크톱 자바 환경에서의 평균값인 4,541바이트에 크게 못미치는 것으로서, 메모리가 한정적인 내장형 시스템의 특성이 반영된 것이라 볼 수 있다.

3.2 필드 및 메소드 개수

어떤 클래스의 인스턴스가 생성될 때는 그 클래스가 갖고 있는 필드 및 자신의 상위 클래스들이 갖고

있는 필드들을 저장할 수 있는 메모리 공간이 필요하므로 필드의 개수는 메모리 사용량에 직접적 영향을 미친다. 메소드 개수와 메모리 사용량과의 관계는 명확치 않지만, 메소드가 많을수록 메소드 테이블 엔트리 수도 늘어나므로 관련성이 있을 것으로 예상된다.

먼저 필드의 개수에 대해 알아보자 (그림 2). 이 그림에서 볼 수 있듯이 예외/오류 클래스들은 모두 필드들을 갖지 않았다 (0/0) 일반적으로 정상 상태에서 예외/오류 사건은 거의 일어나지 않으며, 일어난다고 하더라도 해당 클래스들은 필드들을 갖지 않으므로 필드로 인해 메모리 사용에 미치는 영향은 없을 것으로 보인다.

반면 일반 클래스들은 0개부터 6개까지의 필드를 가지며, 그 분포도 넓게 퍼져 있음을 볼 수 있다 (4.53/7.78). 전체적인 평균은 1.58개의 필드를 갖는 것으로 조사되었으며, 표준편차는 5.08 이다.

메소드 개수의 분포에 대해서는 그림 3에 나타내었다. 특기할만한 점은 예외/오류 클래스들의 경우 전부가 5개 미만의 메소드를 갖는다는 것이다 (2.07/0.26). 즉 이들 클래스들은 필드도 메소드도 거의 갖고 있지 않은 적은 크기의 클래스들인 것이다. 실제로 이들 클래스들의 대부분은 생성자 메소드 (constructor method) 들만 2개 정도 가지는 것으로 조사되었다.

일반 클래스들의 경우 5개부터 30개까지의 메소드를 갖는 경우가 많고, 그 분포도 비교적 넓게 퍼져 있음을 볼 수 있다 (17/10.48). 전체적인 평균은 7.28개의 메소드를 갖는 것으로 조사되었으며, 표준편차는 9.43 이다.

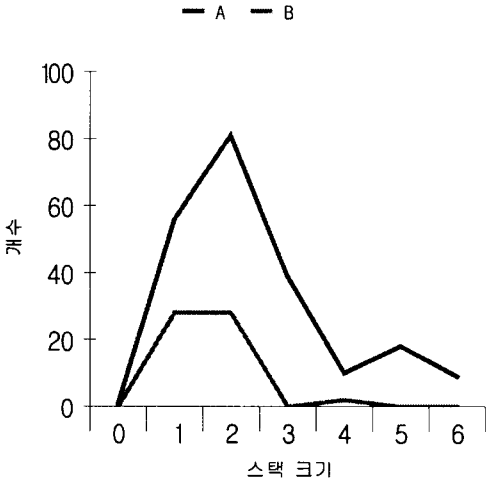

그림 4 스택 크기의 분포

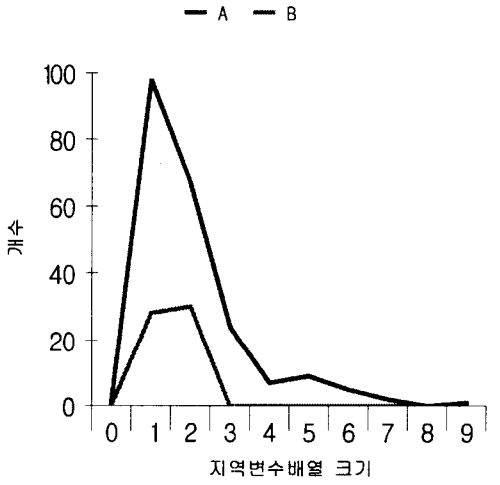

그림 5 지역변수배열 크기의 분포

3.3 메소드 정보

조사대상인 43개의 클래스들이 갖고 있는 메소드들의 총 개수는 272개로 나타났다. 이들 중 일반 클래스들의 메소드는 214개이었고, 예외/오류 클래스들의 메소드는 58개로 조사되었다. 예외/오류 클래스의 개수는 전체 클래스의 65% 를 차지하지만 그들이 갖고 있는 메소드의 수는 전체의 21% 에 불과했다. 이 클래스들이 갖는 메소드 개수가 작은 것은 3.2 절에서 이미 밝힌 바 있다.

이 절에서는 각 메소드들의 정보, 즉 필요로 하는 스택의 크기, 지역변수배열의 크기, 그리고 코드 길이 등에 대해 알아보자.

어떤 메소드가 호출되면 이 메소드를 위한 별도의 스택이 할당되므로 스택의 크기는 메모리 사용량에 직접 영향을 미친다. 그림 4는 스택 크기의 분포를 나타낸 것이다. 예외/오류 클래스의 메소드들은 대개 1-2개의 스택 크기를 (1.59/0.67), 일반 클래스의 메소드들은 1-4개의 스택 크기를 (2.43/1.38) 요구하는 것을 볼 수 있다. 전체적 평균은 2.25 이며, 표준편차는 1.31 이다. 자바가 스택기반의 실행환경을 가지지만, 메소드당 필요한 스택의 크기는 그리 크지 않음을 알 수 있다.

지역변수배열은 메소드 호출시 각종 인자 전달 및 메소드 내의 지역변수를 저장하는 목적으로 사용된다. 그림 5를 보면 메소드들은 대개 1-3개 정도의 지역변수를 저장할 수 있는 지역변수배열을 필요로 하는 것을 볼 수 있다. 예외/오류 클래스들의 메소드와 (1.52/0.50) 일반 클래스들의 메소드 간에도 큰 차이가 없었다 (2.01/1.39). 전체적인 평균은 1.90, 표준편차는 1.27로 나타났다.

마지막으로 그림 6은 각 메소드들이 갖는 코드의 크기를 보여주는 것이다. 20바이트 이하의 크기를 갖는 메소드들이 많으며, 일반 클래스들의 메소드는 100 바이트를 넘어서는 것들도 꽤 있는 편이다. 오류/예외 클래스의 메소드들은 평균 6.03바이트의 코드 크기를 가지며, 표준편차는 2.87이다. 일반 클래스들의 메소드

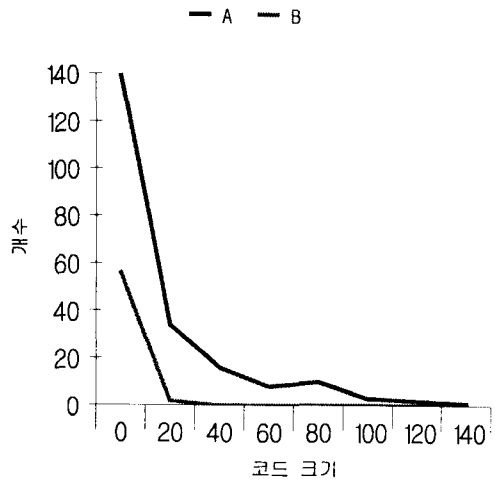

그림 6 코드 크기의 분포

들은 평균 24.22, 표준편차 28.98 이며, 전체적인 평균은 20.35, 표준편차 26.79로 조사되었다.

IV. 결 론

본 논문에서는 상용 내장형 자바 시스템의 일종인 simpleRTJ 가 제공하는 클래스 라이브러리에 포함된 클래스들을 분석해보았다. 즉 이 클래스들의 종류별 구분, 계층구조, 파일의 크기, 필드 및 메소드의 개수, 각 메소드들이 필요로 하는 스택과 지역변수배열의 크기, 그리고 코드크기 등에 대해 조사했다.

이 연구는 simpleRTJ 라고 하는 특정 플렛폼에 대해 이루어진 것이지만, 내장형 자바 시스템을 위한 클래스들이 대체적으로 어떤 특징을 갖는다는 것을 짐작하게 한다. 썬 마이크로시스템사의 내장형 자바 환경인 J2ME 의 CLDC 클래스들에 대한 조사도 본 연구와 유사한 결과를 가지는 것으로 조사되었다. 즉 이 연구를 통해 우리는 내장형 자바 시스템을 위한 클래스의 구성 및 형식에 대해 일반적 해석을 할 수 있게 된 것이다.

이 연구의 결과는 내장형 시스템이 필요로 하는 메모리 양의 예측, 특히 자바가상기계의 설계시 요구되어지는 자바 스택, 지역변수배열 등의 크기를 얼마로 할 것인가 등에 도움을 줄 수 있을 것으로 기대된다. 또한 클래스 적재에 따른 지연시간 해석, 클래스의 인스턴스를 만들 때 소요되는 메모리의 크기 예측, 필요한 시간 예측 등에도 사용될 수 있을 것이다.

참고문헌

[1] S. Helal, "Pervasive Java," IEEE Pervasive Computing, pp.82-85, Jan-Mar 2002

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

[3] RTJ Computing, simpleRTJ: a small footfrint Java VM for embedded and consumer devices, http://www.rtjcom.com

[4] Sun Microsystems, Connected, Limited Device Configuration Specification, Version 1.0a, May 2000