자바가상머신을 위한 클래스 로더 시스템 설계 및 구현

유용선°, 류현수, 김용희, 이철훈
충남대학교 컴퓨터공학
{ysryu°, ysyoo, yonghee, chlee}@flower.ce.cnu.ac.kr

The design and Implementation of Class Loader System for Java Virtual Machine

YONG SUN RYU°, HYUN SOO RYU, YONG HEE KIM, CHEOL-HOON LEE
Dept. of Computer Engineering, Chungnam National Univ.

요 약

최근 무선인터넷이 발달함에 따라, 핸드폰이나 PDA 및 정보가전용 기기들에 플랫폼 독립성, 이식성, 네트워크 이동성의 장점을 갖는 자바기술을 적용한 서비스가 증가하고 있다. 자바 플랫폼은 각각의 하드웨어 플랫폼에 맞게 포팅된 가상머신이 존재하여, 컴파일된 바이트 코드를 해석하기 때문에 플랫폼 독립성을 갖게 된다. 그러나 sun사의 JVM을 사용할 경우 고가의 *royalty*를 지불해야 하기 때문에 경쟁력을 높이기 위해 "클린룸(Clean Room)"에서 개발한 국산 KVM이 절실하며, 이에 국내에서 활발한 연구가 이루어지고 있는 실정이다. 본 논문에서는 자바가상머신의 일부분인 클래스 로더 시스템에 관해 설계 및 구현한다.

1. 서 론

기존의 C/C++의 문제점은 칩마다 기계어 코드가 달랐기 때문에 한 플랫폼에서 짠 프로그램이 다른 플랫폼에서는 동일하게 수행되는지 보장 할 수 없고, 또 앞으로 새로운 플랫폼이 등장했을 때 역시 그 실행 여부를 보장할 수 없다는 것이었다. 이러한 문제점을 해결하기 위해 JAVA는 하드웨어 플랫폼상에 자바가상머신을 두어, 컴파일된 코드를 자바 인터프리터를 통해 바이트 코드로 변환된 명령어(instruction)들을 수행하게 된다. 따라서, 하드웨어 플랫폼에 자바가상머신이 탑재하게 되면, 플랫폼에 관계없이 컴파일된 자바코드는 동일하게 작동된다.

자바기술은 플랫폼 독립성 이외에, 네트워크 이동성, 보안 등의 장점을 갖고 있으며, 최근들어 무선인터넷 서비스를 제공하는 핸드폰, PDA 및 정보가전용 기기들에 적용되고 있다. 하지만, Sun사의 자바가상머신을 각종 기기들에 포팅하게 될 때, 로열티를 지불하는 문제가 발생할 수 있다. 따라서, 자바 스펙(Specification)에 맞는 독자적인 자바가상머신의 개발이 필요하다.

본 논문에서는 자바가상머신에서 바이트 코드를 해석하는 클래스 로더 시스템에 관해 설계 및 구현한다.

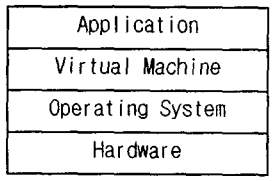
본 논문의 2장은 관련연구로써 자바가상머신의 전반적인 구조를 살펴보고, 3장에서는 자바 스펙에서 정의된 클래스 로더 시스템에 관해 설계 및 구현한다. 마지막으로 4장에서는 결론 및 향후연구과제를 기술한다.

* 본 논문은 한국과학재단이 지정한 지역협력연구센터(ARC)인 충남대학교 소프트웨어연구센터의 지원으로 수행된 과제의 결과입니다.

2. 관련연구

2.1 가상머신이란?

가상머신(Virtual Machine)은 추상적인 개념으로 하드웨어 또는 운영체제에 독립적인 실행엔진이나 컴퓨터 구조를 갖는 Software machine을 뜻한다. 그 특징으로 플랫폼 독립성을 제공, 소프트웨어의 동적 다운로딩, 호스트 운영체제의 메모리에 대한 추상적인 접근 등이 있다.

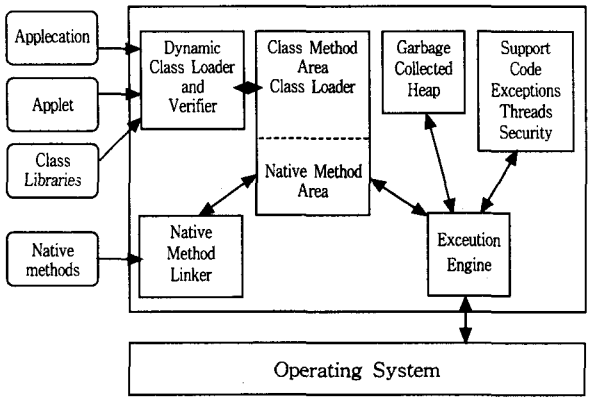


[그림1] Typical High-Level Architecture

2.2. 자바가상머신

자바가상머신은 "바이트 코드를 실행시킬 수 있는 소프트웨어로 만들어진 운영체제"로 자바가상머신 때문에 플랫폼에 관계없이 한번 컴파일된 자바 프로그램을 재컴파일없이 사용할 수 있다. 그리고, 이식성을 위해 플랫폼에 관계없이 데이터(Int, long, double)의 크기도 매우 엄격하게 고정되어 있다. 모든 오피코드(opcode)는 1byte의 크기를 갖고 있고 자바가상머신이 모든 코드를 byte-to-byte방식으로 실행한다.

자바가상머신은 내부적으로 클래스 로더 시스템, 런타임 데이터 영역, 실행엔진으로 구성되며, 컴파일된 코드를 인터프리터를 통해 마치 자신이 가상의 기계인 것처럼 바이트 코드로 변환된 인스트럭션(instruction)을 수행한다.



[그림2] 자바가상머신 내부 구조

2.2.1 클래스 로더 시스템(Class Loader System)

타입을 찾고 로딩하는 자바가상머신 구현의 일부분이다. 클래스 로더 시스템은 클래스를 위한 이진 데이터를 로딩하거나 임포팅(importing)한다. 뿐만 아니라, 임포트된 클래스의 정확성을 검증하고, 클래스(static) 변수를 위한 메모리를 할당하고 초기화하며, 심벌 레퍼런스의 분해과정(resolution)을 돕는다. 이러한 행동은 로딩, 링킹, 초기화 과정을 거쳐 이루어진다

2.2.2 런타임 데이터 영역(Runtime data areas)

자바가상머신은 프로그램이 실행하는 동안 사용되는 여러 개의 런타임 데이터 영역을 정의하며, 이런 데이터 영역의 일부는 자바가상머신이 시작될 때 할당되고, 자바가상머신이 종료될 때 반환된다. 다른 데이터 영역은 쓰레드(thread)가 생성될 때 할당되고 쓰레드가 종료될 때 반환된다.

- 메소드 영역(Method area)

로드된 타입에 대한 정보는 메소드 영역이라 불리는 메모리의 논리(logical) 영역에 저장된다. 자바가상머신은 타입을 로드하고, 로드된 타입은 클래스 로더를 통해 x.class 화일(이진 데이터 스트림)을 읽어 가상머신에게 전달한다. 가상머신은 이진 데이터로부터 타입에 대한 정보를 추출하여 메소드 영역에 그 정보를 저장한다. 클래스에서 선언된 클래스(static) 변수들을 위한 메모리로 메소드 영역에 저장한다.

- Heap

자바 애플리케이션을 실행할 때 새로운 객체(인스턴스, 배열)를 저장하는 메모리로 모든 쓰레드들이 이 영역을 공유한다. 따라서, thread-safe을 보장하기 위한 방법이 필요하며, 자바가상머신은 힙을 관리하기 위해 가비지 컬렉션을 수행한다.

- 자바 스택(Java stacks)

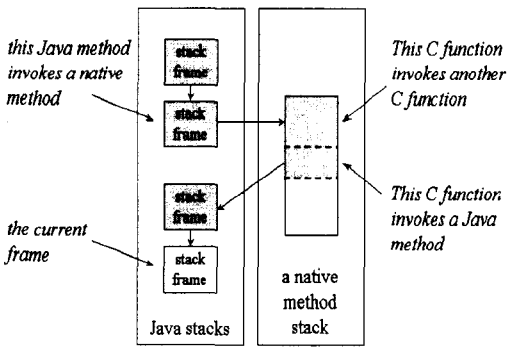
프로그램이 수행 도중 어떤 메소드가 호출되었는지를 기록하고, 각 메소드의 호출과 관련된 데이터의 정보를 기록한다. 새로운 쓰레드가 실행될 때, 자바가상머신은 쓰레드를 위한 새로운 자바 스택을 생성하며, 두개의 연산(pushing, popping)으로 메소드가 호출될 때 스택 프레임(Stack Frame)을 푸시하며, 종료시에는 팝한다.

- PC 레지스터 (Program Counter registers)

실행중인 프로그램 각각의 쓰레드는 그 쓰레드가 시작될 때 생성된 자신의 PC 레지스터를 가진다. PC 레지스터는 1-word 크기이며, 네이티브 포인터와 복귀주소를 가진다. 쓰레드가 자바 메소드를 실행할 때, PC 레지스터는 쓰레드에 의해 실행되는 현재 명령어(current instruction)의 주소를 포함한다. 이 주소는 네이티브 포인터나 메소드 바이트코드의 시작으로부터 옵셋(offset)이 된다.

- 네이티브 메소드 스택 (Native method stacks)

쓰레드가 네이티브 메소드를 호출할 때 생성되는 스택으로 자바가상머신의 구조나 보안성 제약을 받지 않는다



[그림3]Java와 native methods를 invoke하는 thread를 위한 stack

2.2.3 실행엔진(Execut ion eng ine)

자바가상머신 구현의 핵심부분으로 실행엔진의 수행은 인스트럭션 집합이며, 각각의 인스트럭션들은 정의되어 있다. 자바가상머신 명세서에서는 실행엔진을 추상적인 명령어 집합, 하드웨어 또는 소프트웨어 구현, 런타임 인스턴스(thread)로 정의한다.

3. 자바클래스 로더 시스템 설계 및 구현

클래스 로딩은 jar 파일과 x.class 파일로부터 동적클래스 로딩을 지원하고 있다. 따라서, 전체 심벌(symbolic) 정보들은 동적으로 런타임시에 링킹되어 사용되게 된다. 한편, 클래스 로딩의 성능은 전체 가상머신의 성능과도 밀접한 관계가 있으므로, 이를 어떻게 구성

하는가에 따라 전체 시스템에 대한 성능 향상이 좌우될 수 있다.

3.1 클래스 로딩

클래스 로딩 시기는 세가지로 구분된다.

- 자바 시스템 클래스 로딩

java.lang.Object, java.lang.Class, java.lnag.Class, java.lang.System, java.lang.Thread 등은 다른 모든 클래스들이 로딩되기 전에 이루어져야 한다. 이는 모든 클래스로딩에 있어 기본적인 클래스 및 필수 클래스들로 구분되어 애플리케이션 자바 클래스 등을 로딩하고자 할 때 기본적인 데이터 타입 및 메소드를 제공하는 역할을 한다.

- 지정클래스 로딩

메인 클래스 로딩시, 명시적으로 지정된 클래스들을 로딩하고자 할 때 이루어진다. 클래스의 상태 정보를 이용하여 이미 로딩되어 있는지 아닌지를 구별할 수 있다.

- 클래스 검증시

클래스 할당시 클래스가 할당하려는 타입과 일치하는지를 검사하고자 할 때, 아직 로딩되지 않은 상태의 클래스이면 로딩하여 검사하게 된다. 이에 해당되는 클래스 로딩에 대해 가상머신의 설계상 검증은 사전 검증 시스템으로 대체할 예정으로 이에 해당하는 클래스 로딩은 제외된다.

3.2 이진 데이터 타입의 로딩

이진 데이터 타입은 6개의 소스로부터 로딩될 수 있다

- 로컬 파일 시스템으로부터 로딩
- 네트워크를 통해서 로딩
- Zip, Jar, Cab에서 압축해제를 통해서 로딩
- 데이터베이스에서 로딩
- 자바 파일을 컴파일하여 클래스 파일 형태로 변환하여 로딩
- 컴퓨터 메모리공간에 차지하고 있는 클래스 파일 로딩

클래스 로더는 파일 시스템에 맞게 구현이 되어야 하며, 기본적으로는 C언어의 표준 라이브러리에 정의된 FILE* 구조체를 사용하고 있다.

- 인터프리터가 x.class 요청시 디스크로부터 읽어와 Internal representation을 생성해 준다

```
static ClassClass *
LoadClassFromFile(char *fn, char *dir, char *class_name)
```

- Zip 또는 Jar file에서 x.class file을 로드한다.

```
static ClassClass *
LoadClassFromZip(zip_t *zipEntry, char *class_name)
```

- 클래스 로더를 통하지 않고, 로컬 메모리에 있는 class 파일을 찾는다

```
ClassClass *LoadClassLocally(char *name)
```

- LoadClassFromFile() or LoadClassFromZip()로드된 x.class로부터 Internalrepresentation 생성

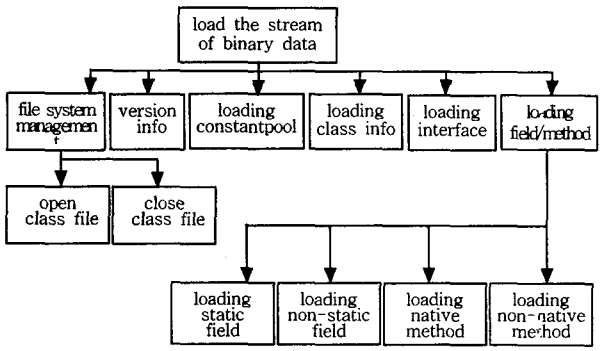
```
bool_t createInternalClass
(unsigned char *ptr, unsigned char *end_ptr,
 ClassClass *cb, struct Hjava_lang_ClassLoader *loader,
 char *name, char **detail)
```

[그림5] x.class로딩에 관련된 클래스 로더 일부 코드

이진 데이터 로딩은 다음과 같은 과정을 가진다.

1. 클래스 버전 정보 확인 (매직넘버:0XCAFEBABE)
2. 컨스탄트 풀 로딩
3. 클래스 정보 로딩
4. 인터페이스 로딩
5. 필드 로딩
6. 메소드 로딩

아래 그림[6]은 위에서 언급한 클래스 로딩 일련 과정의 데이터 흐름을 보여준다.



[그림6] 자바클래스 data 로딩 flow

4. 결론 및 향후과제

본 논문에서는 자바가상머신 명세를 토대로 자바가상머신의 일부분인 클래스 로더 시스템을 구현하였다. 클래스 로더 시스템은 클래스를 위한 이진 데이터를 로딩하거나 임포팅하며, 임포트된 클래스의 정확성을 검증하고, 정적 변수를 위한 메모리를 할당하고 초기화하며, 심벌릭 레퍼런스의 분해과정(resolution)을 수행한다.

하지만, 전반적인 자바가상머신의 개발을 위해서 동적 메모리 관리를 위한 부분과, 보다 빠르게 명령어를 처리할 수 있는 인터프리터, 네이티브 함수를 위한 인터페이스 등이 필요하며, 향후과제로 이러한 모듈들은 구현해야 할 것이다.

참고문헌

[1] Java Virtual Machine Specification : Available at; java.sun.com/doc/book/vmspec/2nd-edition/html/VMSpecTOC.doc.html
[2] Java Virtual Machine Profier Interface : Available at; java.sun.com/j2se/1.3/docs/guide/jvmpi/
[3] java ClassLoader : Available at; java.sun.com/j2se/1.3/docs/api/java/lang/ClassLoader.html
[4] Bill Venners, "Inside the Java Virtual Machine"
[5] ibm.com/developerWorks, Understanding the java ClassLodader