(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 9/445 (2006.01)

(45) 공고일자 2006년12월06일
(11) 등록번호 10-0654428
(24) 등록일자 2006년11월29일

(21) 출원번호 10-2004-0002655
(22) 출원일자 2004년01월14일
심사청구일자 2004년01월14일

(65) 공개번호 10-2005-0074766
(43) 공개일자 2005년07월19일

(73) 특허권자 삼성전자주식회사
경기도 수원시 영통구 매탄동 416

(72) 발명자 박종목
서울특별시강남구대치동901-42201호

(74) 대리인 김동진
정상빈

심사관 : 한선경

전체 청구항 수 : 총 15 항

(54) 자바 프로그램의 처리 속도를 향상시키는 시스템 및 그 방법

(57) 요약

본 발명의 자바 프로그램의 처리 속도를 향상시키는 방법은 클래스 로더가 소정의 저장영역에 저장된 클래스 파일을 소정 크기의 물리적인 단위로 나누어서 메모리로 전송(또는 로딩)하는 단계와, 상기 클래스 로더가 로딩한 물리적인 단위의 클래스 파일을 로드 타임 컴파일러(Load-Time Compiler)가 링킹 및 컴파일하는 단계와, 상기 로드 타임 컴파일러가 링킹 및 컴파일하는 단계에서 상기 수신된 물리적인 단위의 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지를 판단하는 단계 및 상기 포함된 논리적인 단위의 정보를 링킹하고 상기 정보가 자바 메소드(method)일 경우 상기 메소드를 컴파일하는 단계를 포함한다.

이에 따라, 자바 가상 머신에서 클래스 파일의 로딩 과정 및 클래스 파일의 컴파일 과정을 동시에 수행하여, 클래스 파일 로딩 시간내에 컴파일 과정이 수행되도록 함으로써, 애플리케이션의 전체 수행 성능을 향상시킬 수 있는 효과가 있다.

대표도

도 5

특허청구의 범위

청구항 1.

자바 프로그램의 클래스 파일을 로딩하는 클래스 로더; 및

상기 클래스 로더로 로딩된 클래스 파일에 다음으로 처리할 논리적인 단위의 정보가 포함되어 있는지를 판단하고, 상기 판단 결과 논리적 단위의 정보가 포함된 경우 컴파일을 수행하는 로드 타임 컴파일러를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 2.

제 1항에 있어서,

상기 클래스 파일은 소정 크기의 물리적인 단위로 전송되는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 3.

삭제

청구항 4.

제 1항에 있어서,

상기 논리적인 단위의 정보는 클래스 헤더, 컨스턴트 풀, 인터페이스, 필드 및 메소드 중 어느 하나인 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 5.

제 1항에 있어서,

상기 로드 타임 컴파일러는,

상기 논리적인 단위의 정보를 컴파일하는 과정에 클래스 파일이 올바르게 구성되었는지를 검증(verification)하는 과정을 수행하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 6.

제 1항에 있어서,

상기 로드 타임 컴파일러는,

상기 논리적인 단위의 정보를 컴파일하는 과정에 클래스 변수들의 초기값 설정을 하는 준비(preparation) 과정을 수행하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 7.

제 1항에 있어서,

상기 로드 타임 컴파일러는,

상기 논리적인 단위의 정보를 컴파일하는 과정에 다른 클래스의 메소드나 필드 등에 대한 참조 값들이 있을 경우 이들을 분해(resolution)하는 과정을 수행하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 8.

제 1항에 있어서,

상기 클래스 로더 및 로드 타임 컴파일러는 각각 별도의 쓰레드를 이용하여 동시에 동작을 수행하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 시스템.

청구항 9.

소정의 저장영역에 저장된 클래스 파일을 클래스 로더가 소정 크기의 물리적인 단위로 메모리에 전송하는 단계;

상기 수신된 물리적인 단위의 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지를 판단하는 단계; 및

상기 판단 결과 상기 클래스 파일에 논리적인 단위가 포함된 경우,판단 상기 논리적 단위의 정보를 컴파일하는 단계를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 10.

제 9항에 있어서,

상기 소정의 저장영역은 네트워크로 연결된 원격 시스템, 디스크, 외부 연결 단자(USB, IEEE1394, RF, Infrared, Bluetooth 등)로 연결된 외부 저장장치 또는 정보 기기 중 어느 하나인 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 11.

제 9항에 있어서,

상기 논리적인 단위의 정보는 클래스 헤더, 컨스턴트 풀, 인터페이스, 필드 및 메소드 중 어느 하나인 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 12.

제 9항에 있어서,

상기 클래스 로더 및 로드 타임 컴파일러는 각각 별도의 쓰레드를 이용하여 동시에 동작을 수행하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 13.

제 9항에 있어서,

상기 수신된 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지를 판단하는 단계는,

상기 수신된 클래스 파일에 논리적인 단위의 정보를 포함하고 있지 않은 경우, 다음 물리적인 단위가 전송되는 것을 대기하는 단계를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 14.

제 9항에 있어서,

상기 논리적인 단위의 정보를 컴파일하는 과정에서 클래스 파일이 올바르게 구성되었는지를 검증(verification)하는 단계를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 15.

제 9항에 있어서,

상기 논리적인 단위의 정보를 컴파일하는 과정에서 클래스 변수들의 초기값 설정을 하는 준비(preparation) 단계를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

청구항 16.

제 9항에 있어서,

상기 논리적인 단위의 정보를 컴파일하는 과정에서 다른 클래스의 메소드나 필드 등에 대한 참조 값들이 있을 경우 이들을 분해(resolution)하는 단계를 포함하는 것을 특징으로 하는 자바 프로그램의 처리 속도를 향상시키는 방법.

명세서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 자바 프로그램의 처리 속도를 향상시키는 시스템 및 그 방법에 관한 것으로서, 더욱 상세하게는 자바 가상 머신(Java Virtual Machine)에서 클래스 파일의 로딩 과정 및 클래스 파일의 컴파일 과정을 동시에 수행할 수 있는 자바 프로그램의 처리 속도를 향상시키는 시스템 및 그 방법을 제공하는 것이다.

도 1은 종래의 자바 가상 머신이 탑재된 정보 기기의 대표적인 구성예를 나타낸 도면으로서, 여기서 정보 기기는 휴대용 전화기기, PDA, Digital TV, set-top box, PC 등으로 이해될 수 있다.

상기 정보 기기는 CPU, 네트워크 인터페이스(Network Interface), 하드디스크나 Flash memory 등의 영구저장 장치(Persistent Storage), 주 메모리(memory), 입력/출력(input/output) 장치, 디스플레이(display) 장치, 및 기타 장치를 위한 외부장치 연결(External Device Connector)을 포함한다.

자바 가상 머신은 이러한 기기의 주 메모리에 상주하게 되며, 자바 애플리케이션을 구성하는 자바 클래스들(Java Classes)을 메모리에 로딩하여 수행하게 된다. 여기서, 영구저장 장치는 하드 디스크나 플래쉬 메모리(flash memory)로 이해될 수 있으며, 입력 장치는 PC의 경우 키보드나 마우스 등의 입력 장치 및 Digital TV나 Set-top box의 경우 리모콘 리시버로 이해될 수 있다. 출력/디스플레이 장치는 PC 의 경우 그래픽 어뎁터(graphic adapter)가 될 수 있고, Digital TV

의 경우 디스플레이 장치가 내장된 형태가 될 수도 있으며, 외장 형태로 연결될 수도 있다. 외부장치 연결은 통상적으로 활용되고 있는 USB, IEEE1394, RF, Infrared, Bluetooth 등의 연결 장치로 이해될 수 있다. 이와 같이 가상 머신이 구동될 수 있는 환경은 모든 정보기기를 포괄하고 있으며 특정한 장치에 국한되지는 않는다.

또한, 자바 클래스들은 네트워크 인터페이스를 통하여 외부로부터 전송받을 수도 있으며, 영구저장 장치로부터 전송되거나 외부장치 연결로 연결된 외부 장치로부터 전송될 수도 있다. 네트워크 인터페이스는 특정 네트워크 연결(Network Connectivity)을 통하여 외부에 존재하는 서버(Server)로 접속과 자바 클래스의 전송을 담당한다. 여기서, 네트워크 연결의 형태는 LAN, Wide Area Network(WAN) 등의 유선 패킷통신망이나, Wireless LAN, CDMA/GSM/GPRS 등의 무선/이동통신망, Digital TV 공중파/Cable/Satellite 등의 유/무선 방송망 등으로 특정 형태에 국한되지 않으며 여러가지 다양한 형태가 될 수 있다.

도 2는 종래의 자바 가상 머신의 구성을 나타낸 도면으로서, 자바 가상머신은 클래스 로더(class loader)(10), 인터프리터(interpreter)(20), 컴파일러(compiler)(30) 및 런타임 시스템(Runtime system)(40)을 기본적으로 포함한다.

클래스 로더(10)는 네트워크 또는 파일 시스템으로부터 자바 클래스 파일을 읽어 들여 클래스 파일에 포함된 내부 정보에 따라 적절히 메모리 상에 배치하고 내부 상태를 초기화한다

인터프리터(20)는 자바 클래스 파일의 수행 코드를 실행하기 위한 구성요소로써, 자바 바이트코드(bytecode) 형태의 명령어들을 인식하여 명령을 수행하는 역할을 한다. 즉, 클래스 로더(10)가 읽어들인 자바 클래스 파일에 포함된 메소드(method)의 바이트 코드를 인식하여 수행하는 것이다.

컴파일러(30)는 자바 클래스 파일의 수행중에 자바 바이트 코드를 정보기기에 탑재된 CPU의 기계어로 변환해주는 것으로서, 일반적으로 JIT 컴파일러(Just-In-Time compiler)가 널리 사용된다. 즉, 클래스 로더(10)가 읽어들인 자바 클래스 파일에 포함된 메소드의 바이트 코드를 CPU의 기계어로 변환하는 것이다.

런타임 시스템(40)은 상기 자바 가상 머신을 구성하는 요소들(클래스 로더, 인터프리터, 컴파일러)을 결합 및 운영하는 것이다.

도 3은 종래의 자바 클래스 파일의 구성을 나타낸 도면이다.

자바 클래스들은 일반적으로 클래스 파일이라는 특정 파일 포맷으로 전송되거나 저장되며, 상기 클래스 파일 포멧에는 클래스에 관련된 여러가지 속성들을 포함한다. 여기서, 클래스 파일 포맷은 기본 속성들을 표시하는 클래스 헤더(class header), 모든 상수들을 관리하는 컨스턴트 풀(constant pool), 클래스 파일의 인터페이스에 대한 정보를 포함하는 인터페이스 부분(interfaces)과 필드들에 대한 정보를 포함하는 필드 부분(fields) 및 클래스 파일에 정의된 각 메소드(method)별로 수행코드가 바이트코드 형태로 저장된 메소드 부분(methods)으로 구성된다.

도 4는 종래의 자바 가상 머신에서 자바 클래스 파일을 전송하고 수행하는 과정을 나타낸 순서도로서, 자바 클래스 파일이 전송되는 과정은 크게 클래스 로딩 단계, 초기화 단계 및 수행 단계로 구분될 수 있다.

먼저, 클래스 로딩 단계에서는 클래스 파일을 네트워크나 디스크로부터 메모리로 전송하는 단계와 링킹 단계로 이루어지며, 상기 링킹 단계는 다시 클래스에 대한 검증(verification) 단계, 클래스 준비 단계(preparation), 그리고 분해(resolution) 단계로 이루어진다.

검증 단계는 클래스 파일의 구조와 내용이 올바르게 작성되었는지를 판별하는 단계로서, 클래스 메소드들에 포함된 바이트코드들도 모두 유효한지 검사하게 된다. 준비 단계는 클래스에 정의된 상수나 변수의 디폴트 값을 지정하는 단계이며, 마지막으로 분해 단계는 컨스턴트 풀(constant pool)에 명시된 심볼(symbol)들을 수행중에 필요한 리소스의 물리적인 위치 정보로 치환하는 단계이다.

그 다음, 클래스 로딩 단계가 끝나면 메모리에 로드된 클래스 파일을 초기화한 후에 수행단계를 실행하는데, 수행 단계에서는 클래스에 포함된 수행 코드의 형태에 따라서 수행 모드가 결정된다.

여기서, 메소드가 바이트코드 형식으로 되어 있으면, 바이트코드를 CPU의 기계어로 변환할 것인지를 판별하고, 만약 기계어로 변환하지 않을 경우 인터프리터(interpreter)가 직접 바이트코드들을 하나하나 읽으면서 수행하게 된다. 바이트코드를 기계어로 변환할 경우에는 컴파일러(compiler)가 바이트코드들을 읽어서 CPU의 기계어로 변환한 후에 직접 변환된 기계어가 수행된다.

그러나, 종래의 기술에서는 클래스 파일의 전송과 링킹이 순차적으로 일어나며, 클래스 파일이 끝까지 전송될 때까지 링킹 단계를 수행하지 않는다. 이러한 특징으로 인해서, 클래스 파일을 전송한 후에 링킹하고 초기화해서 수행하는데까지 많은 시간의 지연이 일어나게 된다. 즉, 클래스 파일의 전송이 네트워크를 통한 전송일 경우에 네트워크의 대역폭이나 클래스 파일의 크기에 따라서 시간 지연이 일어나게 되고, 링킹 단계에서도 클래스 파일을 검증하는 단계에서 CPU를 많이 소모하게 된다.

뿐만 아니라, 수행시간에 메소드를 컴파일하는 과정에서 추가적으로 시간이 소모되는데, 특히 고도의 컴파일 기술을 사용하면 할수록 더 많은 시간이 컴파일하는 과정에서 소모되게 된다. 결과적으로 애플리케이션의 수행시간은 이러한 일련의 과정들로 인해 전체적인 성능이 저하되는 문제점이 있다.

한국공개특허 2000-0057010(바이트코드 실행 프로세스, 방법 및 데이터 처리 시스템과 그를 구현한 컴퓨터 프로그램 제품)은 클래스 로더를 통해 실행을 위한 바이트코드를 로딩하고, 바이트코드의 컴파일 여부를 판정한 후 JIT 컴파일러로 전송하고, 바이트코드가 JIT 컴파일 되어야 한다는 판정이 아닐 경우 인터프리터로 전송하는 바이트코드 실행 프로세스를 개시하고 있으나, 바이트 코드를 로드하는 과정인 전송 단계와 링킹단계, 그리고 컴파일 단계를 동시에 병행하는 방법에 대해서는 전혀 언급하고 있지 않다.

발명이 이루고자 하는 기술적 과제

본 발명은 상기한 문제점을 해결하기 위하여 안출된 것으로서, 본 발명의 목적은 자바 가상 머신에서 클래스 파일의 로딩 과정 및 클래스 파일의 컴파일 과정을 동시에 수행함으로써, 애플리케이션의 전체 수행 성능을 향상시킬 수 있는 자바 프로그램의 처리 속도를 향상시키는 시스템 및 방법을 제공하는 것이다.

발명의 구성

상기 목적을 달성하기 위하여 본 발명의 자바 프로그램의 처리 속도를 향상시키는 시스템은, 자바 프로그램의 클래스 파일을 로딩하는 클래스 로더 및 상기 클래스 로더가 로딩중인 클래스 파일을 링킹 및 컴파일하는 로드 타임 컴파일러를 포함하는 것을 특징으로 한다.

또한, 본 발명의 자바 프로그램의 처리 속도를 향상시키는 방법은 클래스 로더가 소정의 저장영역에 저장된 클래스 파일을 소정 크기의 물리적인 단위로 나누어서 메모리로 전송(또는 로딩)하는 단계와, 상기 클래스 로더가 로딩한 물리적인 단위의 클래스 파일을 로드 타임 컴파일러가 링킹 및 컴파일하는 단계와, 상기 로드 타임 컴파일러가 링킹 및 컴파일하는 단계에서 상기 수신된 물리적인 단위의 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지를 판단하는 단계 및 상기 포함된 논리적인 단위의 정보를 링킹하고 상기 정보가 자바 메소드(method)일 경우 상기 메소드를 컴파일하는 단계를 포함하는 것을 특징으로 한다.

이하, 본 발명의 바람직한 실시예를 첨부된 도면을 참조하여 상세히 설명한다.

도 5는 본 발명의 자바 프로그램의 처리 속도를 향상시키는 시스템을 개략적으로 나타낸 도면으로서, 클래스 로더(110), 로드 타임 컴파일러(loader time compiler)(120) 및 런타임 시스템(130)을 포함하여 구성된다.

클래스 로더(110)는 자바 프로그램의 클래스 파일을 메모리에 로딩하는 것으로, 즉 소정의 저장영역으로부터 전송되는 소정 크기의 물리적인 단위로 클래스 파일을 로딩한다. 또한, 클래스 로더(110)는 물리적인 단위의 클래스 파일이 현재까지 전송된 전송량을 표시해 둠으로써, 상기 로드 타임 컴파일러(120)가 클래스 파일의 링킹 및 컴파일을 수행하는데 참조할 수 있도록 한다. 여기서, 소정의 저장영역은 네트워크로 연결된 원격 시스템 또는 디스크 등으로 이해될 수 있으며, 상기 클래스 파일은 소정 크기의 단위(예를 들어, 네트워크일 경우 패킷, 디스크일 경우 블록 등)로 분할되어 전송된다.

로드 타임 컴파일러(120)는 클래스 로더(110)가 로딩한 클래스 파일을 링킹 및 컴파일 하는 것으로, 즉, 상기 클래스 로더(110)로 로딩중인 물리적인 단위의 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지를 판단하고, 상기 논리적인

단위의 정보를 링킹하고 상기 정보가 메소드일 경우 상기 메소드에 저장된 바이트 코드에 대한 컴파일을 수행한다. 여기서, 로드 타임 컴파일러(120)는 상기 판단을 상기 클래스 로더(110)가 표시한 현재까지 전송된 클래스 파일의 전송량을 참조하여 클래스 파일의 링킹 및 컴파일을 수행한다. 즉, 로드 타임 컴파일러(120)는 클래스 로더(110)가 네트워크나 디스크 등의 저장영역으로부터 전송된 클래스 파일을 메모리로 로드한 후, 현재까지 전송된 클래스 파일의 전송량을 표시하면, 상기 표시된 클래스 파일의 전송량을 확인 한 후 전송된 분량만큼에 포함된 논리적인 단위의 정보를 링킹하고 컴파일 함으로써 클래스 파일의 전송이 완료되기 전에 미리 클래스 파일의 컴파일을 수행한다. 한편, 상기 논리적인 단위의 정보는 클래스 파일을 구성하는 클래스 헤더, 컨스턴트 풀, 인터페이스, 필드 및 메소드 영역으로 이해될 수 있다.

또한, 로드 타임 컴파일러(120)는 링킹 및 컴파일 수행 단계에서 링킹(linking) 과정에서 필요한 검증(verification), 준비(preparation) 및 분해(resolution) 과정을 함께 수행한다.

런타임 시스템(130)은 클래스 파일을 수행하기 위해 필요한 자원을 관리한다.

한편, 클래스 로더(110) 및 로드 타임 컴파일러(120)는 각각 별도의 쓰레드(thread) 로 동시에 동작을 수행한다.

한편, 종래의 가상 머신에 있었던 인터프리터(20)와 컴파일러(30)는 본 발명에서의 구현 방법에 따라서 옵션 사항으로 포함시킬 수도 있고 제외할 수도 있다.

즉, 기본적으로 모든 클래스 파일을 로드시간에 컴파일을 하는 방식을 따른다면 인터프리터(20)와 컴파일러(30)는 필요가 없으나, 구현 방법에 따라서 특정 클래스 파일이나 상기 클래스 파일의 특정 메소드는 컴파일을 수행하지 않는 방법을 사용할 경우에는 인터프리터(20)와 컴파일러(30)가 수행 시간에 바이트코드를 직접 수행하거나 다시 컴파일을 수행해야 한다.

또한, 본 발명의 실시예는 자바 가상 머신(100)을 예로 설명하고 있지만 본 발명은 Microsoft의 Common Language Runtime(CLR)이나 Smalltalk의 가상 머신과 같이 동등한 기능을 제공하는 다른 가상 머신에서도 적용이 가능하다.

도 6은 본 발명의 자바 프로그램의 처리 속도를 향상시키는 방법을 개략적으로 나타낸 순서도로서, 먼저 클래스 로더(110)가 소정의 저장영역에서 소정 크기의 물리적인 단위로 클래스 파일을 메모리로 전송(또는 로딩)하기 시작한다 (S100).

여기서, 클래스 로더(110)는 클래스 파일의 현재까지 전송된 물리적인 단위의 전송량을 표시해 둠으로써, 상기 로드 타임 컴파일러(120)가 클래스 파일의 컴파일을 수행하는데 참조할 수 있도록 한다.

그 다음, 로드 타임 컴파일러(120)는 클래스 로더(110)가 표시한 현재까지 전송된 클래스 파일의 전송량을 확인한 후, 메모리에 로딩된 물리적인 단위의 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지 판단한다(S110). 여기서, 논리적인 단위의 정보는 클래스 파일을 구성하는 클래스 헤더, 컨스턴트 풀, 인터페이스, 필드 및 메소드 영역으로 이해될 수 있다.

그 다음, 상기 판단 결과 현재까지 전송된 클래스 파일에 다음으로 처리해야 할 논리적인 단위의 정보가 모두 포함된 경우 해당 논리적인 단위의 정보를 링킹 및 컴파일 하고(S120), 판단 결과 클래스 파일에 다음으로 처리해야 할 논리적인 단위의 정보가 포함되지 않은 경우, 로드 타임 컴파일러(120)는 클래스 로더(110)가 다음 물리적인 단위를 메모리에 로딩할 때까지 쓰레드를 중지시킴(suspend)으로써 대기한다(S130).

그 다음, 클래스 로더(110)는 다음 물리적인 단위가 메모리에 로딩되면, 클래스 파일의 전송량의 증가 여부를 표시하고 중지된 로드 타임 컴파일러(120) 쓰레드를 재개(wakeup)시킨다(S140). 여기서, 상기 단계 S110, 단계 S120, 단계 S130 및 단계 S140 과정은 로드 타임 컴파일러(120)가 클래스 파일에 포함된 논리적인 단위의 정보를 모두 처리할 때까지 계속 수행된다(S150).

한편, 클래스 로더(110) 및 로드 타임 컴파일러(120)는 각각 별도의 쓰레드로 동시에 동작을 수행한다.

이하, 도 7에서 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지 판단하는 과정 및 클래스 파일을 링킹 및 컴파일 하는 과정을 자세히 설명한다.

도 7은 본 발명에 따른 로드 타임 컴파일러의 링킹 및 컴파일 동작 과정을 나타낸 순서도로서, 상기 도 6에서 클래스 파일에 논리적인 단위의 정보가 포함되어 있는지 판단하는 과정 및 클래스 파일을 링킹 및 컴파일하는 과정을 상세히 설명하는 것이다.

먼저, 로드 타임 컴파일러(120)는 클래스 로더(110)가 메모리에 로드중인 클래스 파일에 다음 처리해야 할 논리적인 단위인 클래스 헤더 정보가 포함되어 있는지를 판단한다(S200).

판단 결과 클래스 파일에 클래스 헤더 영역의 정보가 모두 포함되어 있으면, 해당 클래스 헤더의 링킹(즉, 검증) 과정을 수행하고(S210), 판단 결과 클래스 파일에 클래스 헤더 영역의 정보가 모두 포함되어 있지 않으면, 클래스 로더(110)가 다음 물리적인 단위를 로딩할 때까지 대기한다(S211).

그 다음, 클래스 로더(110)가 로딩한 클래스 파일에 다음으로 처리해야 할 컨스턴트 풀 영역의 정보, 인터페이스 영역의 정보, 필드 영역의 정보 및 메소드 영역의 정보가 모두 포함되어 있는지를 판단하는 과정과, 그에 따른 링킹 과정 및 다음 클래스 파일이 로딩되는 과정(S220 내지 S291)은 상기 단계 S200 내지 단계 S211 과정과 동일하게 수행된다. 단, 메소드 영역 정보를 처리하는 과정에서는 메소드에 저장된 바이트 코드를 컴파일하는 과정을 추가로 수행한다.

이상에서 본 발명에 대하여 상세히 기술하였지만, 본 발명이 속하는 기술 분야에 있어서 통상의 지식을 가진 사람이라면, 첨부된 청구범위에 정의된 본 발명의 정신 및 범위를 벗어나지 않으면서 본 발명을 여러 가지로 변형 또는 변경하여 실시할 수 있음은 자명하며, 따라서 본 발명의 실시예에 따른 단순한 변경은 본 발명의 기술을 벗어날 수 없을 것이다.

발명의 효과

상기한 바와 같이 이루어진 본 발명에 따르면, 자바 가상 머신에서 클래스 파일의 로딩 과정 및 클래스 파일의 컴파일 과정을 동시에 수행하여, 클래스 파일 로딩 시간내에 컴파일 과정이 수행되도록 함으로써, 애플리케이션의 전체 수행 성능을 향상시킬 수 있는 효과가 있다.

또한, 클래스 파일이 로딩된 이후에 컴파일 단계에 의한 시간 지연을 최소화할 수 있는 동시에 클래스 파일의 메소드 수행이 직접 CPU의 기계어로 변환된 형태로 수행되게 함으로써, 클래스 파일의 전송 및 로딩 후의 컴파일 시간과 클래스의 수행시간을 모두 단축시킬 수 있으며, 이는 종래의 컴파일을 수행하는 방식보다 클래스 파일의 크기를 최소화할 수 있어서 클래스 파일의 전송시간도 줄일 수 있는 효과가 있다.

또한, 본 발명은 클래스 파일의 표준 포맷을 그대로 활용할 수 있어서 가상 머신의 장점인 이식성을 최대한 활용할 수 있다.

도면의 간단한 설명

도 1은 종래의 자바 가상 머신이 탑재된 정보 기기의 대표적인 구성예를 나타낸 도면.

도 2는 종래의 자바 가상 머신의 구성을 나타낸 도면.

도 3은 종래의 자바 클래스 파일의 구성을 나타낸 도면.

도 4는 종래의 자바 가상 머신에서 자바 클래스 파일을 전송하고 수행하는 과정을 나타낸 순서도.

도 5는 본 발명의 자바 프로그램의 처리 속도를 향상시키는 시스템을 개략적으로 나타낸 도면.

도 6은 본 발명의 자바 프로그램의 처리 속도를 향상시키는 방법을 개략적으로 나타낸 순서도.

도 7은 본 발명에 따른 로드 타임 컴파일러의 컴파일 동작 과정을 나타낸 순서도.

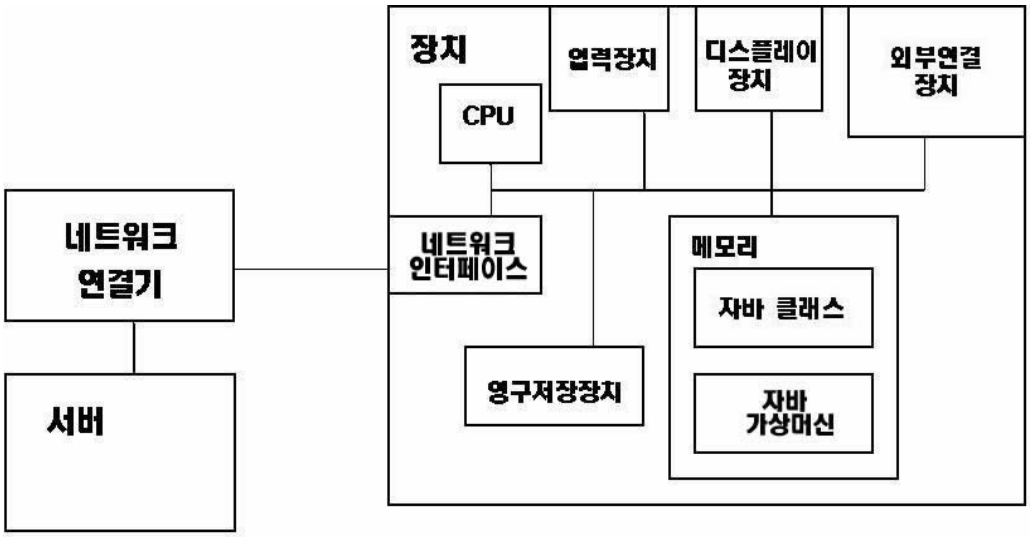
<도면의 주요 부분에 관한 부호의 설명>

100 : 자바 가상 머신 110 : 클래스 로더

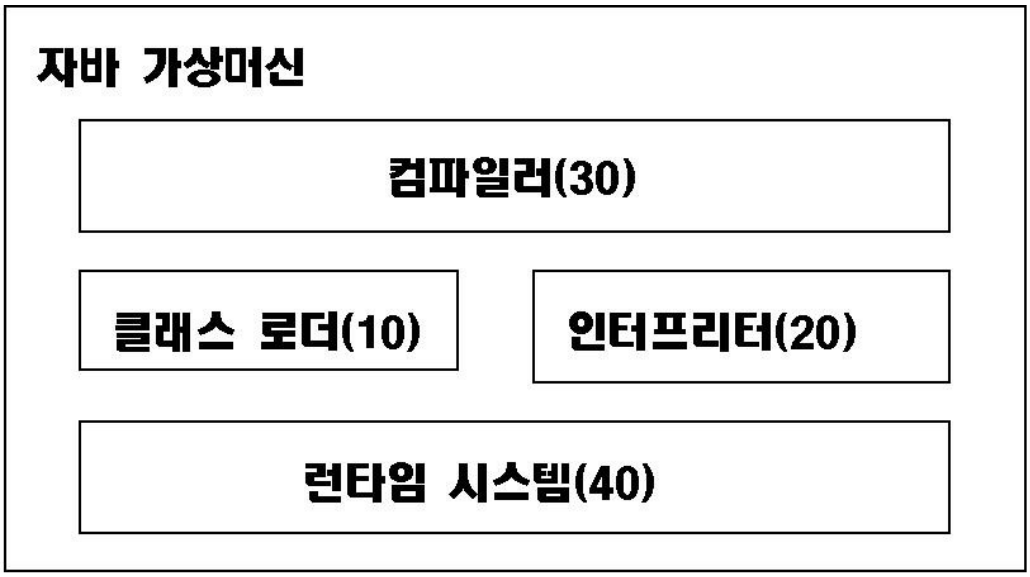
120 : 로드 타임 컴파일러 130 : 런타임 시스템

도면

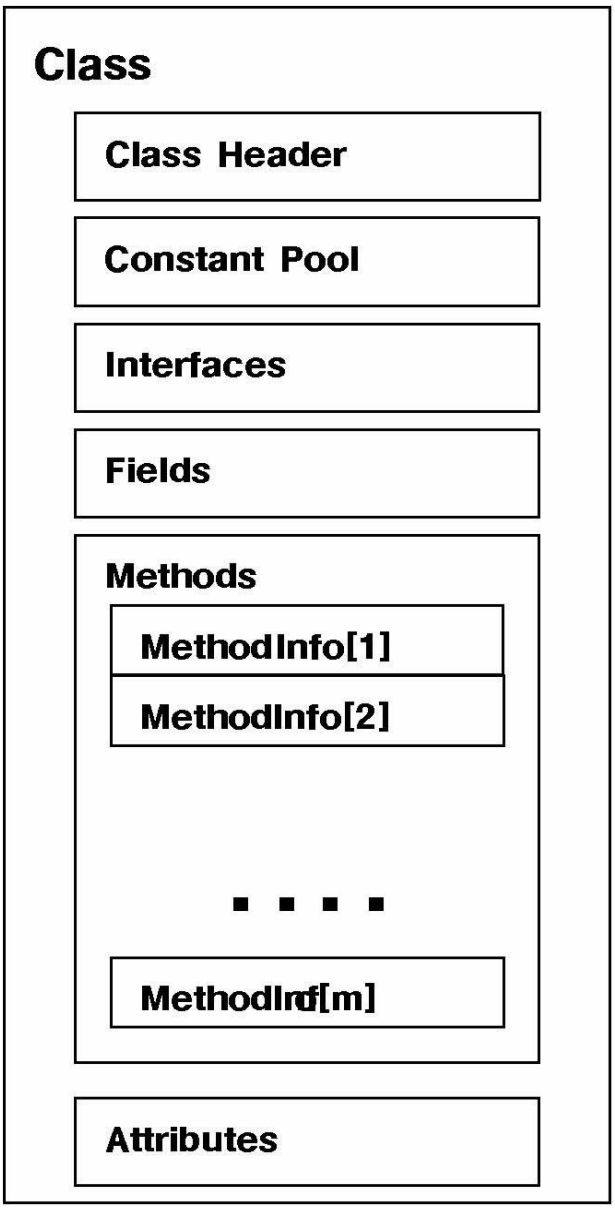
도면1



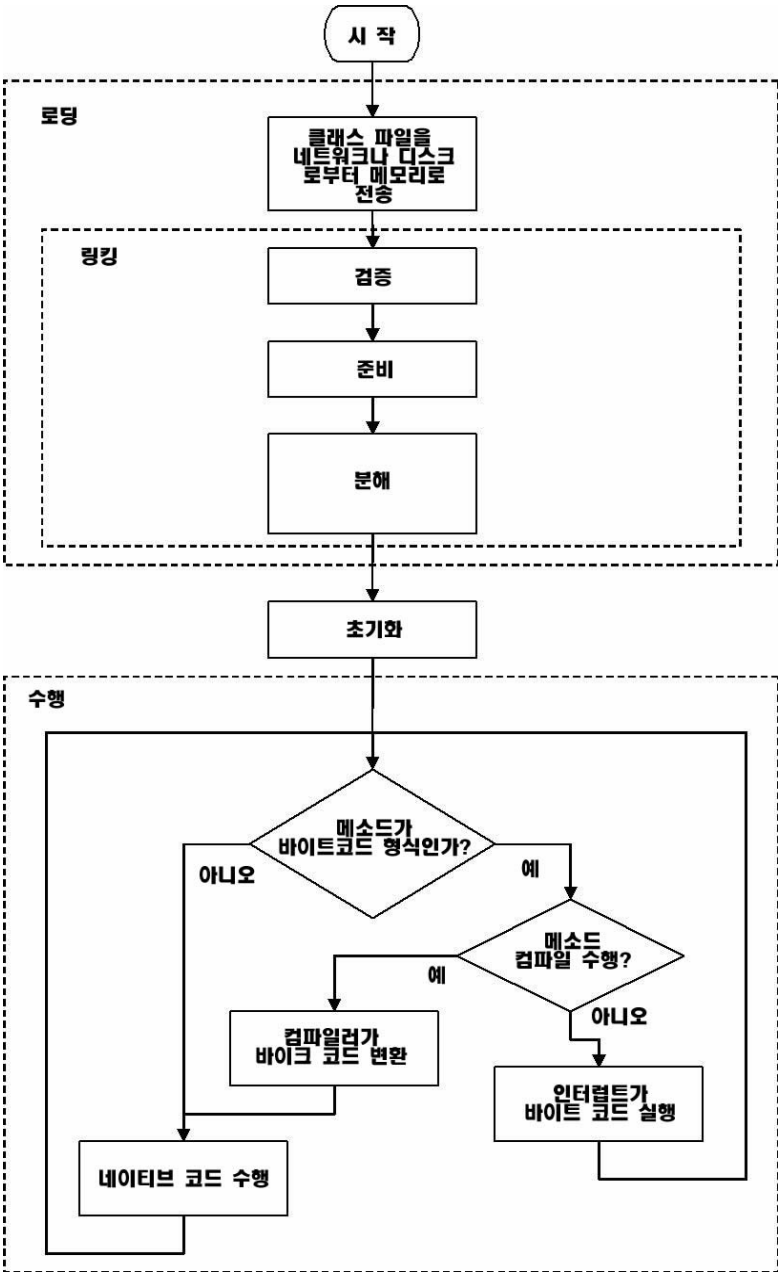
도면2



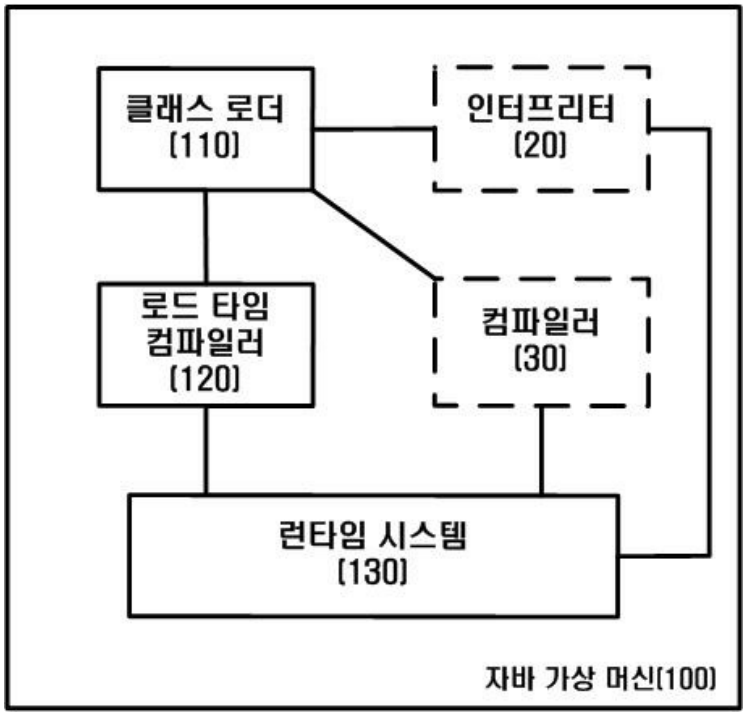
도면3


Class
Class Header
Constant Pool
Interfaces
Fields
Methods
MethodInfo[1]
MethodInfo[2]
MethodInfo[m]
Attributes

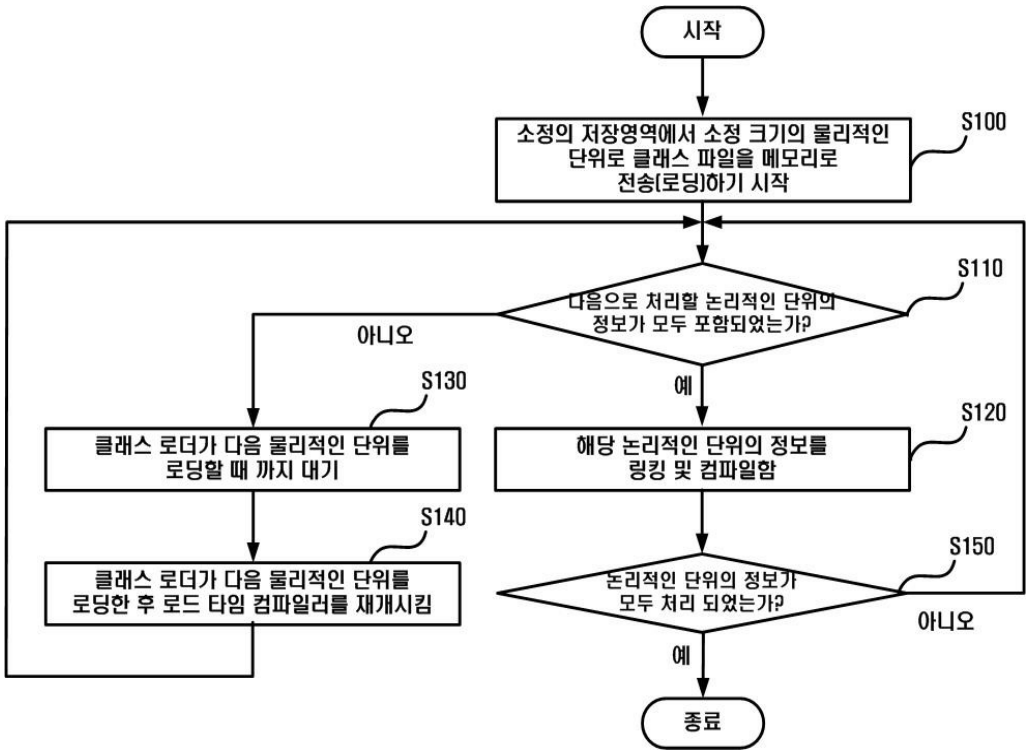
도면4


시 작
로딩
클래스 파일을
네트워크나 디스크
로부터 메모리로
전송
링킹
검증
준비
분해
초기화
수행
메소드가
바이트코드 형식인가?
아니오
예
메소드
컴파일 수행?
예
아니오
컴파일러가
바이크 코드 변환
인터럽트가
바이트 코드 실행
네이티브 코드 수행

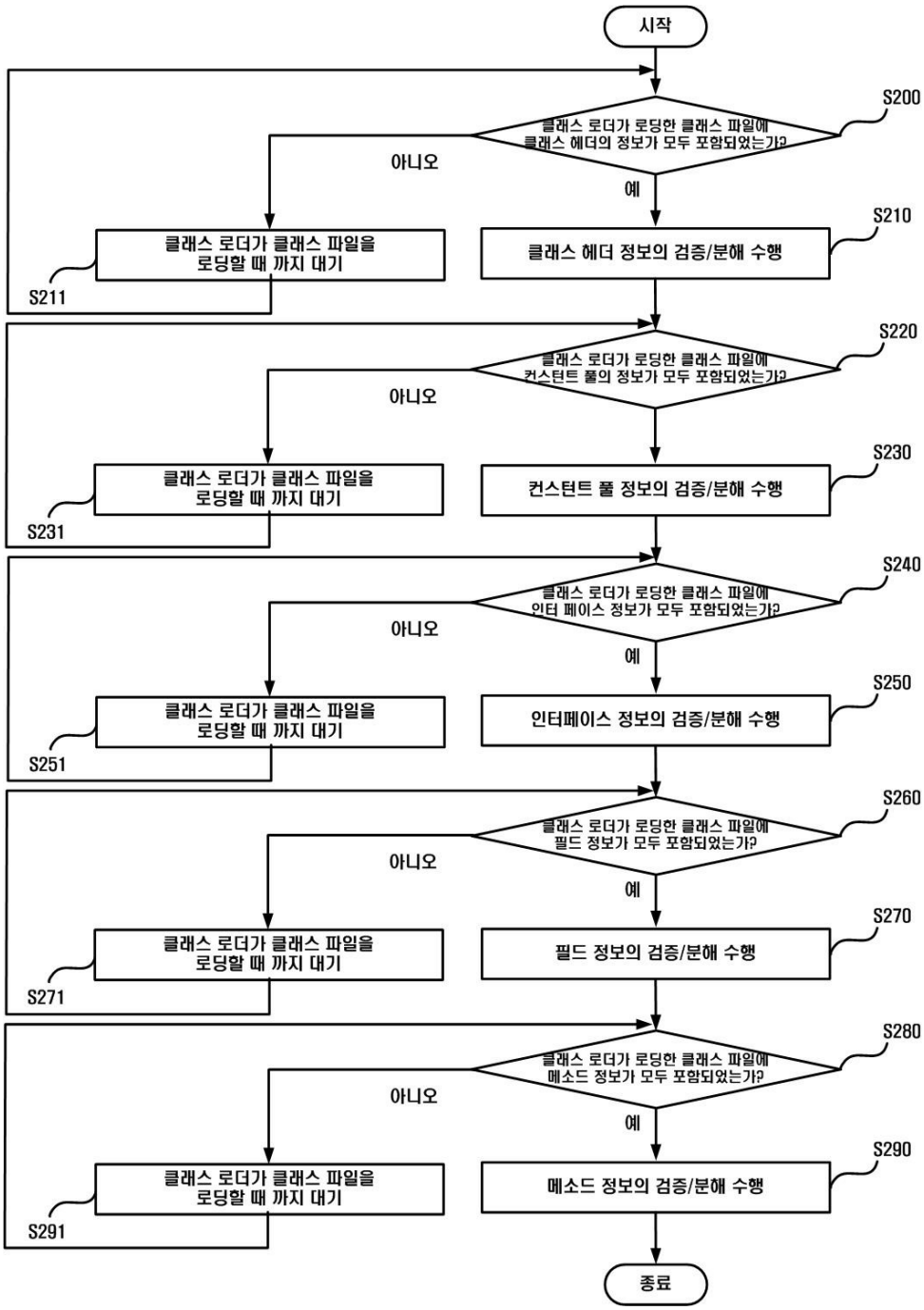
도면5


클래스 로더
(110)
인터프리터
(20)
로드 타임
컴파일러
(120)
컴파일러
(30)
런타임 시스템
(130)
자바 가상 머신(100)

도면6


시작
소정의 저장영역에서 소정 크기의 물리적인 단위로 클래스 파일을 메모리로 전송(로딩)하기 시작
S100
다음으로 처리할 논리적인 단위의 정보가 모두 포함되었는가?
S110
아니오
예
S130
클래스 로더가 다음 물리적인 단위를 로딩할 때 까지 대기
S120
해당 논리적인 단위의 정보를 링킹 및 컴파일함
S140
클래스 로더가 다음 물리적인 단위를 로딩한 후 로드 타임 컴파일러를 재개시킴
S150
논리적인 단위의 정보가 모두 처리 되었는가?
아니오
예
종료

도면7


시작
S200
클래스 로더가 로딩한 클래스 파일에 클래스 헤더의 정보가 모두 포함되었는가?
아니오
예
S210
클래스 헤더 정보의 검증/분해 수행
S211
클래스 로더가 클래스 파일을 로딩할 때 까지 대기
S220
클래스 로더가 로딩한 클래스 파일에 컨스턴트 풀의 정보가 모두 포함되었는가?
아니오
예
S230
컨스턴트 풀 정보의 검증/분해 수행
S231
클래스 로더가 클래스 파일을 로딩할 때 까지 대기
S240
클래스 로더가 로딩한 클래스 파일에 인터 페이스 정보가 모두 포함되었는가?
아니오
예
S250
인터페이스 정보의 검증/분해 수행
S251
클래스 로더가 클래스 파일을 로딩할 때 까지 대기
S260
클래스 로더가 로딩한 클래스 파일에 필드 정보가 모두 포함되었는가?
아니오
예
S270
필드 정보의 검증/분해 수행
S271
클래스 로더가 클래스 파일을 로딩할 때 까지 대기
S280
클래스 로더가 로딩한 클래스 파일에 메소드 정보가 모두 포함되었는가?
아니오
예
S290
메소드 정보의 검증/분해 수행
S291
클래스 로더가 클래스 파일을 로딩할 때 까지 대기
종료