자바 공부내용 정리

2021년 8월 3일 화요일 오후 1:14

1. Java Architecture

Java가 뭐야 ?

Write Once, Run Everywhere 라는 하나의 철학을 지칭하는 개념
즉 한곳에서 개발한 프로그램을 어디에서나 수행하기위한 프로그램

자바는 총 4가지 상호 연관된 기술을 묶어 놓았다.

- The Java Programming Language
- The Java Class File format
- The Java Application Programming interface(JAVA API)
- The Java Virtual Machine(JVM)

Java Programming Language로 프로그램을 작성하고
작성한 프로그램을 Compile하면 Java Class File Format으로 변경
이를 수행하기 위해서는 Jaa Virtual Machine(JVM)을 구동시켜 Class파일로 로딩
JVM으로 로딩된 프로그램은 단독으로 수행하는 것이 아니라 Java Application Programming Interface와 동적으로 연결되어서 실행된다.

구체적으로
Java Programming Language는
객체지향, Multi-Threading, 구조화된 에러 핸들링, Garbage Collection, Dynamic Linking, Dynamic Extension등의기술이 접목 되어있다. 해당하는 기술들은 다른 언어에서도 특성을 가지고 있지만
이 중에서 Dynamic Linking은 특별하다.
Java의 Class 파일은 엄밀히 말하면, 실행 가능한 형태로 변경된 것이 아닌 JVM 이 읽을 수 있는형태의 언어로 번역된 것으로 이해할 수 있다. 이것은 JVM 위에서 다시 실행 가능한 형태로 변형 된다. 실행을 위한 Linking 작업은 그때 일어나게 된다. Class 파일은 실행시 Link 를 할 수 있도록 Symbolic Reference 만을 가지고 있다. 이 Symbolic Reference는 Runtime 시점에서 메모리상에서 실제로 존재하는 물리적인 주소로 대체되는 작업인 Lingking이 일어나게 되는 것이다. 이러한 Link 작업은 필요할 때마다 동적으로 이루어지기 Eans에 이를 가르켜 Dynamic Linking 이라고 한다

이 Dynamic Linking 이라는 기술 덕분에 Class 파일의 크기를 작게 유지할 수 있다.

또한 Compile된 파일만 있으면 그대로 수행이 가능해서 플랫폼에 독립적이다.
이것은 Network를 통해 객체를 전송하고 배포하는데 있어 파일의 크기는 작을수록 유리하다.

하지만 무엇보다 프로그램 언어로서 Java가 지니고 있는 특징은 생산성을 극대화 한다는 데 있다.
Java는 Runtime Memory 를 직접 핸들링 하지 않는다. Garbage Collection과 같은 기술을 채용했고 이는 개발 및 운영에 소요되는 시간과 노력을 많이 단축시켜준다

그 밖에도 객체지향의 특성인 Source Code의 재사용, Array Bound Check 등과 같은 엄격한 Type Rule, 그리고 Object Reference Check 등을 통한 프로그램의 안정성 제고 등도 Java의 장점이다

The Jaa Class File Format

Java는 개발자가 작성한 프로그램을 Compiler를 통해 Class 파일로 재생성 되는 과정을 거치게 된다.
이렇게 생선된 Class 파일은 다음의 네 가지 특징을 가지게 된다.

- Compact 한 형태
- Bytecode 로의 변경
- Platform 독립적
- Network Byte Order의 사용

Jaa 의 철학을 가장 두드러지게 대변하는 것이 바로 이 Class File Format 이다. Class파일은 Bytecode를 Binary 형태로 담아놓은 것이라 정의할 수 있다. Bytecode 는 JVM이 읽을 수 있는 언어를 의미한다. 다시 말해 Java에서 Compile 이라고 하는 것은 Text 기반의 사용자 친화적인 Java 언어를 JVM이 읽을 수 있는 언어로 번역한 것에 지나지 않는다. JVM은 Class를 로딩한 후 여기서 Bytecode 을 읽어 들여 실행이 가능하도록 해석(Interpret)하는 과정을 거치게 된다. Class 가 Load 된 후에는 JIT Compiler나 Hotspot Compiler와 같은 Execution Engine을 거쳐 실행된다 (이 부분은 추후에 5장에서 설명)

Bytecode 는 JVM을 위한 언어이고 모든 Code가 Bytecode 의 Binary 형태로 실체화된 Class 라는 것으로 배포된다는 점은 Platform의 제약을 뛰어 넘을 수 있게 되었다는 것을 의미하기 때문이다 JVM이 Bytecode를 읽을 수 있는 한, JVM이 어디에 설치되어 있는지는 고려 대상에서 제외된다. Unix 설치되어있건 Windows 설치되어있건 모두 동일하게 수행될 것이기 때문에 플랫폼에 독립적이라 모든 환경에서 실행이 가능하다

Bytecode는 Source Code를 단순히 JVM의 언어로 번역해 놓은 것이기 때문에 Source Code

와 비슷한 크기를 가지고 있다. Compiler를 통해 실행파일로 변경되는 과정에서 Library 들을 포함하는 C++, Delphi와 같은 언어에 비한다면 아주 작은 크기라 할 수 있다. 이렇게 작은 크기를 유지할 수 있는 이유는 Class 파일에는 실제로 참조하는 라이브러리를 포함하고 있지 않고 , 단지 Symbolic Reference 만을 가지고 있기 때문이다

이 Symbolic Reference는 참조하고자 하는 대상의 이름만으로 참조관계를 구상한 것을 의미한다. 참조하는 객체의 특정 메모리 번지로 참조관계를 구성한 것이 아닌 참조 대상의 이름만을 지칭하는 것이다. Symbolic Reference는 그 이름에 맞는 객체의 주소를 찾아서 연결하는 작업을 수행한다. 앞에서 우리는 이러한 작업을 Dynamic Linking 이라고 하였다. Java는 이 Dynamic Linking 덕분에 Class 파일을 Compact 한 형태를 유지할 수 있다.

Class File Format의 마지막 특징은 Network Byte Order를 사용한다. Byte Order는 메모리의 주소 값을 할당하는 방식을 의미하는데, ZEON이나 Itanium 과 같은 Intel 계열의 CPU는 Little Endian 이라는 Byte Order을 사용하는데, 이것은 메모리 주소의 번지수를 점점 작게 부여하는 것을 의미한다. 그러나 Unix 머신에서 주로 사용되는 SPARC, RISC 계열의 CPU는 Big Endian 방식을 사용한다. 이는 번지수를 점점 크게 부여하는 것을 의미한다. 이경우 두 CPU에 맞게 생성된 프로그램들은 서로 호환이 되지 않는 문제를 가진다
Network Byte Order는 서로 다른 계열의 CPU 끼리 데이터를 전송 받을 때의 문제점을 해결하기 위해 정해진 일종의 약속이다. 예를 들어 RISC 계열의 CPU를 사용하는 Unix 머신에서 Pentium CPU를 장착한 Windows로 데이터를 보내면 ByteOrder를 고려하지 않고 보낸다면 서로의 데이터는 주소가 반대로 되어있어 제대로 전송되지 않을 것이다. 그렇기 때문에 Network를 통해 데이터를 전송할 때는 통일된 방식을 따르기로 약속했는데 이것이 바로 Network Byte Order이다

Java Application Interface
Java API는 한마디로 Runtime Library의 집합이다.
앞서 Class 파일을 수행하기 위해서는 JRE가 필요하다고 하였다. 이 JRE는 말 그대로 Java 실행 환경이다. 여기에는 Java Virtual Machine 과 Java API, 그리고 Native Method 등이 포함되어 있다.

Java API 는 OS 시스템과 Java 프로그램 사이를 이어주는 가교의 역할을 한다.
Java API는 Native Method를 통해 OS 자원과 연계되어 있고 다른 한 편으로는 Java 프로그램과 맞닥드리고있다

Java 에서는 Class 파일 내에 있는 Symbolic Reference를 이용하여 Runtime 시 해당 Instance에 접근하게 된다. 그러면 이 Instance에 대한 내용, 즉 실제 File에 대한 접근을

Native Method를 통해 OS에 명령을 전달하게 된다. 이후 OS는 실제 File IO를 일으키게 되고, 이 File IO의 결과는 다시 Native Method를 통해 Java API로 전달되는 과정을 거쳐 프로그램이 실행되는 것이다.

Java Virtual Machine(JVM)

Java 의 4가지 구성요소 중 가장 핵심적인 것을 고르라고 하면 당연히 JVM이다.
그 이름에 자신의 모든 특성을 담고 있다. JVM은 먼저 JAVA를 위한 것이다. JAVA는 이것을 의미하고 있다.
그리고 JVM은 물리적인 형태가 아닌 Software로 그리고 무엇보다 하나의 개념으로 존재한다 이것이
VIRTUAL 이 뜻하는 것이다. 그리고 JVM은 독자적으로 작동할 수 있는 메커니즘과 구조를 가지고 있다. 이는 하나의 축약된 컴퓨터와 같다. 그렇기 때문에 Machine이라는 단어를 사용하게 된 것이다.

Runtime Data Ares

Runtime Data Ares는 Process로서의 JVM이 프로그램을 수행하기 위해 OS로부터 할당 받는 메모리 영역이라고 정으 ㅣ내릴 수 있다. JVM에서 Class 파일의 시랭은 Class Loader를 통해 로딩되면서 시작된다