JVM

It is basically a program that provides the runtime environment necessary for Java programs to execute.It is a Runtime Engine responsible to run java based applications

java Virtual Machine (JVM) is an abstract computer machine that is responsible for executing Java bytecode (a highly optimized set of instructions) on a particular hardware platform. It is also called Java run-time system.

it is abstract so it is not physicaly present

The specification of JVM is provided by Sun Microsystem whose implementation provides a runtime environment to execute our Java applications. JVM implementation is known as Java Runtime Environment (JRE).

Since JVM is platform-dependent, therefore, it is available for many hardware and software platforms.

The most popular JVM is HotSpot that is produced by Oracle. It is available for many operating systems such as Windows, Linux, Solaris, and Mac OS.

We cannot run the Java program unless JVM is available for the appropriate hardware and operating system platform.

Virtual Machine is a software simulation of a machine which can perform operations similar to physical machine.

For eg. Calculator software in operatign system, which is not physically present, but performs all the functions similar to physical calculator.

Types of Virtual Machine :

1. Hardware Based or System Based Virtual Machine

2. Application Based or Process Based Virtual Machine

JVM (Java Virtual Machine) is Application Based Virtual Machine

writes public class Test

Devloper -------------> {

---------- ----------------> javac ------------> java bytecode file ----------------->JVM----------->operating System

--------- is compiled By Generates Test.class is executed by on

-----------

}

Test.java

(javasource file)

Internal Architecture of Java Virtual Machine (JVM)

.Class file -----------> class loader sub system

^

|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_V\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| |

|Method Area Heap java Stack PC Register Native Method stack |

| |

| Runtime Data Areas |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

^ ^

| |

\_\_\_\_\_\_\_ V\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_V\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | | | | |

| Interpreter JIT Compiler |<-------->| Native Method Interface|<------------| Java Native Libraries |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

Execution Engine

^

|

\_\_\_\_\_\_\_\_\_\_\_V\_\_\_\_\_\_\_\_\_

| Operating System |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

JVM contains the following main components that are as follows:

Class loader sub system

Runtime data areas

Execution engine

Native method interface

Java native libraries

Operating system

Runtime data areas consist of the following sub-components that are as follows:

Method area

Heap

Java stacks

PC register

Native method stacks

How JVM works Internally?

Java Virtual Machine performs the following operations for execution of the program. They are as follows:

a) Load the code into memory.

b) Verifies the code.

c) Executes the code

d) Provides runtime environment.

When we make a program in Java, .java program code is converted into a .class file consisting of byte code instructions by the Java compiler.

This Java compiler is outside of JVM.

Now, Java Virtual Machine performs the following operations that are as follows:

1. This .class file is transferred to the class loader sub system of JVM

In JVM, class loader sub system is a module or program that performs the following functions:

a) First of all, the class loader sub system loads .class file into the memory.

b) Then bytecode verifier verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the further execution process is rejected immediately.

c) If byte code instructions are proper, it allots the necessary memory to execute the program.

This memory is divided into 5 separates parts that is called run-time data areas. It contains the data and results during the execution of the program.

These areas are as follows:

1. Class (Method) Area:(thread not safe)

Class (Method) area is a block of memory that stores the class code, code of variables, and methods of the Java program. Here methods mean functions declared in

the class. (fully qulified class name ,fully qulified parent class name,varibales information ,method information and constructor information,static variables)

Method area is created when JVM is started.

• It stores.class file information and static variables.

• Per JVM one memory area, therefore multiple threads can access this area, so it is not thread safe.

2 Heap:(thread not safe)

This is the runtime data area where objects are created by JVM. When JVM loads a class, a method area and a heap area are immediately built in it.

(also stored arrays in heap area beacuse array is also an object and instance variables)

• Heap area is created when JVM is started.

• It stores objects, instance variables and arrays (as every arrays is an object in java).

• It can be accessed by multiple threads, so the data stored in heap area is not thread safe.

3 Stacks:(thread safe)

Method code is stored in the Method area. But during the execution of a method, it requires some more memory to store the data and results.

This memory is allocated on Java stacks.

Java stacks are those memory areas where Java methods are executed. In Java stacks, a separate frame is created where the method is executed.

Each time a method is called, a new frame is created into the stack. When method invocation is completed, a frame associated with it is destroyed.

JVM always creates a separate thread (or process) to execute each method.

(stores local variables and current running method)

• Whenever a new thread is created, a separate stack area will also be created

• It stores the current running method and local variables.

• When the method is completed, the corresponding entry from the stack will be removed.

• After completing all method calls, the stack will become empty and that empty stack will be destroyed by thee JVM just before terminating the thread.

• The data stored in the stack is available only for the corresponding thread and not available to the remaining thread, so this area is thread safe

4. PC Register:(thread safe)

PC (Program Counter) registers are those registers (memory areas) that contain the memory address of JVM instructions currently being executed.

(it contains next executing instruction address)

It holds the address of next executing instruction.

• For every thread, a separate pc register is created, so it is also safe.

5.Native Method Stack:(threas safe)

Methods of Java program are executed on Java stacks. Similarly, native methods used in the program or application are executed on Native method stacks.

Generally, to execute the native methods, Java native method libraries are needed. These header files are located and connected to JVM by a program, known as Native method interface.

(all the native methods present in our programs that stores)

All native method calls invoked by the thread will be stored in corresponding native method stack.

• For every thread separate native method stack will be created

• It is also thread safe.

Execution Engine

Execution engine consists of 2 parts: Interpreter and JIT (Just In Time) compiler.

They convert the byte code instructions into machine code so that the processor can execute them.

In Java, JVM implementation uses both interpreter and JIT compiler simultaneously to convert byte code into machine code. This technique is called adaptive optimizer.

JIT Compiler in Java

JIT compiler in Java is the part of JVM that is used to increase the speed of execution of a Java program.

it is used to improve the performance of the execution of the program.

It helps to reduce the amount of time needed for the execution of the program.

How Interpreter and JIT compiler work simultaneously in Java?

interpreter it reads each line seprately and compile and then executes

Native method interface

Java native libraries

classloader

loading (3 components) linking initilaization

1 Bootstrap classloader verify all static variables are assigned with original values

2 Extention class loader prepare static blocks will be executed from top to bottom

3 Application class loader Resolve

classloader is resposible for manily 3 tasks

1 loading(it loads all .class files in memory) (Bootstrap classloader ,Extention class loader , application class loader)

2 linking (verification ,prepartion(it allocates class level data (static variables) in memory and intilizes its defult value Resolution(is the process of replacing the symbolic names in our programs with original name refernces)

3 Intilization (it replaces default values of static variables by its original values and executes all static blocks from top to bottom)

Class Loader SubSystem

Loading: It will read .class file and store corressponding information in the method area.

• For each class file, JVM will store following information in the method area:

1. Fully qualified class name.

2. Fully qualified parent class name.

3. Methods information.

4. Variables information.

5. Constructors information.

6. Modifiers information.

7. Constant pool information. etc...

3 types of class loaders:

1. Bootstrap Class Loader: Responsible to load the classes present in rt.jar (rt.jar file is present in bootstrap classpath i.e. jdk\jre\lib)

2. Extention Class Loader: Responsible to load the classes from extention class path(i.e. jdk\jre\lib\ext\\*.jar)

3. Application Class Loader: Responsible to load the classes from application classpath.It internally uses environment variable class path

Class Loader SubSystem

Linking: In linking 3 activities are performed:

1. Verification

2. Preparation

3. Resolution

• Verification

In this process Byte Code Verifer checks wether the class file is

generated by valid compiler or not and whether.class file is properly formated or not. If verification fails,

then JVM will provide "java.lang.VerifyError" exception. Because of this process, java is secured.

•Preparation

In this process JVM will allocate memory for class level static variables & assign default values (not original values)

Resolution

In this process symbolic names present in our program are replace with original memory references from method area.

Initialisation:In this process, two activities will be performed:

1. All static variables are assigned with original values.

2. static blocks will be executed from top to bottom

Execution engine(central component of jvm)

interpreter | Jit compiler | Garbage collector ,Security Manager etc|

intermediate code Generator

|

V

Code Optimizer

|

V

Target code Generator

|

V

Machine code/native code

Execution engine

Execution Engine is responsible to execute java class file.

It contains mainly two components:

1. Interpreter

2. JIT Compiler

Interpreter:

A module that alternately decodes and executes every statement or line in some body of code. The Java interpreter decodes and executes bytecode for the

Java virtual machine

byte code --->interpret-------->machine code or native code----------->execute

JIT Complier:

JIT stands for Just-in-Time which means that code gets compiled when it is needed, not before runtime.

The main purpose of JIT compiler is to improve performance.

JVM maintains a count as of how many time a function is executed. If this

count exceeds a predefined limit or say threshold value, the JIT compiles

the code into machine language which can directly be executed by the

processor (unlike the normal case in which javac compile the code into

bytecode and then java - the interpreter interprets this bytecode line by line

converts it into machine code and executes).

Also next time this function is calculated same compiled code is executed

again unlike normal interpretation in which the code is interpreted again line by line. This makes execution faster.

JIT compilation is applicable only for repeatedly required methods, not for

every method

Java Native Interface (JNI)

• An interface that allows Java to interact with code written in another language.

It acts as mediator for java method calls & the corresponding native libraries i.e. JNI is responsible to provide information about native libraries to the JVM.

• Native Method Library provides or holds native library information.

• The java command-line utility is an example of one such application, that launches Java code in a Java Virtual Machine.