Java Interview Questions and Answers

Java/J2ee interview Questions and Answers.

Sunday, 2 April 2017

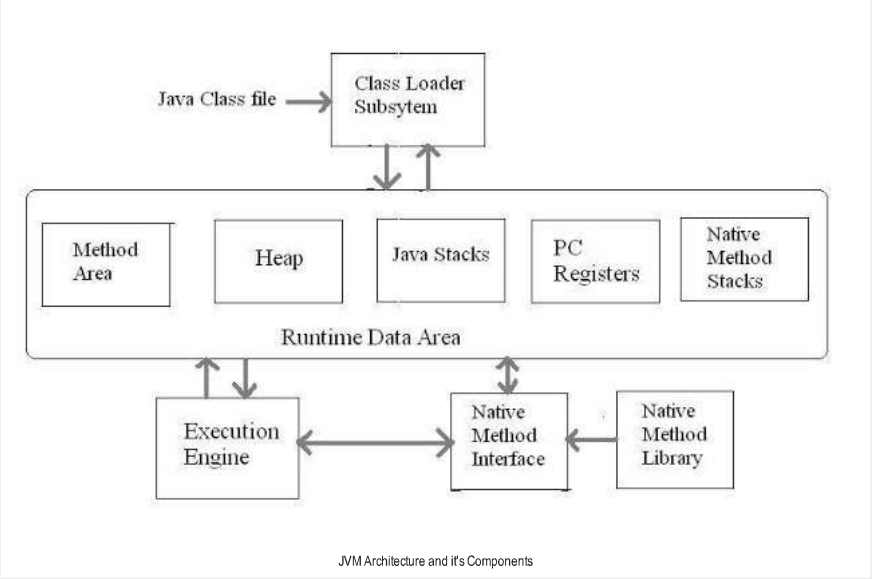
Java Vertual Machine(JVM) Architecture in Java

In this post, we will learn more about JVM Architecture in Java and components of JVM in detail.

What is JVM ?

JVM is a software implementation of a physical machine. Java was developed with the concept of *Write Once Runs Anywhere*, which runs on a Virtual Machine. The compiler compiles the Java file into a Java .class file, then that .class file is input to the JVM, which loads and executes the class file.

JVM Architecure Design:



JVM Architecture and it's Components

How does the JVM works?

As shown in the above diagram, a JVM is divided into three main subsystems,

1) **Class Loader Subsystem**
2) **Runtime Data Area**
3) **Exceution Engine**

1) Class Loader subsystem

Java's **dynamic class loading** functionality is handled by the class loader subsystem. It loads, links and initializes the class when it refer to the class for the first time at run time, not at compile time. It performs three major functionality such as Loading, Linking and Intialization.

1.1) Loading

Classes will be loaded by this component. There are three class loaders which can help to load the classes at run time as BootStrap ClassLoader, Extension ClassLoader and Application ClassLoader.

1) BootStrap ClassLoader
Responsible for loading the classes from the bootstrap classpath , nothing but *rt.jar*. Highest priority will be given to this loader

2) Extension ClassLoader
Responsible for loading classes which are inside *ext* folder (jre/lib).

3) Application ClassLoader
Responsible for loading Application Level Classpath, path mentioned environment variable etc.

1.2) Linking

1) Verify - Byte code verifier will verify if the generated byte code is proper or not. If not a proper then will get Verification error.

2) Prepare - For all static variables memory will be allocated and assigned with default values.

Bangalore is the best city to live in India?

Yes
No
Vote Show results

Votes so far: 52
Days left to vote: 99

Popular Posts

Difference between Loose Coupling and Tight Coupling in Java With Examples.

Difference between Comparable & Comparator Interface in Java

How HashMap works internally in Java?

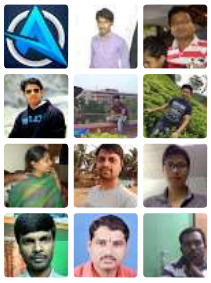
Singleton Design Pattern in java with example

About Me

Anil Nivargi
G+ Follow 49
View my complete profile

Google+ Followers

Anil Nivargi
Add to circles



49 have me in circles View all

Total Pageviews

46,996

Blog Archive

▼ 2017 (2)
▼ April (1)
Java Vertual Machine(JVM) Architecture in Java
► February (1)

3) Resolve - For all symbolic memory references are replace with original references from method area.

1.3) Intialization

This is the final phase of the class loading , here all static variables will be assigned with the original values and the static block will be executed.

2) Runtime Data Area

The Runtime Data Area is divided into 5 major components, as below

1) Method Area

All the class level data will be stored here, including the static variables. There is only one method area per JVM and it's shared resource.

2) Heap area

All the objects and their corresponding instance variables and Arrays will be stored here. There is only one Heap Area per JVM. Since the Method and Heap area share memory for multiple threads, data stored is not thread safe.

3) Stack Area

For every thread, a separate runtime stack will be created. For every method call , one entry will be made in the stack memory which is called as **Stack Frame.** All Local variables will be created in the stack memory. The Stack Area is a **thread safe** since it's not a shared resource. The Stack Area is divided into three subentities : **Local Variable Array, Operand Stack & Frame data.**

4) PC Registers

Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC Register will be updated with the next instruction

5) Native Method Stacks

Native method stack holds native method information. For every thread, a separate native method stack will be created.

3) Execution Engine

The byte code which is assigned to the Runtime Data Area will be executed by the Execution Engine. The Execution Engine reads the byte code and executes the code step by step as follows,

1) Interpreter - The Interpreter interprets the byte code faster but executes slowly. The disadvantage of interpreter is that when one method is called multiple times, every time a new interpretation is required.

2) JIT Compiler - The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of interpreter in converting the byte code, but when it finds the repeated code it uses the **JIT Compiler**, which compiles the entire byte code and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.

3) Garbage Collector - Collects and removes the unreferenced objects. Garbage collection can be triggered by calling "**System.gc**()", but the execution is not gauranteed. Garbage collection of the JVM collects the objects that are created.

Java Native Interface(JNI) : -

JNI will be interacting with the **Native Method Libraries** and provides the Native Libraries for the Execution Engine.

Native Method Libraries: -

It is a collection of the Native Libraries which is required for the Execution Engine.

Posted by Anil Nivargi at 4/02/2017 12:27:00 am 2 comments: Links to this post

Labels: Core Java Interview Questions with Answers, java

Location:India Bengaluru, Karnataka, India

Home Older Posts

Subscribe to: Posts (Atom)

► 2016 (5)
► 2015 (10)
► 2014 (31)