JVM Tutorial

by Ashok Kumar *on* 03:09:00 *in* Core Java, JAVA, JVM

What is Virtual?

This is not having physical existence

What is Virtual machine?

* Virtual machine is a simple software program which simulates the functions of a physical machine.

* This is not having physical existence, but which can perform all operations like physical machines.

* Physical machine whatever it is doing, all those things we can able to do on the virtual machine.

* Best example of virtual machine is calculator in computer; it is worked like physical calculator.

All virtual machines categorized in to 2 types
1. Hardware based or System based Virtual Machine
2. Software based or Application based or process based Virtual Machine

Hardware based virtual machines

Physically only one physical machine is there but several logical machines we can able to create on the single physical machine with strong isolation (worked independently) of each other. This type of virtual machines is called hardware based virtual machines.

Main advantage of hardware based virtual machine is effective utilization of hardware resources. Most of the times these types of virtual machines associated with admin role (System Admin, Server Admin etc). Programmers never associated with the hardware based virtual machines.
Examples of hardware based virtual machines are
1. KVM (Kernel Based VM) for Linux systems
2. VMware
3. Xen
4. Cloud Computing

Software based virtual machines

These virtual machines acts as run time engine to run a particular programming language applications.
Examples of software based virtual machines are
1. JVM (Java Virtual Machine) acts as runtime engine to run Java applications

FACEBOOK

Waytoeasylearn
472 likes
Like Page Send Mess
Be the first of your friends to like this

ABOUT ME

Ashok Kumar
Follow 71
View my complete profile

FOLLOW ME

FOLLOW ON TWITTER
LIKE ON FACEBOOK
SUBSCRIBE ON YOUTUBE
FOLLOW ON INSTAGRAM

POPULAR POSTS

Core Java Tutorials Index
JVM Tutorial
XML Tutorial
Threads And Concurrency Tutorial
Hibernate Generator Classes

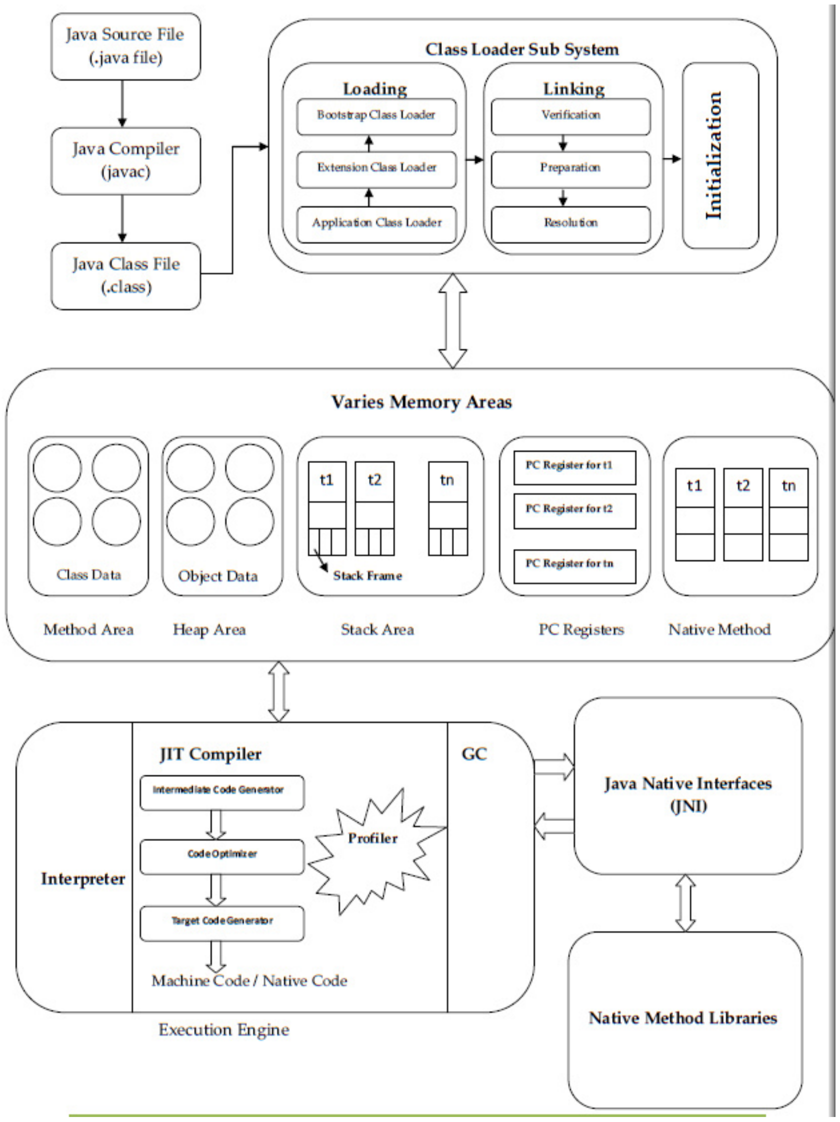
RECENT COMMENTS

Hadoop Tutorial Index

2. PVM (Parrot Virtual Machine) acts as runtime engine to run Perl applications
3. CLR (Common Language Runtime) acts as runtime engine to run .NET based applications

JVM

JVM is the part of JRE and it is responsible to load and run the java class file. The following picture depicts basic architecture of the JVM.



The first component in JVM is Class Loader Sub System

1. Class Loader Sub System

This system is responsible for loading .class file with 3 activities

i. Loading
ii. Linking
iii. Initialization

i. Loading

Loading means read .class file from hard disk and store corresponding binary data inside method area of JVM. For each .class file JVM will store following information

1. Fully qualified name of class
2. Fully qualified name of immediate parent
3. Whether .class file represents class|interface|enum

Ashok Kumar Apr 26, 2018

Hive Architecture
Ashok Kumar Apr 26, 2018

Apache Hive Introduction
Ashok Kumar Apr 26, 2018

Hadoop Counters
Ashok Kumar Apr 17, 2018

Speculative Execution in Hadoop
Ashok Kumar Apr 16, 2018

TAGS

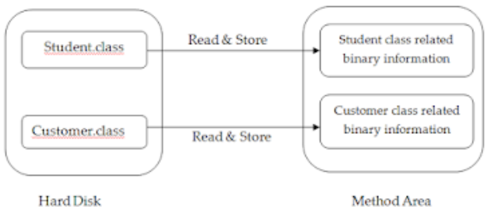
ADVANCED JAVA (15) BIGDATA (49)
CASSANDRA (4) CORE JAVA (47)
DATABASE (22) DATASTRUCTURES (5)
FRAMEWORKS (19) HADOOP (27)
HBASE (15) HIBERNATE (15) HIVE (2)
JAVA (63) JAVA DESIGN PATTERNS (11)
JAVA REFLECTION (1) JENKINS (4) JSVC
JVM (3) KAFKA (2) LOG4J (1)
NETAPP (2) PL SQL (10) PROTOBUF (2)
REGULAR EXPRESSION (1) SPRING (4)
SQL (12) STORAGE (5) TESTNG (4)
XML (7) ZOOKEEPER (1)

BLOG ARCHIVE

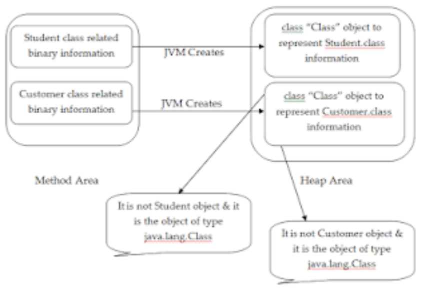
April (13)

4. Methods|Constructors|Variables information
5. Modifiers information
6. Constant Pool information

After loading the class file and store inside method area, immediately JVM will perform one activity i.e., create an object of type java.lang.Class in heap area.


Student.class
Customer.class
Read & Store
Read & Store
Student class related binary information
Customer class related binary information
Hard Disk
Method Area

Created object is not student object or customer object. It is a predefined class “Class” object that is presently in java.lang package. The created object is represents either student class binary information or customer class binary information.

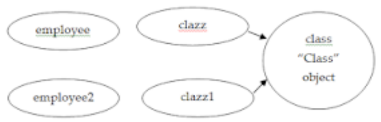

Student class related binary information
Customer class related binary information
JVM Creates
JVM Creates
class “Class” object to represent Student.class information
class “Class” object to represent Customer.class information
Method Area
Heap Area
It is not Student object & it is the object of type java.lang.Class
It is not Customer object & it is the object of type java.lang.Class

Here the created class “Class” object is used by programmer. For example,

Code
```
package com.ashok.jvm;

public class Employee {
   private int empId;
   private String empName;

   public int getEmpId() {
      return empId;
   }
   public void setEmpId(int empId) {
      this.empId = empId;
   }

   public String getEmpName() {
      return empName;
   }
   public void setEmpName(String empName) {
      this.empName = empName;
   }
}
```
Code
```
package com.ashok.jvm;

import java.lang.reflect.Method;
import java.lang.reflect.Field;

public class Test {
   public static void main(String[] args) throws ClassNotFoundException {
      Class clazz = Class.forName("com.ashok.jvm.Employee");
      Method[] methods = clazz.getDeclaredMethods();
      for(Method method : methods) {
          System.out.println(method);
      }
      Field[] fields = clazz.getDeclaredFields();
      for(Field field : fields) {
          System.out.println(field);
      }
   }
}

Output is :
public java.lang.String com.ashok.jvm.Employee.getEmpName()
public void com.ashok.jvm.Employee.setEmpId(int)
public void com.ashok.jvm.Employee.setEmpName(java.lang.String)
public int com.ashok.jvm.Employee.getEmpId()
private int com.ashok.jvm.Employee.empId
private java.lang.String com.ashok.jvm.Employee.empName
```

For every loaded .class file, only one class "Class" object will be created by JVM, even though we are using that class multiple times in our program. Example,

Code
```
package com.ashok.jvm;

public class Test2 {
   public static void main(String[] args) {
      Employee employee = new Employee();
      Class clazz = employee.getClass();
      Employee employee2 = new Employee();
      Class clazz1 = employee2.getClass();
      System.out.println(clazz.hashCode());
      System.out.println(claz.hashCode());
   }
}
Output is:
1704856573
1704856573
```



Here Employee class object created two times, but class is loaded only once. In the above program even through we are using Employee class multiple times only one class Class object got created.

ii. Linking

After "loading" activity JVM immediately perform Linking activity. Linking once again contain 3 activities,

1. Verification
2. Preparation

3. Resolution

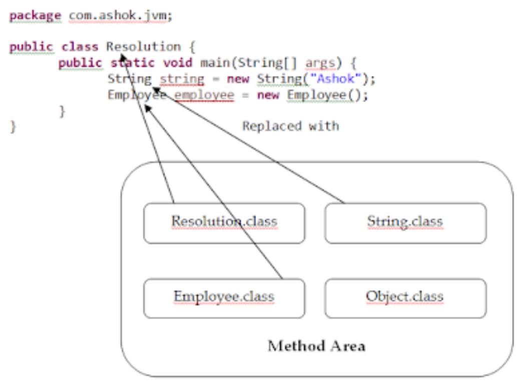
Java language is the secure language. Through java spreading virus, malware these kind of this won't be there. If you execute old language executable files (.exe) then immediately we are getting alert message saying you are executing .exe file it may harmful to your system.

But in java .class files we never getting these alert messages. What is the reason is inside JVM a special component is there i.e., Byte Code Verifier. This Byte Code Verifier is responsible to verify weather .class file is properly formatted or not, structurally correct or not, generated by valid compiler or not. If the .class file is not generated by valid compiler then Byte Code Verifier raises runtime error java.lang.VerifyError. This total process is done in **verification** activity.

In **preparation** phase, JVM will allocate memory for class level static variables and assigned default values.
E.g. For int ---> 0, For double ---> 0.0, For boolean ---> false

Here just default values will be assigned and original values will be assigned in initialization phase.

Next phase is **Resolution**. It is the process of replacing all symbolic references used in our class with original direct references from method area.



```
package com.ashok.jvm;

public class Resolution {
        public static void main(String[] args) {
                String string = new String("Ashok");
                Employee employee = new Employee();
        }
}
```

For the above class, class loader sub system loads **Resolution.class, String.class, Student.class** and **Object.class**. Every user defined class the parent class is Object.class so every sub class its parent class must be loaded. The names of these classes are stored in "Constant Pool" of "Resolution" class.
In Resolution phase these names are replaced with Actual references from the method area.

iii. Initialization

In Initialization activity, for class level static variables assigns original values and static blocks will be executed from top to bottom.

While Loading, Linking and Initialization if any error occurs then we will get runtime Exception saying java.lang.LinkageError. Previously we discussed about VerifyError. This is the child class of LinkageError.
Types of class loaders in class loader subsystem

1. Bootstrap class loader/ Primordial class loader
2. Extension class loader
3. Application class loader/System class loader

1. Bootstrap class loader

Bootstrap class loader is responsible for to load classes from bootstrap class path. Here bootstrap class path means, usually in java application internal JVM uses rt.jar. All core java API classes like String class, StringBuilder class, StringBuffer class, java.lang packages, java.io packages etc are available in rt.jar. This rt.jar path is known as bootstrap class path and the path of rt.jar is

Code
```
jdk --> jre --> lib --> rt.jar
```

This location by default consider as bootstrap class path. This Bootstrap class loader is responsible for loading all the classes inside this rt.jar. This Bootstrap class loader is implemented not in java it is implemented by native languages like C, C++ etc.

2. Extension class loader

The extension class loader is the child of bootstrap class loader. This class loader is responsible to load classes from extension class path

Code
```
jdk --> jre --> lib-->ext -->*.jar
```

The extension class loader is responsible for loading all the classes present in the ext folder. This Extension class loader is implemented in java only. The class name of extension class loader is

Code
```
sun.misc.Launcher$ExtClassLoader.class
```

3. Application class loader

The Application class loader is the child of Extension class loader. This class loader is responsible to load classes from Application class path. Application class path means classes in our application (Environment variable class path). It internally uses environment vatiable path. This Application class loader is implemented in java only. The class name of application class loader is

Code
```
sun.misc.Launcher$AppClassLoader.class
```

E.g

Code
```
package com.ashok.jvm;

public class LoaderTest {
   public static void main(String[] args) {
       System.out.println(String.class.getClassLoader());
       System.out.println(Employee123.class.getClassLoader());
       System.out.println(LoaderTest.class.getClassLoader());
   }
}

Output:
null  // Because Bootstrap class loader is not java object it is designed with C or
sun.misc.Launcher$ExtClassLoader@33909752 //Here i am created class Employee123 and
sun.misc.Launcher$AppClassLoader@73d16e93
```

Next Tutorial JVM Tutorial Part 2

Tags # Core Java # JAVA