|  |  |
| --- | --- |
| Difference between Stack and heap memory   |  | | --- | | * Stack = region of memory for temporaries Stack pointer pushed on function entry Popped on function exit Stack data: new every time stack is used for local variables Heap = distinct region of memory for persistent objects Allocate persistent data on heap with new keyword Dynamically allocated memory (heap)exists until it is released either explicitly by the programmer C/C++ ? explicitly managed | |
| |  | | --- | | * STACK memory is referred as temporary memory,if you come out of the program the memory of the variable will not no more there.[eg., int a; memory for a will not maintained after v came out from the program].  HEAP memory is referred as permanent memory,memory allocated for the object will be maintained even if we came out of the program.[eg.memory for OBJECT will remains there ever]. | |
| |  | | --- | | * Heap is allocated to a process by OS on start; therefore if program will be terminated entire heap memory will be freed. | |
| |  | | --- | | * The heap (also known as the free store) is a large pool of memory used for dynamic allocation. In C++, when you use the new operator to allocate memory, this memory is assigned from the heap.   The call stack (usually referred to as the stack) has a much more interesting role to play.about the call stack, which refers to a particular portion of memory. | |
| |  | | --- | | * Stack is a temporary memory.it is used for local variables.when u come out from the program , the allocated memory of variable was freed.    heap:it is a parmanent memory.it is used for dynamic memory allocation.when we come out from the program still the memory will be allocated like that. | |
| |  | | --- | | * Stack Segment : used as a container for local variables and their temporary information.  Heap segment:used to store data dynamically at run time using dynamic variables.  A dynamic variable is stored in heap, outside the data and stack segments. The attributes of a dynamic  (pointer) for ex: int \*x; Name: an arbitrary identifier Type: integer address Scope: local or global (depending n the situation) Address:the addressin the data segment (if it is global) or stack segment (if it is local) Size: 16 bits(8bits for segment+8bits for offset of the address it pint to)  This tells the compiler we have a pointer x that may hold the strting address of a dynamic variable of tpe integer. | |
| |  | | --- | | * Heap: The heap is a large set of memory maintained for persistent storage and the memory is dynamically allocated.If the user came out of the program the allocated memory of the data will not be removed. Stack: Stack is maintained for local variables. If the user came out of the program the memory allocated to the data will be deleted.   [**Difference between Stack and Heap memory in Java**](http://javarevisited.blogspot.in/2013/01/difference-between-stack-and-heap-java.html)  Difference between stack and heap memory is common [programming question](http://javarevisited.blogspot.com/2011/06/top-programming-interview-questions.html) asked by beginners learning Java or any other programming language. Stack and heap memory are two terms programmers starts hearing once they started programming but without any clear and definite explanation. Lack of knowledge on [what is heap in Java](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) and what is stack memory in Java, results in misconcepts related to stack and heap. To add to this confusion, stack is also a data structure which is used to store elements in LIFO(Last In First out) order and available in Java API as java.util.Stack. In general both stack and heap are part of memory, a program is allocated and used for different purposes. Java program runs inside JVM which is launched as a process by "java" command. Java also uses both stack and heap memory for different needs. In our last article [10 points on Java heap space](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) I have touched base on Java heap space and in this article we will see difference between stack and heap memory in Java.    Difference between Stack vs Heap in Java  [Difference between heap and statck in Java - Interview Question](http://1.bp.blogspot.com/-p7EVk3HnwGw/UCJuX-7mWpI/AAAAAAAAAb0/rTwwhrgC3Dc/s1600/di-logo-java-orange.png)  Here are few differences between stack and heap memory in Java:  1) Main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call, while heap memory is used to store objects in Java. No matter, where object is created in code e.g. as member variable, local variable or class variable,  they are always created inside heap space in Java.  2) Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has there own stack which can be specified using -Xss JVM parameter, similarly you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of heap and -Xmx is maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).  3) If there is no memory left in stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError  in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).  4) If you are using [Recursion](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html), on which method calls itself, You can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is lot lesser than size of  heap memory in Java.  5) Variables stored in stacks are only visible to the owner Thread, while objects created in heap are visible to all thread. In other words stack memory is kind of private memory of Java Threads, while heap memory is shared among all threads.  That's all on **difference between Stack and Heap memory in Java**. As I said, It’s important to understand what is heap and what is stack in Java and which kind of variables goes where, how you can run out of stack and heap memory in Java etc. Let us know if you are familiar with any other difference between stack and heap memory in java  Read more: <http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html#ixzz2zPsKFBXu> [10 Examples of HotSpot JVM Options in Java](http://javarevisited.blogspot.in/2011/11/hotspot-jvm-options-java-examples.html) There are hundreds of **JVM parameters** or **JVM Options** exists inside sun JDK and its virtually impossible to keep track of every single [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html)option and based on my experience we don't even use most of JVM flags except couple of important JVM option related to java heap size, java options for printing garbage collection details and most likely JVM switches for setting up remote debugging in Java. but there are many other useful category of JVM parameters which you at least like to be familiar even if not intending to use it more frequently. In this article we will see examples of 10 different categories of **JVM parameter** which I found useful and use more frequently than other. I would recommend to get a full knowledge of what does a particular JVM options does by referring official list of JVM options. JVM parameters in Java On the basis of how we specify **JVM option it can be divided into two parts**, JVM Options which starts with –X and those which starts with -XX:  1)    *JVM Options that begin with -X* are non-standard (thy are not guaranteed to be supported on all JVM implementations), and are subject to change without notice in subsequent releases of the JDK.  2)    *JVM Options or parameters which are specified with -XX* are not stable and are not recommended for casual use. These options are subject to change without notice also.  [Frequently used JVM parameters for heap, GC and debugging](http://4.bp.blogspot.com/-KMPS7glbJhY/T8cc_8QtQqI/AAAAAAAAAX8/hmMqLBx7m4k/s1600/Hotspot2_JVM_Parameters_GC_Heap.png)  I was thinking about writing post on JVM options when I completed my post on [**Java Heap Size**](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) and [**Java Garbage Collection**](http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html) because these are two main area where we see usages of various JVM flags. But it didn’t happened even after I covered OutOfMemoryError post which has some[**JVM option to solve OutOfMemoryError in Java**](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html). Now I am happy that I have completed this piece of information and its ready to be published. As always I look for your feedback, suggestions and any other JVM flags which I have missed and you guys find useful to share.   Good knowledge of JVM options specially related to GC tuning is important for time critical application e.g. **high volume low latency electronic trading platform** where every micro seconds matter. though getting right combination requires lot of profiling and trial and error and depends heavily on nature of trading application. Important Points about JVM Options: 1)    Boolean JVM options can be  turned on with -XX:+ and can be turned off with -XX:-.  2)    Numeric JVM Options can be set with -XX:=. Numbers can include 'm' or 'M' for megabytes, 'k' or 'K' for kilobytes, and 'g' or 'G' for gigabytes (for example, 32k is the same as 32768).  3)    String JVM options can be set by using -XX:=, and usually used to specify a file, a path, or a list of commands.  The command **java -help** lists the standard options (standard across different JVM implementations) for the Java application launcher. The **command java -X can be used to see the Java application launcher's non-standard** (X for extension specific to that JVM) arguments.The -X options are non-standard and subject to change without notice. If you wish to detect which JVM arguments your currently running Java application is using, you can use the ManagementFactory.getRuntimeMXBean().getInputArguments()  Now here is my list of important JVM flags, switches, options or parameters which is most commonly used while running Java applications:  **1) JVM memory options related to java heap size**  Following three JVM options are used to specify initial and max heap size and thread stack size while running Java programs.  **-Xms**        set initial Java heap size  **-Xmx**        set maximum Java heap size  **-Xss**>         set java thread stack size  **2) JVM option to print gc details**  **-verbose:gc** logs garbage collector runs and how long they're taking. I generally use this as my first tool to investigate if GC is a bottleneck for a given application.  **-XX:+PrintGCDetails** includes the data from -verbose:gc but also adds information about the size of the new generation and more accurate timings.  **-XX:-PrintGCTimeStamps**  Print timestamps at garbage collection.  **3) JVM parameters to specify Java Garbage collector**  **-XX:+UseParallelGC**      Use parallel garbage collection for scavenges  **-XX:-UseConcMarkSweepGC** Use concurrent mark-sweep collection for the old generation. (Introduced in 1.4.1)  **-XX:-UseSerialGC**        Use serial garbage collection. (Introduced in 5.0.)  beware when you use GC Parameters if you are working on time critical application e.g. high frequency trading application. As  GC is time consuming operation and its desired to create a balance.  **4) JVM debug options JVM options for remote debugging**  -Xdebug -Xnoagent -Xrunjdwp:transport=dt\_socket,server=y,suspend=n,address=8000 to read more about remote debugging check [How to Setup Java remote debugging in Eclipse](http://javarevisited.blogspot.com/2011/02/how-to-setup-remote-debugging-in.html) and [10 Java debugging tips in Eclipse](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html)  **5) JVM options related to profiling**  -Xprof  -Xrunhprof  **6) JVM options related to java classpath**  **Xbootclasspath** specifies classpath entries you want loaded without verification. The JVM verifies all classes it loads to ensure they don't try to dereference an object with an int, pop extra entries off the stack or push too many, and so on. This verification is part of the reason why the JVM is very stable, but it's also rather costly, and responsible for a large part of start up delay. Putting classes on the bootclasspath skips this cost, but should only be used when you know the classes have been verified many times before. In JRuby, this reduced startup time by half or more for a simple script. The -**Xbootclasspath** option can be used to either prepend (/p) or append (/a) resources to the bootstrap classpath. You Can read more about Java Classpath in my articles [How Classpath Works in Java](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html) and [How to Solve ClassNotFoundException in Java](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html)  **7) JVM options to change  Perm Gen Size**  These JVM optiosn are quite useful to solve [java.lang.OutOfMemoryError:Perm Gen Space](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).  -XX:PermSize and MaxPermSize  -XX:NewRatio=2  Ratio of new/old generation sizes.  -XX:MaxPermSize=64m     Size of the Permanent Generation.  **8) JVM parameters to trace classloading and unloading**  **-XX:+TraceClassLoading** and **-XX:+TraceClassUnloading** are two JVM options which we use to print logging information whenever classes loads into JVM or unloads from JVM. These JVM flags are extremely useful if you have any memory leak related to classloader and or suspecting that classes are not unloading or garbage collected.  **9) JVM switches related to logging**  -XX:+TraceClassLoading and -XX:+TraceClassUnloading print information class loads and unloads. Useful for investigating if you have a class leak or if old classes (like JITed Ruby methods in JRuby) are getting collected or not. You can read more about logging in Java on my post [10 Tips while logging in Java](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html)  **-XX:+PrintCompilation** prints out the name of each Java method Hotspot decides to JIT compile. The list will usually show a bunch of core Java class methods initially, and then turn to methods in your application. In JRuby, it eventually starts to show Ruby methods as well  **10) JVM Switches for debugging purpose**  -XX:HeapDumpPath=./java\_pid.hprof  Path to directory or file name for heap dump.  -XX:-PrintConcurrentLocks       Print java.util.concurrent locks in Ctrl-Break thread dump.  -XX:-PrintCommandLineFlags   Print flags that appeared on the command line.  That’s all on JVM Options, I understand its not possible to remember all JVM flags but at-least having an idea of what kind of JVM flags are available is good asset. Image for JVM parameters is from Java tuning and Nutshell.  For full list of JVM options you can refer these link from Oracle Java site: [Java Hotspot VM Options](http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html)  Read more: <http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html#ixzz2zPsadHCZ> | |