How They Work: In a Nutshell

A guide to Eyamaz's JVM arguments

This guide is an explanation into how the different sections of my current iteration of JVM arguments work and what they do. In a nutshell (as simply as possible :P)

Memory

This section of the arguments affects the memory allocated to the running VM.

-Xms2048m -Xmx2048m -XX:PermSize=256m -XX:ReservedCodeCacheSize=512m -XX:NewRatio=2 -XX:SurvivorRatio=2 -XX:TargetSurvivorRatio=80 -XX:MaxTenuringThreshold=8

Xms/Xmx control the overall size of the Heap. Specifically for how these arguments function, I keep these equal primarily to avoid the overhead of the need to calculate and dynamically resize the Heap on the fly.

PermSize controls the *initial* size of the Permanent Generation. This portion of the memory contains string literals and interpreted methods. I set this to 256 specifically for over the top sized modpacks. I don't expressly set a maximum size for PermGen since it rarely ever exceeds the initial size from my testing. If needed, -XX:MaxPermSize=256m can be appended to the arguments.

ReservedCodeCacheSize controls the maximum size of the CodeCache. This portion of the memory contains compiled bytecode. With this current set of arguments, I have yet to see this limit reached. If a modpack ever forces this limit to be reached, JIT will automatically poll the CodeCache and remove the least used methods.

//Eyamaz: The amount of memory java specifically reserves for the use of the VM is equal to Xms + PermSize + InitialCodeCacheSize with a maximum memory allocation of Xmx + MaxPermSize + ReservedCodeCacheSize. With these arguments at base values, our required memory is 2048m + 256m + 512m or 2816m (I list this as a 3GB requirement, even though its a little less.)

NewRatio controls the size of eden space, or young generation. Setting this to 2 allows me to keep the eden space at ⅓ of the total heap size, regardless as to what Xmx is set at.

SurvivorRatio controls the size of the To and From survivor spaces. These spaces are the location that live objects are moved through when going from the eden space to the old generation.

TargetSurvivorRatio and **MaxTenuringThreshold** are used to configure the amount of data kept alive when moving to and from the survivor spaces.

Garbage Collection

This section of the arguments affects how the memory is managed.

-XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:SoftRefLRUPolicyMSPerMB=0 -XX:MaxGCPauseMillis=40
-XX:GCPauseIntervalMillis=150 -XX:MaxGCMinorPauseMillis=7 -XX:+CMSClassUnloadingEnabled
-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses -XX:+UseCMSInitiatingOccupancyOnly
-XX:CMSInitiatingOccupancyFraction=50 -XX:+BindGCTaskThreadsToCPUs

UseParNewGC and **UseConcMarkSweepGC** are used to enable the Parallel New garbage collector and the Concurrent Mark Sweep garbage collector. As of right now, I am still studying the source code to see how to tweak the G1GC to be a viable garbage collector. However, it has proven not to be in previous testing.

SoftRefLRUPolicyMSPerMB this is used to configure the amount of time in milliseconds that Soft References are kept alive. Setting this number to anything higher than 0 can result in memory leaks! I'm still testing with this variable to see if I can prevent this leak another way.

MaxGCPauseMillis, **GCPauseIntervalMillis**, and **MaxGCMinorPauseMillis** affect the amount of time the garbage collectors are allowed to spend doing their job. These are specifically used to avoid stop the world activity from garbage collection. Yes, they do affect ParNew and CMS as of J7, they do not in J6 (unless this feature was backported.)

CMSClassUnloadingEnabled and **ExplicitGCInvokesConcurrentAndUnloadsClasses** are used in place of DisableExplicitGC for when a "performance" mod, or vanilla minecraft, makes a call to system.gc().

*//Keybounce: "Vanilla client wants to do a full GC every time you change dimensions. This can easily mean that your client does nothing for 10 seconds, and potentially over 30 seconds. While going into the nether is safe (unless something shoots out the portal while you are in it), and going into the deep dark *was* safe (now mobs can spawn at any light level), going into arbitrary mystcraft ages and being unable to react for 10-20 seconds can be death."*

UseCMSInitiatingOccupancyOnly and **CMSInitiatingOccupancyFraction** are used to control when the CMS collector begins its collection.

BindGCTaskThreadsToCPUs attempts to lock each separate GC thread to a specific CPU (starting at CPU0 and working up.) This allows us to forgo trying to specify a specific number of threads for ParNew and CMS as we wont have GC threads fighting for the same core.

The Tiered Compiler

This section of the arguments are used to tweak the compiler.

-XX:+TieredCompilation -XX:Tier0ProfilingStartPercentage=0 -XX:Tier3InvocationThreshold=3 -XX:Tier3MinInvocationThreshold=2 -XX:Tier3CompileThreshold=2 -XX:Tier3BackEdgeThreshold=10 -XX:Tier4InvocationThreshold=4 -XX:Tier4MinInvocationThreshold=3 -XX:Tier4CompileThreshold=2 -XX:Tier4BackEdgeThreshold=8 -XX:TieredCompileTaskTimeout=1000 -XX:+UseFastEmptyMethods -XX:-DontCompileHugeMethods -XX:+AlwaysCompileLoopMethods -XX:+CICompilerCountPerCPU

TODO

Other Optimizations

This section of the arguments are misc. optimizations.

-XX:+UseStringCache -XX:+UseNUMA

TODO