JVM参数

在JVM调整过程中，主要是对JVM参数做的调整，以下我们队JVM主要参数做逐一介绍。JVM参数有很多，其实我们直接使用默认的JVM参数，不去修改都可以满足大多数情况。但是如果你想在有限的硬件资源下，部署的系统达到最大的运行效率，那么进行相关的JVM参数设置是必不可少的。下面我们就来对这些JVM参数进行详细的介绍。

JVM参数主要分为以下三种：标准参数、非标准参数、不稳定参数。

1 标准参数

标准参数，顾名思义，标准参数中包括功能以及输出的结果都是很稳定的，基本上**不会随着JVM版本的变化而变化**。标准参数以-开头，如：java -version、java -jar等，通过java -help可以查询所有的标准参数，

我们可以通过 -help 命令来检索出所有标准参数。

```
➜  ~ java -help
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server,
                  because you are running on a server-class machine.


    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                  Warning: this feature is deprecated and will be removed
```

关于这些命令的详细解释，可以参考官网：

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html

-help 也是一个标准参数，再比如使用比较多的 -version也是。

显示Java的版本信息。

```
➜  ~ java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

2 非标准参数

非标准参数以-X开头，是标准参数的扩展。对应前面讲的标准化参数，这是非标准化参数。表示在将来的JVM版本中可能会发生改变，但是这类以-X开始的参数变化的比较小。

我们可以通过 Java -X 命令来检索所有-X 参数。

```
➜  ~ java -X
    -Xmixed           mixed mode execution (default)
    -Xint             interpreted mode execution only
    -Xbootclasspath:<directories and zip/jar files separated by :>
                      set search path for bootstrap classes and resources
    -Xbootclasspath/a:<directories and zip/jar files separated by :>
                      append to end of bootstrap class path
    -Xbootclasspath/p:<directories and zip/jar files separated by :>
                      prepend in front of bootstrap class path
    -Xdiag            show additional diagnostic messages
    -Xnoclassgc       disable class garbage collection
    -Xincgc           enable incremental garbage collection
    -Xloggc:<file>    log GC status to a file with time stamps
    -Xbatch           disable background compilation
    -Xms<size>        set initial Java heap size
    -Xmx<size>        set maximum Java heap size
    -Xss<size>        set java thread stack size
    -Xprof            output cpu profiling data
    -Xfuture          enable strictest checks, anticipating future default
    -Xrs              reduce use of OS signals by Java/VM (see documentation)
    -Xcheck:jni       perform additional checks for JNI functions
    -Xshare:off       do not attempt to use shared class data
    -Xshare:auto      use shared class data if possible (default)
    -Xshare:on        require using shared class data, otherwise fail.
    -XshowSettings    show all settings and continue
    -XshowSettings:all
                      show all settings and continue
```

我们可以通过设置非标准参数来配置堆的内存分配，常用的非标准参数有：

-Xmn新生代内存的最大值，包括Eden区和两个Survivor区的总和，写法如：-Xmn1024，-Xmn1024k，-Xmn1024m，-Xmn1g 。

-Xms堆内存的最小值，默认值是总内存/64（且小于1G），默认情况下，当堆中可用内存小于40%(这个值可以用-XX: MinHeapFreeRatio 调整，如-X:MinHeapFreeRatio=30)时，堆内存会开始增加，一直增加到-Xmx的大小。

-Xmx堆内存的最大值，默认值是总内存/64（且小于1G），如果Xms和Xmx都不设置，则两者大小会相同，默认情况下，当堆中可用内存大于70%时，堆内存会开始减少，一直减小到-Xms的大小；

整个堆的大小=年轻代大小+年老代大小，堆的大小不包含持久代大小，如果增大了年轻代，年老代相应就会减小，官方默认的配置为年老代大小/年轻代大小=2/1左右；

建议在开发测试环境可以用Xms和Xmx分别设置最小值最大值，但是在线上生产环境，Xms和Xmx设置的值必须一样，原因与年轻代一样——防止抖动；

-Xss每个线程的栈内存，默认1M，一般来说是不需要改的。

-Xrs减少JVM对操作系统信号的使用。

-Xprof跟踪正运行的程序，并将跟踪数据在标准输出输出；适合于开发环境调试。

-Xnoclassgc关闭针对class的gc功能；因为其阻止内存回收，所以可能会导致OutOfMemoryError错误，慎用；

-Xincgc开启增量gc（默认为关闭）；这有助于减少长时间GC时应用程序出现的停顿；但由于可能和应用程序并发执行，所以会降低CPU对应用的处理能力。

-Xloggc:file与-verbose:gc功能类似，只是将每次GC事件的相关情况记录到一个文件中，文件的位置最好在本地，以避免网络的潜在问题。

3 不稳定参数

这是我们日常开发中接触到最多的参数类型。这也是非标准化参数，相对来说不稳定，随着JVM版本的变化可能会发生变化，主要用于**JVM调优**和debug。

不稳定参数以-XX 开头，此类参数的设置很容易引起JVM 性能上的差异，使JVM存在极大的不稳定性。如果此类参数设置合理将大大提高JVM的性能及稳定性。

不稳定参数分为三类：

性能参数：用于JVM的性能调优和内存分配控制，如内存大小的设置；

行为参数：用于改变JVM的基础行为，如GC的方式和算法的选择；

调试参数：用于监控、打印、输出jvm的信息；

不稳定参数语法规则：

布尔类型参数值：

- -XX:+
- -XX:-

示例：-XX:+UseG1GC：表示启用G1垃圾收集器

数字类型参数值：

- -XX:

示例：-XX:MaxGCPauseMillis=500 ： 表示设置GC的最大停顿时间是500ms

字符串类型参数值：

- -XX:

示例：-XX:HeapDumpPath=./dump.core

4 常用参数

如以下参数示例

```
 –Xms4g –Xmx4g –Xmn1200m –Xss512k –XX:NewRatio=4 –XX:SurvivorRatio=8 –
XX:PermSize=100m –XX:MaxPermSize=256m –XX:MaxTenuringThreshold=15 –
XX:MaxDirectMemorySize=1G –XX:+DisableExplicitGC
```

上面为Java7及以前版本的示例，在Java8中永久代的参数-XX:PermSize和-XX:MaxPermSize已经失效。这在前面章节中已经讲到。

参数解析：

- -Xms4g：初始化堆内存大小为4GB，ms是memory start的简称，等价于-XX:InitialHeapSize。
- -Xmx4g：堆内存最大值为4GB，mx是memory max的简称，等价于-XX:MaxHeapSize。
- -Xmn1200m：设置年轻代大小为1200MB。增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。
- -Xss512k：设置每个线程的堆栈大小。JDK5.0以后每个线程堆栈大小为1MB，以前每个线程堆栈大小为256K。应根据应用线程所需内存大小进行调整。在相同物理内存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在3000~5000左右。
- -XX:NewRatio=4：设置年轻代（包括Eden和两个Survivor区）与年老代的比值（除去持久代）。设置为4，则年轻代与年老代所占比值为1：4，年轻代占整个堆栈的1/5
- -XX:SurvivorRatio=8：设置年轻代中Eden区与Survivor区的大小比值。设置为8，则两个Survivor区与一个Eden区的比值为2:8，一个Survivor区占整个年轻代的1/10
- -XX:PermSize=100m：初始化永久代大小为100MB。
- -XX:MaxPermSize=256m：设置持久代大小为256MB。
- -XX:MaxTenuringThreshold=15：设置垃圾最大年龄。如果设置为0的话，则年轻代对象不经过Survivor区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在Survivor区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。
- -XX:MaxDirectMemorySize=1G：直接内存。报java.lang.OutOfMemoryError: Direct buffer memory异常可以上调这个值。
- -XX:+DisableExplicitGC：禁止运行期显式地调用System.gc()来触发fulll GC。

  注意: Java RMI的定时GC触发机制可通过配置-Dsun.rmi.dgc.server.gcInterval=86400来控制触发的时间。