JVM启动参数

官方文档：https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html
推荐配置说明：http://calvin1978.blogcn.com/?p=1602

重要的参数

Xms
英文解释：Initial heap size(in bytes)
中文释义：堆区初始值
使用方法：-Xms2g 或 -XX:InitialHeapSize=2048m
Xmx
英文解释：Maximum heap size(in bytes)
中文释义：堆区最大值
使用方法：-Xmx2g 或 -XX:MaxHeapSize=2048m
Xmn
英文解释：Maximum new generation size(in bytes)
中文释义：新生代最大值
使用方法：-Xmn512m 或 -XX:MaxNewSize=512m
PermSize
英文解释：Initial size of permanent generation(in bytes)
中文释义：永久代初始值
使用方法：-XX:PermSize=128m
MaxPermSize
英文解释：Maximum size of permanent generation(in bytes)
中文释义：永久代最大值
使用方法：-XX:MaxPermSize=256m
Xss
英文解释：Thread Stack Size(in Kbytes)
中文释义：线程栈最大值
使用方法：-Xss256k 或 -XX:ThreadStackSize=256k

GC策略相关

通过这些参数可以对JVM的GC性能进行调优
SurvivorRatio
英文解释：Rato of eden/survivor space size
中文释义：eden区和survivor的比值，新生代分为1个eden区和2个survivor区（s1,s2）
使用方法：-XX:SurvivorRatio=6
使用经验：假如设为6，则表示每个survivor区跟eden区的比值为1:6,每个survivor区占新生代的1/8
假如设为3，则表示每个suvivor区和eden区的比例是1:3，每个survivor区占新生代的1/5
PretenureSizeThreshold
英文解释：Maximum size in bytes of objects allocated in DefNew generation;zero means no

maximum
中文释义：可以在新生代直接分配的对象最大值，0表示没有最大值
使用方法：-XX:PretenureSizeThreshold=1000000
使用经验：设置该参数，可以使大于这个值的对象直接在老年代分配，避免在Eden区和Survivor区发生大量的内存复制，该参数只对Serial和ParNew收集器有效，Parallel Scavenge并不认识该参数

MaxTenuringThreshold
英文解释：Maximum value fo tenuring threshold
中文释义：年轻代最大年龄
使用方法：-XX:MaxTenuringThreshold=10
使用经验：每个对象在坚持过一次Minor GC之后，年龄就增加1，当超过这个参数值时就进入老年代，最大支持15

UseSerialGC
英文解释：Use the Serial garbage collector
中文释义：年轻代使用Serial垃圾收集器
使用方法：
开启 -XX:+UseSerialGC
关闭 -XX:-UseSerialGC
使用经验：不推荐使用，性能太差，老年代将会使用SerialOld垃圾收集器

UseParNewGC
英文解释：Use parallel threads in the new generation
中文释义：年轻代使用ParNew垃圾收集器
使用方法：
开启 -XX:+UseParNewGC
关闭 -XX:-UseParNewGC

ParallelGCThreads
英文解释：Number of parallel threads parallel gc will use
中文释义：并行执行gc的线程数
使用方法：-XX:ParallelGCThreads=16

UseParallelGC
英文解释：Use the Parallel Scavenge garbage collector
中文释义：年轻代使用Parallel Scavenge垃圾收集器
使用方法：
开启 -XX:+UseParallelGC
关闭 -XX:-UseParallelGC
使用经验：Linux下1.6,1.7,1.8默认开启，老年代将会使用SerialOld垃圾收集器

ParallelOldGC
英文解释：Use the Parallel Old garbage collector
中文释义：年轻代使用Parallel Scavenge收集器
使用方法：
开启 -XX:+ParallelOldGC
关闭 -XX:-ParallelOldGC
使用经验：老年代将会使用Parallel Old收集器

UseConcMarkSweepGC
英文解释：Use Concurrent Mark-Sweep GC in the old generation
中文释义：老年代使用CMS收集器（如果出现"Concurrent Mode Failure"，会使用SerialOld收集器）
使用方法：

开启 -XX:+UseConcMarkSweepGC
关闭 -XX:-UseConcMarkSweepGC
使用经验：年轻代将会使用ParNew收集器

CMSInitiatingOccupancyFraction

英文解释：Percentage CMS generation occupancy to start a CMS collection cycle. A negative value means that CMSTriggerRatio is used
中文释义：触发执行CMS回收的当前年代区内存占用的百分比，负值表示使用CMSTriggerRatio设置的值
使用方法：-XX:+CMSInitiatingOccupancyFraction=75

UseCMSInitiatingOccupancyOnly

英文解释：Only use occupancy as a criterion for staring a CMS collection
中文释义：只根据占用情况作为开始执行CMS收集的标准
使用方法：
开启 -XX:+UseCMSInitiatingOccupancyOnly
关闭 -XX:-UseCMSInitiatingOccupancyOnly

UseCMSCompactAtFullCollection

英文解释：Use Mark-Sweep-Compact algorithm at full collections
中文释义：使用CMS执行Full GC时对内存进行压缩
使用方法：
开启 -XX:+UseCMSCompactAtFullCollection
关闭 -XX:-UseCMSCompactAtFullCollection

CMSFullGCsBeforeCompaction

英文解释：Number of CMS full collection done before compaction if > 0
中文释义：多少次FGC后进行内存压缩
使用方法：-XX:CMSFullGCsBeforeCompaction=1

CMSClassUnloadingEnabled

英文解释：Whether class unloading enabled when using CMS GC
中文释义：当使用CMS GC时是否启用类卸载功能
使用方法：
开启 -XX:+CMSClassUnloadingEnabled
关闭 -XX:-CMSClassUnloadingEnabled

CMSParallelRemarkEnabled

英文解释：Whether parallel remark enabled (only if ParNewGC)
中文释义：是否启用并行标记（仅限于ParNewGC）
使用方法：
开启 -XX:+CMSParallelRemarkEnabled
关闭 -XX:-CMSParallelRemarkEnabled

DisableExplicitGC

英文解释：Ignore calls to System.gc()
中文释义：禁用System.gc()触发FullGC
使用方法：
开启 -XX:+DisableExplicitGC
关闭 -XX:-DisableExplicitGC
PS:不建议开启，如果开启了这个参数可能会导致对外内存无法及时回收造成对外内存溢出

GC日志相关

通过这些参数可以对JVM的GC日志输出进行配置，方便分析

Xloggc

英文解释：GC log file

中文释义：GC日志文件路径

使用方法：-Xloggc:/data/gclog/gc.log

UseGCLogFileRotation

英文解释：Rotate gclog files(for long running applications). It requires -Xloggc:

中文释义：滚动GC日志文件，须配置Xloggc

使用方法：

开启 -XX:+UseGCLogFileRotation

关闭 -XX:-UseGCLogFileRotation

NumberOfGCLogFiles

英文解释：Number of gclog files in rotation(default:0,no rotation)

中文释义：滚动GC日志文件数，默认0，不滚动

使用方法：-XX:NumberOfGCLogFiles=4

GCLogFileSize

英文解释：GC log file size,requires UseGCLogFileRotation. Set to 0 to only trigger rotation via jcmd

中文释义：GC文件滚动大小，需配置UseGCLogFileRotation，设置为0表示仅通过jcmd命令触发

使用方法：-XX:GCLogFileSize=100k

PrintGCDetails

英文解释：Print more details at garbage collection

中文释义：GC时打印更多详细信息

使用方法：

开启 -XX:+PrintGCDetails

关闭 -XX:-PrintGCDetails

可以通过jinfo -flag [+|-]PrintGCDetails 或 jinfo -flag PrintGCDetails= 来动态开启或设置值

PrintGCDateStamps

英文解释：Print date stamps at garbage collection

中文释义：GC时打印时间戳信息

使用方法：

开启 -XX:+PrintGCDateStamps

关闭 -XX:-PrintGCDateStamps

可以通过jinfo -flag [+|-]PrintGCDateStamps 或 jinfo -flag PrintGCDateStamps= 来动态开启或设置值

PrintTenuringDistribution

英文解释：Print tenuring age information

中文释义：打印存活实例年龄信息

使用方法：

开启 -XX:+PrintTenuringDistribution

关闭 -XX:-PrintTenuringDistribution

PrintGCApplicationStoppedTime

英文解释：Print the time of application has been stopped

中文释义：打印应用暂停时间

使用方法：

开启 -XX:+PrintGCApplicationStoppedTime
关闭 -XX:-PrintGCApplicationStoppedTime
PrintHeapAtGC
英文解释：Print heap layout before and after each GC
中文释义：GC前后打印堆区使用信息
使用方法：
开启 -XX:+PrintHeapAtGC
关闭 -XX:-PrintHeapAtGC
异常相关
通过这些参数可以在JVM异常情况下执行某些操作，以保留现场做分析用
HeapDumpOnOutOfMemoryError
英文解释：Dump heap to file when java.lang.OutOfMemoryError is thrown
中文释义：抛出内存溢出错误时导出堆信息到指定文件
使用方法：
开启 -XX:+HeapDumpOnOutOfMemoryError
关闭 -XX:-HeapDumpOnOutOfMemoryError
可以通过jinfo -flag [+|-]HeapDumpOnOutOfMemoryError 或 jinfo -flag HeapDumpOnOutOfMemoryError= 来动态开启或设置值
HeapDumpPath
英文解释：When HeapDumpOnOutOfMemoryError is on, the path(filename or directory) of the dump file(defaults to java_pid.hprof in the working directory)
中文释义：当HeapDumpOnOutOfMemoryError开启的时候，dump文件的保存路径，默认为工作目录下的java_pid.hprof文件
使用方法：-XX:HeapDumpPath=/data/dump/jvm.dump

其他

server
英文解释：server mode
中文释义：服务端模式
使用方法：-server
TieredCompilation
英文解释：Enable tiered compilation
中文释义：启用多层编译
使用方法：
开启 -XX:+TieredCompilation
关闭 -XX:-TieredCompilation

JVM启动推荐配置

```
#!/bin/bash

# 使用指南:
# 1. 修改本文件中的LOGDIR 和 APPID变量
# 2. 根据实际情况需求，反注释掉一些参数。
# 3. 修改应用启动脚本，增加 "source ./jvm-options.sh"，或者将本文件内容复制进应用启动脚本里.
```

```
# 4. 修改应用启动脚本，使用输出的JAVA_OTPS变量，如java -jar xxx的应用启动语句，修改为
java $JAVA_OPTS -jar xxx。


# change the jvm error log and backup gc log dir here
LOGDIR="./logs"

# change the appid for gc log name here
APPID="myapp"

JAVA_VERSION=$(java -version 2>&1 | awk -F '"' '/version/ {print $2}')


# Enable coredump
ulimit -c unlimited

## Memory Options##

MEM_OPTS="-Xms4g -Xmx4g -XX:NewRatio=1"

if [[ "$JAVA_VERSION" < "1.8" ]]; then
  MEM_OPTS="$MEM_OPTS -XX:PermSize=128m -XX:MaxPermSize=512m"
else
  MEM_OPTS="$MEM_OPTS -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m"
fi

# 启动时预申请内存
MEM_OPTS="$MEM_OPTS -XX:+AlwaysPreTouch"

# 如果线程数较多，函数的递归较少，线程栈内存可以调小节约内存，默认1M。
#MEM_OPTS="$MEM_OPTS -Xss256k"

# 堆外内存的最大值默认约等于堆大小，可以显式将其设小，获得一个比较清晰的内存总量预估
#MEM_OPTS="$MEM_OPTS -XX:MaxDirectMemorySize=2g"

# 根据JMX/VJTop的观察，调整二进制代码区大小避免满了之后不能再JIT，JDK7/8，是否打开多层编译
的默认值都不一样
#MEM_OPTS="$MEM_OPTS -XX:ReservedCodeCacheSize=240M"


## GC Options##

GC_OPTS="-XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -
XX:+UseCMSInitiatingOccupancyOnly"

# System.gc() 使用CMS算法
GC_OPTS="$GC_OPTS -XX:+ExplicitGCInvokesConcurrent"

# CMS中的下列阶段并发执行
```

```
GC_OPTS="$GC_OPTS -XX:+ParallelRefProcEnabled -
XX:+CMSParallelInitialMarkEnabled"

# 根据应用的对象生命周期设定，减少事实上的老生代对象在新生代停留时间，加快YGC速度
GC_OPTS="$GC_OPTS -XX:MaxTenuringThreshold=3"

# 如果OldGen较大，加大YGC时扫描OldGen关联的卡片，加快YGC速度，默认值256较低
GC_OPTS="$GC_OPTS -XX:+UnlockDiagnosticVMOptions -
XX:ParGCCardsPerStrideChunk=1024"

# 如果JVM并不独占机器，机器上有其他较繁忙的进程在运行，将GC线程数设置得比默认值(CPU核数*5/8
)更低以减少竞争，反而会大大加快YGC速度。
# 另建议CMS GC线程数简单改为YGC线程数一半.
#GC_OPTS="$GC_OPTS -XX:ParallelGCThreads=12 -XX:ConcGCThreads=6"

# 如果CMS GC时间很长，并且明显受新生代存活对象数量影响时打开，但会导致每次CMS GC与一次YGC连
在一起执行，加大了事实上JVM停顿的时间。
#GC_OPTS="$GC_OPTS -XX:+CMSScavengeBeforeRemark"

# 如果永久代使用不会增长，关闭CMS时ClassUnloading，降低CMS GC时出现缓慢的几率
#if [[ "$JAVA_VERSION" > "1.8" ]]; then
#  GC_OPTS="$GC_OPTS -XX:-CMSClassUnloadingEnabled"
#fi


## GC log Options, only for JDK7/JDK8 ##

# 默认使用/dev/shm 内存文件系统避免在高IO场景下写GC日志时被阻塞导致STW时间延长
if [ -d /dev/shm/ ]; then
    GC_LOG_FILE=/dev/shm/gc-${APPID}.log
else
  GC_LOG_FILE=${LOGDIR}/gc-${APPID}.log
fi


if [ -f ${GC_LOG_FILE} ]; then
  GC_LOG_BACKUP=${LOGDIR}/gc-${APPID}-$(date +'%Y%m%d_%H%M%S').log
  echo "saving gc log ${GC_LOG_FILE} to ${GC_LOG_BACKUP}"
  mv ${GC_LOG_FILE} ${GC_LOG_BACKUP}
fi

#打印GC日志，包括时间戳，晋升老生代失败原因，应用实际停顿时间(含GC及其他原因)
GCLOG_OPTS="-Xloggc:${GC_LOG_FILE} -XX:+PrintGCDetails -XX:+PrintGCDateStamps -
XX:+PrintPromotionFailure -XX:+PrintGCApplicationStoppedTime"


#打印GC原因，JDK8默认打开
if [[ "$JAVA_VERSION" < "1.8" ]]; then
  GCLOG_OPTS="$GCLOG_OPTS -XX:+PrintGCCause"
```

```
fi


# 打印GC前后的各代大小
#GCLOG_OPTS="$GCLOG_OPTS -XX:+PrintHeapAtGC"

# 打印存活区每段年龄的大小
#GCLOG_OPTS="$GCLOG_OPTS -XX:+PrintTenuringDistribution"

# 如果发生晋升失败，观察老生代的碎片
#GCLOG_OPTS="$GCLOG_OPTS -XX:+UnlockDiagnosticVMOptions -
XX:PrintFLSStatistics=2"


# 打印安全点日志，找出GC日志里非GC的停顿的原因
#GCLOG_OPTS="$GCLOG_OPTS -XX:+PrintSafepointStatistics -
XX:PrintSafepointStatisticsCount=1 -XX:+UnlockDiagnosticVMOptions -XX:-
DisplayVMOutput -XX:+LogVMOutput -XX:LogFile=/dev/shm/vm-${APPID}.log"


## Optimization Options##

OPTIMIZE_OPTS="-XX:-UseBiasedLocking -XX:AutoBoxCacheMax=20000 -
Djava.security.egd=file:/dev/./urandom"


# 关闭PerfData写入，避免高IO场景GC时因为写PerfData文件被阻塞，但会使得jstats，jps不能使
用。
#OPTIMIZE_OPTS="$OPTIMIZE_OPTS -XX:+PerfDisableSharedMem"

# 关闭多层编译，减少应用刚启动时的JIT导致的可能超时，以及避免部分函数C1编译后最终没被C2编译。
但导致函数没有被初始C1编译。
#if [[ "$JAVA_VERSION" > "1.8" ]]; then
#  OPTIMIZE_OPTS="$OPTIMIZE_OPTS -XX:-TieredCompilation"
#fi

# 如果希望无论函数的热度如何，最终JIT所有函数，关闭GC时将函数调用次数减半。
#OPTIMIZE_OPTS="$OPTIMIZE_OPTS -XX:-UseCounterDecay"

## Trouble shooting Options##

SHOOTING_OPTS="-XX:+PrintCommandLineFlags -XX:-OmitStackTraceInFastThrow -
XX:ErrorFile=${LOGDIR}/hs_err_%p.log"


# OOM 时进行HeapDump，但此时会产生较高的连续IO，如果是容器环境，有可能会影响他的容器
#SHOOTING_OPTS="$SHOOTING_OPTS -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=${LOGDIR}/"
```

```
# async-profiler 火焰图效果更好的参数
#SHOOTING_OPTS="$SHOOTING_OPTS -XX:+UnlockDiagnosticVMOptions -
XX:+DebugNonSafepoints"

# 在非生产环境，打开JFR进行性能记录（生产环境要收License的哈）
#SHOOTING_OPTS="$SHOOTING_OPTS -XX:+UnlockCommercialFeatures -
XX:+FlightRecorder -XX:+UnlockDiagnosticVMOptions -XX:+DebugNonSafepoints"



## JMX Options##

#开放JMX本地访问，设定端口号
JMX_OPTS="-Djava.rmi.server.hostname=127.0.0.1 -
Dcom.sun.management.jmxremote.port=7001 -Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false"


## Other Options##

OTHER_OPTS="-Djava.net.preferIPv4Stack=true -Dfile.encoding=UTF-8"


## All together ##

export JAVA_OPTS="$MEM_OPTS $GC_OPTS $GCLOG_OPTS $OPTIMIZE_OPTS $SHOOTING_OPTS
$JMX_OPTS $OTHER_OPTS"

echo JAVA_OPTS=$JAVA_OPTS
```

4核8G的普通云服务器推荐配置

```
java -Xms4g -Xmx4g -Xmn2g -XX:NewRatio=1 -XX:MaxDirectMemorySize=512m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=512m -XX:+UseConcMarkSweepGC -
XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly -
XX:+CMSParallelInitialMarkEnabled -XX:MaxTenuringThreshold=3-
XX:+ExplicitGCInvokesConcurrent -XX:+ParallelRefProcEnabled -
XX:+PerfDisableSharedMem -XX:+AlwaysPreTouch -XX:-OmitStackTraceInFastThrow -
XX:-UseBiasedLocking-XX:AutoBoxCacheMax=10240-XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/home/admin/logs/ -Xloggc:/home/admin/logs/gc.log -
XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintPromotionFailure -
XX:+PrintGCApplicationStoppedTime -Djava.security.egd=file:/dev/./urandom -
Djava.net.preferIPv4Stack=true -jar /home/admin/app/app.jar
```