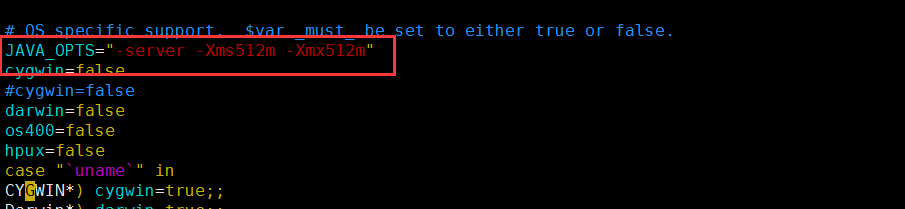
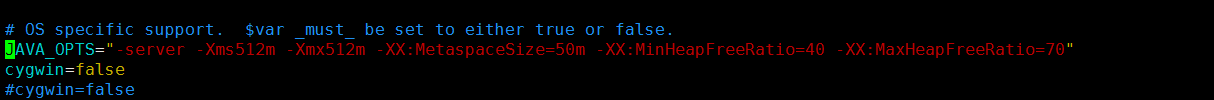
# Tomcat堆栈配置

## 自测tomcat jvm配置

修改tomcat目录/bin/catalina.sh 文件





添加jvm配置

JAVA\_OPTS="-server -Xms512m -Xmx512m"

vim光标定位

G 移至行行首

nG 移至第n行行首

n+ 移n行行首

n- 移n行行首

n$ 移n行(1表示本行)行尾

0 所行行首

$ 所行行尾

^ 所行首字母

h,j,k,l 左移移移右移

H 前屏幕首行行首

M 屏幕显示文件间行行首

L 前屏幕底行行首

dd删除一行

## 部分参说明：

-server:一定要作为第一个参数，在多个CPU时性能佳

-Xms：初始Heap大小，使用的最小内存,cpu性能高时此值应设的大一些

-Xmx：Java heap最大值，使用的最大内存

-XX:PermSize:设定内存的永久保存区域

-XX:MaxPermSize:设定最大内存的永久保存区域

-XX:MaxNewSize:

+XX:AggressiveHeap 会使得 Xms没有意义。这个参数让jvm忽略Xmx参数,疯狂地吃完一个G物理内存,再吃尽一个G的swap。

-Xss：每个线程的Stack大小

-verbose:gc 现实垃圾收集信息

-Xloggc:gc.log 指定垃圾收集日志文件

-Xmn：young generation的heap大小，一般设置为Xmx的3、4分之一

-XX:+UseParNewGC ：缩短minor收集的时间

-XX:+UseConcMarkSweepGC ：缩短major收集的时间

提示：此选项在Heap Size 比较大而且Major收集时间较长的情况下使用更合适。

tomcat 的jvm 内存溢出问题的解决

1、首先是：java.lang.OutOfMemoryError: Java heap space

解释：

JVM堆的设置是指java程序运行过程中JVM可以调配使用的内存空间的设置.JVM在启动的时候会自动设置Heap size的值，其初始空间(即-Xms)是物理内存的1/64，最大空间(-Xmx)是物理内存的1/4。可以利用JVM提供的-Xmn -Xms -Xmx等选项可进行设置。Heap size 的大小是Young Generation 和Tenured Generaion 之和。

提示：在JVM中如果98％的时间是用于GC且可用的Heap size 不足2％的时候将抛出此异常信息。

提示：Heap Size 最大不要超过可用物理内存的80％，一般的要将-Xms和-Xmx选项设置为相同，而-Xmn为1/4的-Xmx值。

解决方法：win下

手动设置Heap size

修改TOMCAT\_HOME/bin/catalina.bat，在“echo ”Using CATALINA\_BASE: $CATALINA\_BASE””上面加入以下行：

set JAVA\_OPTS=%JAVA\_OPTS% -server -Xms800m -Xmx800m -XX:MaxNewSize=256m

2、其次是：java.lang.OutOfMemoryError: PermGen space

原因：

PermGen space的全称是Permanent Generation space,是指内存的永久保存区域，这块内存主要是被JVM存放Class和Meta信息的,Class在被Loader时就会被放到PermGen space中，它和存放类实例(Instance)的Heap区域不同,GC(Garbage Collection)不会在主程序运行期对PermGen space进行清理，所以如果你的应用中有很CLASS的话,就很可能出现PermGen space错误，这种错误常见在web服务器对JSP进行pre compile的时候。如果你的WEB APP下都用了大量的第三方jar, 其大小超过了jvm默认的大小(4M)那么就会产生此错误信息了。

解决方法：win下

手动设置MaxPermSize大小

修改TOMCAT\_HOME/bin/catalina.bat（Linux下为catalina.sh），在“echo ”Using CATALINA\_BASE: $CATALINA\_BASE””上面加入以下行：

set JAVA\_OPTS=%JAVA\_OPTS% -server -XX:PermSize=128M -XX:MaxPermSize=512m

catalina.sh加set JAVA\_OPTS='-Xms64 -Xmx512'无效解决方案

set JAVA\_OPTS='-Xms64 -Xmx512'

上面无效的话这样写：

declare -x JAVA\_OPTS="-Xms128m -Xmx256"

"declare -x"一定要加，不然就会报-x指令无效，

还有后面的引号也要注意加上 wq后，重启tomcat即可，ps ax后看到tomcat的进程会变成类似

/opt/jdk1.6.0/bin/java -Xms128m -Xmx256m -Djava.endorsed.dirs=/opt/tomcat/common/e

在windows下，如果使用控制台的方式，也就是直接执行startup.bat方式启动，可以通过修改catalina.bat

set JAVA\_OPTS=-Xms256m -Xmx512m

注意，linux下有单引号，win下不用，如果加上单引号，tomcat根本起不来。

如下一些问题

java -Xmx1610M -version

java -Xmx1610m -version

## 网摘的tomcat内存溢出解决方案

Tomcat内存溢出的原因

　　在生产环境中tomcat内存设置不好很容易出现内存溢出。造成内存溢出是不一样的，当然处理方式也不一样。

　　这里根据平时遇到的情况和相关资料进行一个总结。常见的一般会有下面三种情况：

　　1.OutOfMemoryError： Java heap space

　　2.OutOfMemoryError： PermGen space

　　3.OutOfMemoryError： unable to create new native thread.

　　Tomcat内存溢出解决方案

　　对于前两种情况，在应用本身没有内存泄露的情况下可以用设置tomcat jvm参数来解决。（-Xms -Xmx -XX：PermSize -XX：MaxPermSize）

　　最后一种可能需要调整操作系统和tomcat jvm参数同时调整才能达到目的。

　　第一种：是堆溢出。

　　原因分析：

JVM堆的设置是指java程序运行过程中JVM可以调配使用的内存空间的设置.JVM在启动的时候会自动设置Heap size的值，其初始空间(即-Xms)是物理内存的1/64，最大空间(-Xmx)是物理内存的1/4。可以利用JVM提供的-Xmn -Xms -Xmx等选项可进行设置。Heap size 的大小是Young Generation 和Tenured Generaion 之和。

在JVM中如果98％的时间是用于GC且可用的Heap size 不足2％的时候将抛出此异常信息。

Heap Size 最大不要超过可用物理内存的80％，一般的要将-Xms和-Xmx选项设置为相同，而-Xmn为1/4的-Xmx值。

　　没有内存泄露的情况下，调整-Xms -Xmx参数可以解决。

　　-Xms：初始堆大小

　　-Xmx：最大堆大小

　　但堆的大小受下面三方面影响：

　　1.相关操作系统的数据模型（32-bt还是64-bit）限制；（32位系统下，一般限制在1.5G~2G；我在2003 server 系统下（物理内存：4G和6G，jdk：1.6）测试 1612M，64位操作系统对内存无限制。）

　　2.系统的可用虚拟内存限制；

　　3.系统的可用物理内存限制。

　　堆的大小可以使用 java -Xmx\*\*\*M version 命令来测试。支持的话会出现jdk的版本号，不支持会报错。

　　-Xms -Xmx一般配置成一样比较好比如set JAVA\_OPTS= -Xms1024m -Xmx1024m

其初始空间(即-Xms)是物理内存的1/64，最大空间(-Xmx)是物理内存的1/4。可以利用JVM提供的-Xmn -Xms -Xmx等选项可

进行设置

实例，以下给出1G内存环境下java jvm 的参数设置参考：

JAVA\_OPTS="-server -Xms800m -Xmx800m -XX:PermSize=64M -XX:MaxNewSize=256m -XX:MaxPermSize=128m -Djava.awt.headless=true "

JAVA\_OPTS="-server -Xms768m -Xmx768m -XX:PermSize=128m -XX:MaxPermSize=256m -XX:

NewSize=192m -XX:MaxNewSize=384m"

CATALINA\_OPTS="-server -Xms768m -Xmx768m -XX:PermSize=128m -XX:MaxPermSize=256m

-XX:NewSize=192m -XX:MaxNewSize=384m"

服务器为1G内存：JAVA\_OPTS="-server -Xms800m -Xmx800m -XX:PermSize=64M -XX:MaxNewSize=256m -XX:MaxPermSize=128m -Djava.awt.headless=true "

服务器为64位、2G内存: JAVA\_OPTS='-server -Xms1024m -Xmx1536m -XX:PermSize=128M -XX:MaxNewSize=256m -XX:MaxPermSize=256m'

-------------------解决方案1：-----------------------------

前提：是执行startup.bat启动tomcat的方式

Linux服务器：

在/usr/local/apache-tomcat-5.5.23/bin 目录下的catalina.sh

添加：JAVA\_OPTS='-Xms512m -Xmx1024m'

或者 JAVA\_OPTS="-server -Xms800m -Xmx800m -XX:MaxNewSize=256m"

或者 CATALINA\_OPTS="-server -Xms256m -Xmx300m"

Windows服务器：

在catalina.bat最前面加入

set JAVA\_OPTS=-Xms128m -Xmx350m

或者set CATALINA\_OPTS=-Xmx300M -Xms256M

（区别是一个直接设置jvm内存，另一个设置tomcat内存，CATALINA\_OPTS似乎可以与JAVA\_OPTS不加区别的使用）

基本参数说明

-client，-server

这两个参数用于设置虚拟机使用何种运行模式，一定要作为第一个参数，client模式启动比较快，但运行时性能和内存管理效率不如server模式，通常用于客户端应用程序。相反，server模式启动比client慢，但可获得更高的运行性能。

在windows上，缺省的虚拟机类型为client模式，如果要使用server模式，就需要在启动虚拟机时加-server参数，以获得更高性能，对服务器端应用，推荐采用server模式，尤其是多个CPU的系统。在Linux，Solaris上缺省采用server模式。

此外，在多cup下，建议用server模式

-Xms<size>

设置虚拟机可用内存堆的初始大小，缺省单位为字节，该大小为1024的整数倍并且要大于1MB，可用k(K)或m(M)为单位来设置较大的内存数。初始堆大小为2MB。加“m”说明是MB，否则就是KB了。

例如：-Xms6400K，-Xms256M

-Xmx<size>

设置虚拟机的最大可用大小，缺省单位为字节。该值必须为1024整数倍，并且要大于2MB。可用k(K)或m(M)为单位来设置较大的内存数。缺省堆最大值为64MB。

例如：-Xmx81920K，-Xmx80M

当应用程序申请了大内存运行时虚拟机抛出java.lang.OutOfMemoryError: Java heap space错误，就需要使用-Xmx设置较大的可用内存堆。

PermSize/MaxPermSize：定义Perm段的尺寸，即永久保存区域的大小，PermSize为JVM启动时初始化Perm的内存大小；MaxPermSize为最大可占用的Perm内存大小。在用户生产环境上一般将这两个值设为相同，以减少运行期间系统在内存申请上所花的开销。

如果用startup.bat启动tomcat,OK设置生效.够成功的分配200M内存.

-------------------解决方案2：------------------------

前提：是执行startup.bat启动tomcat的方式

手动设置Heap size

Windows服务器：

修改TOMCAT\_HOME/bin/catalina.bat，在“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

Java代码

set JAVA\_OPTS=%JAVA\_OPTS% -server -Xms800m -Xmx800m -XX:MaxNewSize=256m

注：JAVA\_OPTS是保留先前设置。

Linux服务器：

修改TOMCAT\_HOME/bin/catalina.sh

在“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

JAVA\_OPTS="$JAVA\_OPTS -server -Xms800m -Xmx800m -XX:MaxNewSize=256m"

注：$JAVA\_OPTS是保留先前设置。

Linux服务器：

修改TOMCAT\_HOME/bin/catalina.sh

在“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

JAVA\_OPTS="$JAVA\_OPTS -server -Xms800m -Xmx800m -XX:MaxNewSize=256m"

注：$JAVA\_OPTS是保留先前设置。

-------------------解决方案3：-----------------------------

前提：是执行windows的系统服务启动tomcat的方式

但是如果不是执行startup.bat启动tomcat而是利用windows的系统服务启动tomcat服务,上面的设置就不生效了,

就是说set JAVA\_OPTS=-Xms128m -Xmx350m 没起作用.上面分配200M内存就OOM了..

windows服务执行的是bin\tomcat.exe.他读取注册表中的值,而不是catalina.bat的设置.

解决办法:

修改注册表HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Tomcat Service Manager\Tomcat5\Parameters\JavaOptions

原值为

-Dcatalina.home="C:\ApacheGroup\Tomcat 5.0"

-Djava.endorsed.dirs="C:\ApacheGroup\Tomcat 5.0\common\endorsed"

-Xrs

加入 -Xms300m -Xmx350m

重起tomcat服务,设置生效

-------------------解决方案4：-----------------------------

前提：是执行windows的系统服务启动tomcat的方式

在安裝tomcat時若有勾選"NT Service(NT/2000/XP only)"

則安裝完成後在安裝目錄的"bin"目錄裡會有一個tomcat.exe的檔案

先把tomcat的服務停掉

在命令列模式下（运行里输入CMD）

將目錄切換到tomcat的bin目錄

用下面的命令把服務移除

tomcat -uninstall "Apache Tomcat 4.1"

接下來，写个批处理。

內容如下

set SERVICENAME=Apache Tomcat 4.1

set CATALINA\_HOME=E:\Tomcat 4.1.24

set CLASSPATH=D:\j2sdk1.4.1\_01\lib

set JAVACLASSPATH=%CLASSPATH%

set JAVACLASSPATH=%JAVACLASSPATH%;?TALINA\_HOME%\bin\bootstrap.jar

set JAVACLASSPATH=%JAVACLASSPATH%;?TALINA\_HOME%\common\lib\servlet.jar

set JAVACLASSPATH=%JAVACLASSPATH%;%JAVA\_HOME%\lib\tools.jar

tomcat.exe -install "%SERVICENAME%" "%JAVA\_HOME%\jre\bin\server\jvm.dll" -Djava.class.path="%JAVACLASSPATH%" -Dcatalina.home="?TALINA\_HOME%" -Xms512m -Xmx768m -start org.apache.catalina.startup.Bootstrap -params start -stop org.apache.catalina.startup.Bootstrap -params stop -out "?TALINA\_HOME%\logs\stdout.log" -err"?TALINA\_HOME%\logs\stderr.log"

注意，从 tomcat.exe -install开始的是最后一行！不要手工回车换行把这一行分成了好几段。保存后在命令行下执行这个bat文件，注意执行的时候将“服务”窗口关闭。

第二种：永久保存区域溢出

　原因分析：

PermGen space的全称是Permanent Generation space,是指内存的永久保存区域，这块内存主要是被JVM存放Class和Meta信息的,Class在被Loader时就会被放到PermGen space中，它和存放类实例(Instance)的Heap区域不同,GC(Garbage Collection)不会在主程序运行期对PermGen space进行清理，所以如果你的应用中有很CLASS的话,就很可能出现PermGen space错误，这种错误常见在web服务器对JSP进行pre compile的时候。如果你的WEB APP下都用了大量的第三方jar, 其大小超过了jvm默认的大小(4M)那么就会产生此错误信息了。但目前的hibernate和spring项目中也很容易出现这样的问题。可能是由于这些框架会动态class，而且jvm的gc是不会清理PemGen space的，超过了jvm默认的大小(4M)，导致内存溢出。

　　建议：将相同的第三方jar文件移置到tomcat/shared/lib目录下，这样可以达到减少jar 文档重复占用内存的目的。

这一个一般是加大-XX：PermSize -XX：MaxPermSize 来解决问题。

　　-XX：PermSize 永久保存区域初始大小

　　-XX：PermSize 永久保存区域初始最大值

　　这一般结合第一条使用，比如set JAVA\_OPTS= -Xms1024m -Xmx1024m -XX：PermSize=128M -XX：PermSize=256M

　　有一点需要注意：java -Xmx\*\*\*M version 命令来测试的最大堆内存是 -Xmx与 -XX：PermSize的和比如系统支持最大的jvm堆大小事1.5G，那 -Xmx1024m -XX：PermSize=768M 是无法运行的。

-----------------解决方案1：-------------------------

Linux服务器：

在catalina.sh的第一行增加：

JAVA\_OPTS=

-Xms64m

-Xmx256m

-XX:PermSize=128M

-XX:MaxNewSize=256m

-XX:MaxPermSize=256m

或者

在“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

JAVA\_OPTS="-server -XX:PermSize=64M -XX:MaxPermSize=128m

Windows服务器：

在catalina.bat的第一行增加：

set JAVA\_OPTS=-Xms64m -Xmx256m -XX:PermSize=128M -XX:MaxNewSize=256m -XX:MaxPermSize=256m

-----------------解决方案2：------------------------

修改TOMCAT\_HOME/bin/catalina.bat（Linux下为catalina.sh），在Java代码

“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

set JAVA\_OPTS=%JAVA\_OPTS% -server -XX:PermSize=128M -XX:MaxPermSize=512m

“echo"Using CATALINA\_BASE: $CATALINA\_BASE"”上面加入以下行：

set JAVA\_OPTS=%JAVA\_OPTS% -server -XX:PermSize=128M -XX:MaxPermSize=512m

catalina.sh下为：

Java代码

JAVA\_OPTS="$JAVA\_OPTS -server -XX:PermSize=128M -XX:MaxPermSize=512m"

JAVA\_OPTS="$JAVA\_OPTS -server -XX:PermSize=128M -XX:MaxPermSize=512m"

　　第三种：无法创建新的线程。

　　这种现象比较少见，也比较奇怪，主要是和jvm与系统内存的比例有关。

　　这种怪事是因为JVM已经被系统分配了大量的内存（比如1.5G），并且它至少要占用可用内存的一半。有人发现，在线程个数很多的情况下，你分配给JVM的内存越多，那么，上述错误发生的可能性就越大。

　　原因分析

（从这个blog中了解到原因：http://hi.baidu.com/hexiong/blog/item/16dc9e518fb10c2542a75b3c.html）：

　　每一个32位的进程最多可以使用2G的可用内存，因为另外2G被操作系统保留。这里假设使用1.5G给JVM，那么还余下500M可用内存。这500M内存中的一部分必须用于系统dll的加载，那么真正剩下的也许只有400M，现在关键的地方出现了：当你使用Java创建一个线程，在JVM的内存里也会创建一个Thread对象，但是同时也会在操作系统里创建一个真正的物理线程（参考JVM规范），操作系统会在余下的 400兆内存里创建这个物理线程，而不是在JVM的1500M的内存堆里创建。在jdk1.4里头，默认的栈大小是256KB，但是在jdk1.5里头，默认的栈大小为1M每线程，因此，在余下400M的可用内存里边我们最多也只能创建400个可用线程。

这样结论就出来了，要想创建更多的线程，你必须减少分配给JVM的最大内存。还有一种做法是让JVM宿主在你的JNI代码里边。

　　给出一个有关能够创建线程的最大个数的估算公式：

　　（MaxProcessMemory - JVMMemory - ReservedOsMemory） / （ThreadStackSize） = Number of threads

　　对于jdk1.5而言，假设操作系统保留120M内存：

　　1.5GB JVM： （2GB-1.5Gb-120MB）/（1MB） = ~380 threads

　　1.0GB JVM： （2GB-1.0Gb-120MB）/（1MB） = ~880 threads

　　在2000/XP/2003的boot.ini里头有一个启动选项，好像是：/PAE /3G ，可以让用户进程最大内存扩充至3G，这时操作系统只能占用最多1G的虚存。那样应该可以让JVM创建更多的线程。

　　因此这种情况需要结合操作系统进行相关调整。

　　因此：我们需要结合不同情况对tomcat内存分配进行不同的诊断才能从根本上解决问题。

检测当前JVM内存使用情况：

System.out.println("JVM MAX MEMORY: " + Runtime.getRuntime().maxMemory()/1024/1024+"M");

System.out.println("JVM IS USING MEMORY:" + Runtime.getRuntime().totalMemory()/1024/1024+"M");

System.out.println("JVM IS FREE MEMORY:" + Runtime.getRuntime().freeMemory()/1024/1024+"M");

这三个方法都是说JVM的内存使用情况而不是操作系统的内存；

　　maxMemory()这个方法返回的是java虚拟机（这个进程）能构从操作系统那里挖到的最大的内存，以字节为单位，如果在运行java程序的时候，没有添加-Xmx参数，那么就是64兆，也就是说maxMemory()返回的大约是64\*1024\*1024字节，这是java虚拟机默认情况下能从操作系统那里挖到的最大的内存。如果添加了-Xmx参数，将以这个参数后面的值为准，例如java -cp ClassPath -Xmx512m ClassName，那么最大内存就是512\*1024\*0124字节。

　　totalMemory()这个方法返回的是java虚拟机现在已经从操作系统那里挖过来的内存大小，也就是java虚拟机这个进程当时所占用的所有内存。如果在运行java的时候没有添加-Xms参数，那么，在java程序运行的过程的，内存总是慢慢的从操作系统那里挖的，基本上是用多少挖多少，直挖到maxMemory()为止，所以totalMemory()是慢慢增大的。如果用了-Xms参数，程序在启动的时候就会无条件的从操作系统中挖-Xms后面定义的内存数，然后在这些内存用的差不多的时候，再去挖。

freeMemory()是什么呢，刚才讲到如果在运行java的时候没有添加-Xms参数，那么，在java程序运行的过程的，内存总是慢慢的从操作系统那里挖的，基本上是用多少挖多少，但是java虚拟机100％的情况下是会稍微多挖一点的，这些挖过来而又没有用上的内存，实际上就是freeMemory()，所以freeMemory()的值一般情况下都是很小的，但是如果你在运行java程序的时候使用了-Xms，这个时候因为程序在启动的时候就会无条件的从操作系统中挖-Xms后面定义的内存数，这个时候，挖过来的内存可能大部分没用上，所以这个时候freeMemory()可能会有些

--------------------解决方案--------------------------

JVM堆大小的调整

　　Sun HotSpot 1.4.1使用分代收集器，它把堆分为三个主要的域：新域、旧域以及永久域。Jvm生成的所有新对象放在新域中。一旦对象经历了一定数量的垃圾收集循环后，便获得使用期并进入旧域。在永久域中jvm则存储class和method对象。就配置而言，永久域是一个独立域并且不认为是堆的一部分。

　　下面介绍如何控制这些域的大小。可使用-Xms和-Xmx 控制整个堆的原始大小或最大值。

　　下面的命令是把初始大小设置为128M：

　　java –Xms128m

　　–Xmx256m为控制新域的大小，可使用-XX:NewRatio设置新域在堆中所占的比例。

　　下面的命令把整个堆设置成128m，新域比率设置成3，即新域与旧域比例为1：3，新域为堆的1/4或32M：

java –Xms128m –Xmx128m

–XX:NewRatio =3可使用-XX:NewSize和-XX:MaxNewsize设置新域的初始值和最大值。

　　下面的命令把新域的初始值和最大值设置成64m:

java –Xms256m –Xmx256m –Xmn64m

　　永久域默认大小为4m。运行程序时，jvm会调整永久域的大小以满足需要。每次调整时，jvm会对堆进行一次完全的垃圾收集。

　　使用-XX:MaxPerSize标志来增加永久域搭大小。在WebLogic Server应用程序加载较多类时，经常需要增加永久域的最大值。当jvm加载类时，永久域中的对象急剧增加，从而使jvm不断调整永久域大小。为了避免调整，可使用-XX:PerSize标志设置初始值。

　　下面把永久域初始值设置成32m，最大值设置成64m。

java -Xms512m -Xmx512m -Xmn128m -XX:PermSize=32m -XX:MaxPermSize=64m

默认状态下，HotSpot在新域中使用复制收集器。该域一般分为三个部分。第一部分为Eden，用于生成新的对象。另两部分称为救助空间，当Eden充满时，收集器停止应用程序，把所有可到达对象复制到当前的from救助空间，一旦当前的from救助空间充满，收集器则把可到达对象复制到当前的to救助空间。From和to救助空间互换角色。维持活动的对象将在救助空间不断复制，直到它们获得使用期并转入旧域。使用-XX:SurvivorRatio可控制新域子空间的大小。

　　同NewRation一样，SurvivorRation规定某救助域与Eden空间的比值。比如，以下命令把新域设置成64m，Eden占32m，每个救助域各占16m：

java -Xms256m -Xmx256m -Xmn64m -XX:SurvivorRation =2

　　如前所述，默认状态下HotSpot对新域使用复制收集器，对旧域使用标记－清除－压缩收集器。在新域中使用复制收集器有很多意义，因为应用程序生成的大部分对象是短寿命的。理想状态下，所有过渡对象在移出Eden空间时将被收集。如果能够这样的话，并且移出Eden空间的对象是长寿命的，那么理论上可以立即把它们移进旧域，避免在救助空间反复复制。但是，应用程序不能适合这种理想状态，因为它们有一小部分中长寿命的对象。最好是保持这些中长寿命的对象并放在新域中，因为复制小部分的对象总比压缩旧域廉价。为控制新域中对象的复制，可用-XX:TargetSurvivorRatio控制救助空间的比例（该值是设置救助空间的使用比例。如救助空间位1M，该值50表示可用500K）。该值是一个百分比，默认值是50。当较大的堆栈使用较低的sruvivorratio时，应增加该值到80至90，以更好利用救助空间。用-XX:maxtenuring threshold可控制上限。

　　为放置所有的复制全部发生以及希望对象从eden扩展到旧域，可以把MaxTenuring Threshold设置成0。设置完成后，实际上就不再使用救助空间了，因此应把SurvivorRatio设成最大值以最大化Eden空间，设置如下：

java … -XX:MaxTenuringThreshold=0 –XX:SurvivorRatio＝50000 …

垃圾回收描述：

垃圾回收分多级，0级为全部(Full)的垃圾回收，会回收OLD段中的垃圾；1级或以上为部分垃圾回收，只会回收Young中的垃圾，内存溢出通常发生于OLD段或Perm段垃圾回收后，仍然无内存空间容纳新的Java对象的情况。

当一个URL被访问时，内存申请过程如下：

A. JVM会试图为相关Java对象在Eden中初始化一块内存区域

B. 当Eden空间足够时，内存申请结束。否则到下一步

C. JVM试图释放在Eden中所有不活跃的对象（这属于1或更高级的垃圾回收）；释放后若Eden空间仍然不足以放入新对象，则试图将部分Eden中活跃对象放入Survivor区/OLD区

D. Survivor区被用来作为Eden及OLD的中间交换区域，当OLD区空间足够时，Survivor区的对象会被移到Old区，否则会被保留在Survivor区

E. 当OLD区空间不够时，JVM会在OLD区进行完全的垃圾收集（0级）

F. 完全垃圾收集后，若Survivor及OLD区仍然无法存放从Eden复制过来的部分对象，导致JVM无法在Eden区为新对象创建内存区域，则出现”out of memory错误”

Java堆相关参数：

ms/mx：定义YOUNG+OLD段的总尺寸，ms为JVM启动时YOUNG+OLD的内存大小；mx为最大可占用的YOUNG+OLD内存大小。在用户生产环境上一般将这两个值设为相同，以减少运行期间系统在内存申请上所花的开销。

NewSize/MaxNewSize：定义YOUNG段的尺寸，NewSize为JVM启动时YOUNG的内存大小；MaxNewSize为最大可占用的YOUNG内存大小。在用户生产环境上一般将这两个值设为相同，以减少运行期间系统在内存申请上所花的开销。

PermSize/MaxPermSize：定义Perm段的尺寸，PermSize为JVM启动时Perm的内存大小；MaxPermSize为最大可占用的Perm内存大小。在用户生产环境上一般将这两个值设为相同，以减少运行期间系统在内存申请上所花的开销。

SurvivorRatio：设置Survivor空间和Eden空间的比例

例：

MEM\_ARGS="-Xms512m -Xmx512m -XX:NewSize=256m -XX:MaxNewSize=256m -XX:PermSize=128m -XX:MaxPermSize=128m -XX:SurvivorRatio=6"

在上面的例子中：

YOUNG+OLD: 512M

YOUNG: 256M

Perm: 128M

Eden: YOUNG\*6/(6+1+1)=192M

Survivor: YOUNG/(6+1+1)=32M

Java堆的总尺寸=YOUNG+OLD+Perm=640M