JVM系列三:JVM參數設置、分析

不管是YGC還是Full GC,GC過程中都會對導致程序運行中中斷,正確的選擇不同的GC策略,調整JVM、GC的參數，可以極大的減少由於GC工作，而導致的程序運行中斷方面的問題，進而適當的提高Java程序的工作效率。但是調整GC是以個極為複雜的過程，由於各個程序具備不同的特點，如：web和GUI程序就有很大區別（Web可以適當的停頓，但GUI停頓是客戶無法接受的），而且由於跑在各個機器上的配置不同（主要cup個數，內存不同），所以使用的GC種類也會不同(如何選擇見GC種類及如何選擇)。本文將注重介紹JVM、GC的一些重要參數的設置來提高系統的性能。

JVM內存組成及GC相關內容請見之前的文章:JVM內存組成 GC策略&內存申請。

JVM參數的含義 實例見實例分析

參數名稱	含義	默認值	
-Xms	初始堆大小	物理內存的1/64(<1GB)	默認(MinHeapFreeRatio參數可以調整)空餘堆內存小於40%時，JVM就會增大堆直到-Xmx的最大限制.
-Xmx	最大堆大小	物理內存的1/4(<1GB)	默認(MaxHeapFreeRatio參數可以調整)空餘堆內存大於70%時，JVM會減少堆直到 -Xms的最小限制

-Xmn年輕代大小(1.4or lator)

ADVERTISEMENT

注意：此處的大小是（eden+ 2 survivor space).與jmap -heap中顯示的New gen是不同的。

整個堆大小=年輕代大小 + 年老代大小 + 持久代大小.

增大年輕代後,將會減小年老代大小.此值對系統性能影響較大,Sun官方推薦配置為整個堆的3/8

-XX:NewSize	設置年輕代大小(for 1.3/1.4)	
-XX:MaxNewSize	年輕代最大值(for 1.3/1.4)	
-XX:PermSize	設置持久代(perm gen)初始值	物理內存的1/64
-XX:MaxPermSize	設置持久代最大值	物理內存的1/4

-Xss每個線程的堆棧大小 JDK5.0以後每個線程堆棧大小為1M,以前每個線程堆棧大小為256K.更具應用的線程所需內存大小進行 調整.在相同物理內存下,減小這個值能生成更多的線程.但是作業系統對一個進程內的線程數還是有限制的,不能無限生成,經驗值在3000~5000左右

ADVERTISEMENT

一般小的應用， 如果棧不是很深， 應該是128k夠用的 大的應用建議使用256k。這個選項對性能影響比較大，需要嚴格的測試。（校長）

和threadstacksize選項解釋很類似,官方文檔似乎沒有解釋,在論壇中有這樣一句話:"」

-Xss is translated in a VM flag named ThreadStackSize」

一般設置這個值就可以了。(0 means use default stack size) [Sparc: 512; Solaris x86: 320 (was 256 prior in 5.0 and earlier); Sparc 64 bit: 1024; Linux amd64: 1024 (was 0 in 5.0 and earlier); all others 0.]-XX:NewRatio=4表示年輕代與年老代所占比值為1:4,年輕代占整個堆棧的1/5

Xms=Xmx並且設置了Xmn的情況下，該參數不需要進行設置。

-XX:SurvivorRatio	Eden區與Survivor區的大小比值	設置為8,則兩個Survivor區與一個Eden區的比值為2:8,一個Survivor區占整個年輕代的1/10
-XX:LargePageSizeInBytes	內存頁的大小不可設置過大， 會影響Perm的大小	=128m
-XX:+UseFastAccessorMethods	原始類型的快速優化	
-XX:+DisableExplicitGC	關閉System.gc	這個參數需要嚴格的測試

如果設置為0的話,則年輕代對象不經過Survivor區,直接進入年老代. 對於年老代比較多的應用,可以提高效率.如果將此值設置為一個較大值,則年輕代對象會在Survivor區進行多次複製,這樣可以增加對象再年輕代的存活 時間,增加在年輕代即被回收的機率

該參數只有在串行GC時才有效.

-XX:+AggressiveOpts	加快編譯		
-XX:+UseBiasedLocking	鎖機制的性能改善		
-Xnoclassgc	禁用垃圾回收		
-XX:SoftRefLRUPolicyMSPerMB	每兆堆空閒空間中SoftReference的存活時間	1s	softly reachable objects will remain alive for some amount of time after the last time they were referenced. The default value is one second of lifetime per free megabyte in the heap
-XX:+UseParallelGC	Full GC採用parallel MSC (此項待驗證)		選擇垃圾收集器為並行收集器.此配置僅對年輕代有效.即上述配置下,年輕代使用並發收集,而年老代仍舊使用串行收集.(此項待驗證)
-XX:+UseParNewGC	設置年輕代為並行收集		可與CMS收集同時使用 JDK5.0以上,JVM會根據系統配置自行設置,所以無需再設置此值
-XX:ParallelGCThreads	並行收集器的線程數		此值最好配置與處理器數目相等 同樣適用於CMS

-XX:+UseParallelOldGC	年老代垃圾收集方式為並行收集(Parallel Compacting)		這個是JAVA 6出現的參數選項
-XX:MaxGCPauseMillis	每次年輕代垃圾回收的最長時間(最大暫停時間)		如果無法滿足此時間,JVM會自動調整年輕代大小,以滿足此值.
-XX:+UseAdaptiveSizePolicy	自動選擇年輕代區大小和相應的Survivor區比例		設置此選項後,並行收集器會自動選擇年輕代區大小和相應的Survivor區比例,以達到目標系統規定的最低相應時間或者收集頻率等,此值建議使用並行收集器時,一直打開.
-XX:GCTimeRatio	設置垃圾回收時間占程序運行時間的百分比		公式為1/(1+n)
-XX:+ScavengeBeforeFullGC	Full GC前調用YGC	true	Do young generation GC prior to a full GC. (Introduced in 1.4.1.)
-XX:+UseConcMarkSweepGC	使用CMS內存收集		測試中配置這個以後,-XX:NewRatio=4的配置失效了,原因不明.所以,此時年輕代大小最好用-Xmn設置.???
-XX:+AggressiveHeap			試圖是使用大量的物理內存 長時間大內存使用的優化，能檢查計算資源（內存， 處理器數量） 至少需要256MB內存 大量的CPU／內存， （在1.4.1在4CPU的機器上已經顯示有提升）
-XX:CMSFullGCsBeforeCompaction	多少次後進行內存壓縮		由於並發收集器不對內存空間進行壓縮,整理,所以運行一段時間以後會產生"碎片",使得運行效率降低.此值設置運行多少次GC以後對內存空間進行壓縮,整理.
-XX:+CMSParallelRemarkEnabled	降低標記停頓		

-XX+UseCMSCompactAtFullCollection在FULL GC的時候， 對年老代的壓縮 CMS是不會移動內存的， 因此， 這個非常容易產生碎片， 導致內存不夠用， 因此， 內存的壓縮這個時候就會被啟用。 增加這個參數是個好習慣。

可能會影響性能,但是可以消除碎片

-XX:+UseCMSInitiatingOccupancyOnly	使用手動定義初始化定義開始CMS收集	禁止hostspot自行觸發CMS GC

-XX:CMSInitiatingOccupancyFraction=70使用cms作為垃圾回收

使用70％後開始CMS收集92

輔助信息

-XX:+PrintGC 輸出形式:[GC 118250K->113543K(130112K), 0.0094143 secs]

[Full GC 121376K->10414K(130112K), 0.0650971 secs]

輸出形式:[GC [DefNew: 8614K->781K(9088K), 0.0123035 secs] 118250K->113543K(130112K), 0.0124633 secs]

[GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs][Tenured: 112761K->10414K(121024K), 0.0433488 secs] 121376K->10414K(130112K), 0.0436268 secs]

可與-XX:+PrintGC -XX:+PrintGCDetails混合使用

輸出形式:11.851: [GC 98328K->93620K(130112K), 0.0082960 secs]

-XX:+PrintGCApplicationStoppedTime	列印垃圾回收期間程序暫停的時間.可與上面混合使用	輸出形式:Total time for which application threads were stopped: 0.0468229 seconds
-XX:+PrintGCApplicationConcurrentTime	列印每次垃圾回收前,程序未中斷的執行時間.可與上面混合使用	輸出形式:Application time: 0.5291524 seconds
-XX:+PrintHeapAtGC	列印GC前後的詳細堆棧信息	
-Xloggc:filename	把相關日誌信息記錄到文件以便分析. 與上面幾個配合使用	
-XX:+PrintClassHistogram	garbage collects before printing the histogram.	

Desired survivor size 1048576 bytes, new threshold 7 (max 15)

new threshold 7即標識新的存活周期的閾值為7。

GC性能方面的考慮

對於GC的性能主要有2個方面的指標：吞吐量throughput（工作時間不算gc的時間占總的時間比）和暫停pause（gc發生時app對外顯示的無法響應）。

1. Total Heap

默認情況下，vm會增加/減少heap大小以維持free space在整個vm中占的比例，這個比例由MinHeapFreeRatio和MaxHeapFreeRatio指定。

一般而言，server端的app會有以下規則：

- 對vm分配儘可能多的memory；
- 將Xms和Xmx設為一樣的值。如果虛擬機啟動時設置使用的內存比較小，這個時候又需要初始化很多對象，虛擬機就必須重複地增加內存。

- 處理器核數增加，內存也跟著增大。

2. The Young Generation

另外一個對於app流暢性運行影響的因素是young generation的大小。young generation越大，minor collection越少；但是在固定heap size情況下，更大的young generation就意味著小的tenured generation，就意味著更多的major collection(major collection會引發minor collection)。

NewRatio反映的是young和tenured generation的大小比例。NewSize和MaxNewSize反映的是young generation大小的下限和上限，將這兩個值設為一樣就固定了young generation的大小（同Xms和Xmx設為一樣）。

如果希望，SurvivorRatio也可以優化survivor的大小，不過這對於性能的影響不是很大。SurvivorRatio是eden和survior大小比例。

- 首先決定能分配給vm的最大的heap size，然後設定最佳的young generation的大小；
- 如果heap size固定後，增加young generation的大小意味著減小tenured generation大小。讓tenured generation在任何時候夠大，能夠容納所有live的data（留10%-20%的空餘）。

經驗&&規則

1. 年輕代大小選擇
    - 響應時間優先的應用:儘可能設大,直到接近系統的最低響應時間限制(根據實際情況選擇).在此種情況下,年輕代收集發生的頻率也是最小的.同時,減少到達年老代的對象.
    - 吞吐量優先的應用:儘可能的設置大,可能到達Gbit的程度.因為對響應時間沒有要求,垃圾收集可以並行進行,一般適合8CPU以上的應用.
    - 避免設置過小.當新生代設置過小時會導致:1.YGC次數更加頻繁 2.可能導致YGC對象直接進入舊生代,如果此時舊生代滿了,會觸發FGC.
2. 年老代大小選擇
    1. 響應時間優先的應用:年老代使用並發收集器,所以其大小需要小心設置,一般要考慮並發會話率和會話持續時間等一些參數.如果堆設置小了,可以會造成內存碎 片,高回收頻率以及應用暫停而使用傳統的標記清除方式;如果堆大了,則需要較長的收集時間.最優化的方案,一般需要參考以下數據獲得:

        並發垃圾收集信息、持久代並發收集次數、傳統GC信息、花在年輕代和年老代回收上的時間比例。

    2. 吞吐量優先的應用:一般吞吐量優先的應用都有一個很大的年輕代和一個較小的年老代.原因是,這樣可以儘可能回收掉大部分短期對象,減少中期的對象,而年老代盡存放長期存活對象.
3. 較小堆引起的碎片問題

    因為年老代的並發收集器使用標記,清除算法,所以不會對堆進行壓縮.當收集器回收時,他會把相鄰的空間進行合併,這樣可以分配給較大的對象.但是,當堆空間較小時,運行一段時間以後,就會出現"碎片",如果並發收集器找不到足夠的空間,那麼並發收集器將會停止,然後使用傳統的標記,清除方式進行回收.如果出現"碎片",可能需要進行如下配置:

    -XX:+UseCMSCompactAtFullCollection:使用並發收集器時,開啟對年老代的壓縮.

    -XX:CMSFullGCsBeforeCompaction=0:上面配置開啟的情況下,這裡設置多少次Full GC後,對年老代進行壓縮
4. 用64位作業系統，Linux下64位的jdk比32位jdk要慢一些，但是吃得內存更多，吞吐量更大
5. XMX和XMS設置一樣大，MaxPermSize和MinPermSize設置一樣大，這樣可以減輕伸縮堆大小帶來的壓力

6. 使用CMS的好處是用儘量少的新生代，經驗值是128M－256M， 然後老生代利用CMS並行收集， 這樣能保證系統低延遲的吞吐效率。 實際上cms的收集停頓時間非常的短，2G的內存， 大約20－80ms的應用程式停頓時間
7. 系統停頓的時候可能是GC的問題也可能是程序的問題，多用jmap和jstack查看，或者killall -3 java，然後查看java控制台日誌，能看出很多問題。(相關工具的使用方法將在後面的blog中介紹)
8. 仔細了解自己的應用，如果用了緩存，那麼年老代應該大一些，緩存的HashMap不應該無限制長，建議採用LRU算法的Map做緩存，LRUMap的最大長度也要根據實際情況設定。
9. 採用並發回收時，年輕代小一點，年老代要大，因為年老大用的是並發回收，即使時間長點也不會影響其他程序繼續運行，網站不會停頓
10. JVM參數的設置(特別是 –Xmx –Xms –Xmn -XX:SurvivorRatio -XX:MaxTenuringThreshold等參數的設置沒有一個固定的公式，需要根據PV old區實際數據 YGC次數等多方面來衡量。為了避免promotion faild可能會導致xmn設置偏小，也意味著YGC的次數會增多，處理並發訪問的能力下降等問題。每個參數的調整都需要經過詳細的性能測試，才能找到特定應用的最佳配置。

promotion failed:

垃圾回收時promotion failed是個很頭痛的問題，一般可能是兩種原因產生，第一個原因是救助空間不夠，救助空間裡的對象還不應該被移動到年老代，但年輕代又有很多對象需要放入救助空間；第二個原因是年老代沒有足夠的空間接納來自年輕代的對象；這兩種情況都會轉向Full GC，網站停頓時間較長。

解決方方案一：

第一個原因我的最終解決辦法是去掉救助空間，設置-XX:SurvivorRatio=65536 -XX:MaxTenuringThreshold=0即可，第二個原因我的解決辦法是設置CMSInitiatingOccupancyFraction為某個值（假設70），這樣年老代空間到70%時就開始執行CMS，年老代有足夠的空間接納來自年輕代的對象。

解決方案一的改進方案：

又有改進了，上面方法不太好，因為沒有用到救助空間，所以年老代容易滿，CMS執行會比較頻繁。我改善了一下，還是用救助空間，但是把救助空間加大，這樣也不會有promotion failed。具體操作上，32位Linux和64位Linux好像不一樣，64位系統似乎只要配置MaxTenuringThreshold參數，CMS還是有暫停。為了解決暫停問題和promotion failed問題，最後我設置-XX:SurvivorRatio=1 ，並把MaxTenuringThreshold去掉，這樣即沒有暫停又不會有promotoin failed，而且更重要的是，年老代和永久代上升非常慢（因為好多對象到不了年老代就被回收了），所以CMS執行頻率非常低，好幾個小時才執行一次，這樣，伺服器都不用重啟了。

-Xmx4000M -Xms4000M -Xmn600M -XX:PermSize=500M -XX:MaxPermSize=500M -Xss256K -XX:+DisableExplicitGC -XX:SurvivorRatio=1 -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 -XX:+CMSClassUnloadingEnabled -XX:LargePageSizeInBytes=128M -XX:+UseFastAccessorMethods -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=80 -XX:SoftRefLRUPolicyMSPerMB=0 -XX:+PrintClassHistogram -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintHeapAtGC -Xloggc:log/gc.log

CMSInitiatingOccupancyFraction值與Xmn的關係公式

上面介紹了promontion faild產生的原因是EDEN空間不足的情況下將EDEN與From survivor中的存活對象存入To survivor區時,To survivor區的空間不足，再次晉升到old gen區，而old gen區內存也不夠的情況下產生了promontion faild從而導致full gc.那可以推斷出：eden+from survivor < old gen區剩餘內存時，不會出現promontion faild的情況，即：

(Xmx-Xmn)*(1-CMSInitiatingOccupancyFraction/100)>=(Xmn-Xmn/(SurvivorRatior+2)) 進而推斷出：

CMSInitiatingOccupancyFraction <=((Xmx-Xmn)-(Xmn-Xmn/(SurvivorRatior+2)))/(Xmx-Xmn)*100

例如：

當xmx=128 xmn=36 SurvivorRatior=1時 CMSInitiatingOccupancyFraction<=((128.0-36)-(36-36/(1+2)))/(128-36)*100 =73.913

當xmx=128 xmn=24 SurvivorRatior=1時 CMSInitiatingOccupancyFraction<=((128.0-24)-(24-24/(1+2)))/(128-24)*100=84.615…

當xmx=3000 xmn=600 SurvivorRatior=1時 CMSInitiatingOccupancyFraction<=((3000.0-600)-(600-600/(1+2)))/(3000-600)*100=83.33

CMSInitiatingOccupancyFraction低於70% 需要調整xmn或SurvivorRatior值。

令：

網上一童鞋推斷出的公式是：:(Xmx-Xmn)*(100-CMSInitiatingOccupancyFraction)/100>=Xmn 這個公式個人認為不是很嚴謹，在內存小的時候會影響xmn的計算。

關於實際環境的GC參數配置見:實例分析 監測工具見JVM監測

JAVA HOTSPOT VM

java jvm 參數 -Xms -Xmx -Xmn -Xss 調優總結

Java HotSpot VM Options

Frequently Asked Questions About the Java HotSpot VM

Java SE HotSpot at a Glance

Java性能調優筆記(內附測試例子 很有用)

* 以上用戶言論只代表其個人觀點，不代表CSDN網站的觀點或立場