通向架構師的道路（第四天）之Tomcat性能調優-讓小貓飛奔 - lifetragedy的專欄 - 博客頻道

分類：

一、總結前一天的學習

從「第三天」的性能測試一節中，我們得知了決定性能測試的幾個重要指標，它們是：

ü 吞吐量

ü Responsetime

ü Cpuload

ü MemoryUsage

我們也在第三天的學習中對Apache做過了一定的優化，使其最優化上述4大核心指標的讀數，那麼我們的Apache調優了，我們的Tomcat也作些相應的調整，當完成今的課程後，到時你的「小貓」到時真的會「飛」起來的，所以請用心看完，這篇文章一方面用來向那位曾寫過「Tomcat如何承受1000個用戶」的作都的敬，一方面又是這篇原文的一個擴展，因為在把原文的知識用到相關的兩個大工程中去後解決了：

1) 承受更大並發用戶數

2) 取得了良好的性能與改善（系統平均性能提升達20倍，極端一個交易達80倍）。

另外值的一提的是，我們當時工程裡用的「小貓」是跑在32位機下的， 也就是我們的JVM最大受到2GB內存的限制，都已經跑成「飛」了。。。。。。如果在64位機下跑這頭「小貓」。。。。。。大家可想而知，會得到什麼樣的效果呢？下面就請請詳細的設置吧！

二、一切基於JVM（內存）的優化

2.1 32位操作系統與64位操作系統中JVM的對比

我們一般的開發人員，基本用的是都是32位的Windows系統，這就導致了一個嚴重的問題即：32位windows系統對內存限制，下面先來看一個比較的表格：

操作系統	操作系統位數	內存限制	解決辦法
Winxp	32	4GB	超級兔子
Win7	32	4GB	可以通過設置/PAE
Win2003	32	可以突破4GB達16GB	必需要裝win2003 advanced server且要打上sp2補丁

Win7	64	無限制	機器能插多少內存，系統內存就能支持到多大
Win2003	64	無限制	機器能插多少內存，系統內存就能支持到多大
Linux	64	無限制	機器能插多少內存，系統內存就能支持到多大
Unix	64	無限制	機器能插多少內存，系統內存就能支持到多大

上述問題解決後，我們又碰到一個新的問題，32位系統下JVM對內存的限制：不能突破2GB內存，即使你在Win2003 Advanced Server下你的機器裝有8GB-16GB的內存，而你的JAVA，只能用到2GB的內存。

其實我一直很想推薦大家使用Linux或者是Mac操作系統的，而且要裝64位，因為必竟我們是開發用的不是打遊戲用的，而Java源自Unix歸於Unix（Linux只是運行在PC上的Unix而已）。

所以很多開發人員運行在win32位系統上更有甚者在生產環境下都會佈署win32位的系統，那麼這時你的Tomcat要優化，就要講究點技巧了。**而在64位操作系統上無論是系統內存還是JVM都沒有受到2GB這樣的限制。**

Tomcat的優化分成兩塊：

ü Tomcat啟動命令行中的優化參數即JVM優化

ü Tomcat容器自身參數的優化（這塊很像ApacheHttp Server）

這一節先要講的是Tomcat啟動命令行中的優化參數。

Tomcat首先跑在JVM之上的，因為它的啟動其實也只是一個java命令行，首先我們需要對這個JAVA的啟動命令行進行調優。

需要注意的是：

這邊討論的JVM優化是基於Oracle Sun的jdk1.6版有以上，其它JDK或者低版本JDK不適用。

2.2 Tomcat啟動行參數的優化

Tomcat 的啟動參數位於tomcat的安裝目錄\bin目錄下，如果你是Linux操作系統就是catalina.sh文件，如果你是Windows操作系統那麼你需要改動的就是catalina.bat文件。打開該文件，一般該文件頭部是一堆的由##包裹著的註釋文字，找到註釋文字的最後一段如：

```
# $Id: catalina.sh 522797 2007-03-27 07:10:29Z fhanik $
# -----------------------------------------------------------------------


# OS specific support.  $var _must_ be set to either true or false.
```

敲入一個回車，加入如下的參數

Linux系統中tomcat的啟動參數

```
export JAVA_OPTS="-server -Xms1400M -Xmx1400M -Xss512k -XX:+AggressiveOpts -XX:+UseBiasedLocking
-XX:PermSize=128M -XX:MaxPermSize=256M -XX:+DisableExplicitGC -XX:MaxTenuringThreshold=31 -
XX:+UseConcMarkSweepGC -XX:+UseParNewGC  -XX:+CMSParallelRemarkEnabled -
XX:+UseCMSCompactAtFullCollection -XX:LargePageSizeInBytes=128m  -XX:+UseFastAccessorMethods -
XX:+UseCMSInitiatingOccupancyOnly -Djava.awt.headless=true "
```

Windows系統中tomcat的啟動參數

```
set JAVA_OPTS=-server -Xms1400M -Xmx1400M -Xss512k -XX:+AggressiveOpts -XX:+UseBiasedLocking -
XX:PermSize=128M -XX:MaxPermSize=256M -XX:+DisableExplicitGC -XX:MaxTenuringThreshold=31 -
XX:+UseConcMarkSweepGC -XX:+UseParNewGC  -XX:+CMSParallelRemarkEnabled -
XX:+UseCMSCompactAtFullCollection -XX:LargePageSizeInBytes=128m  -XX:+UseFastAccessorMethods -
XX:+UseCMSInitiatingOccupancyOnly -Djava.awt.headless=true
```

上面參數好多啊，可能有人寫到現在都沒見一個tomcat的啟動命令裡加了這麼多參數，當然，這些參數隻是我機器上的，不一定適合你，尤其是參數後的value（值）是需要根據你自己的實際情況來設置的。

參數解釋：

ü -server

我不管你什麼理由，只要你的tomcat是運行在生產環境中的，這個參數必須給我加上

因為tomcat默認是以一種叫java –client的模式來運行的，server即意味著你的tomcat是以真實的production的模式在運行的，這也就意味著你的tomcat以server模式運行時將擁有：更大、更高的並發處理能力，更快更強捷的JVM垃圾回收機制，可以獲得更多的負載與吞吐量。。。更。。。還有更。。。

Y給我記住啊，要不然這個-server都不加，那是要打屁股了。

ü -Xms–Xmx

即JVM內存設置了，把Xms與Xmx兩個值設成一樣是最優的做法，有人說Xms為最小值，Xmx為最大值不是挺好的，這樣設置還比較人性化，科學化。人性？科學？你個頭啊。

大家想一下這樣的場景：

一個系統隨著並發數越來越高，它的內存使用情況逐步上升，上升到最高點不能上升了，開始回落，你們不要認為這個回落就是好事情，由其是大起大落，在內存回落時它付出的代價是CPU高速開始運轉進行垃圾回收，此時嚴重的甚至會造成你的系統出現「卡殼」就是你在好好的操作，突然網頁像死在那邊一樣幾秒甚至十幾秒時間，因為JVM正在進行垃圾回收。

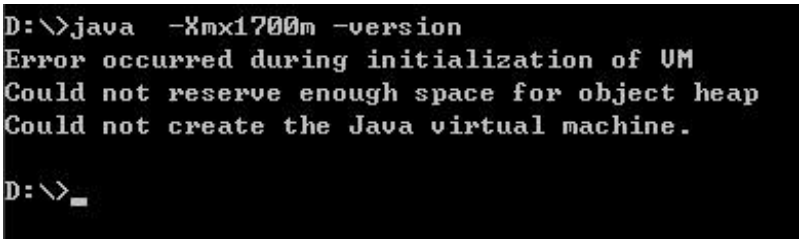
因此一開始我們就把這兩個設成一樣，使得Tomcat在啟動時就為最大化參數充分利用系統的效率，這個道理和jdbcconnection pool裡的minpool size與maxpool size的需要設成一個數量是一樣的原理。

如何知道我的JVM能夠使用最大值啊？拍腦袋？不行！

在設這個最大內存即Xmx值時請先打開一個命令行，鍵入如下的命令：

看，能夠正常顯示JDK的版本信息，說明，這個值你能夠用。不是說32位系統下最高能夠使用2GB內存嗎？即：2048m，我們不防來試試

可以嗎？不可以！不要說2048m呢，我們小一點，試試1700m如何



嘿嘿，連1700m都不可以，更不要說2048m了呢，2048m只是一個理論數值，這樣說吧我這邊有幾台機器，有的機器-Xmx1800都沒問題，有的機器最高只能到-Xmx1500m。

因此在設這個-Xms與-Xmx值時一定一定記得先這樣測試一下，要不然直接加在tomcat啟動命令行中你的tomcat就再也起不來了，要飛是飛不了，直接成了一隻瘟貓了。

ü –Xmn

設置年輕代大小為512m。整個堆大小=年輕代大小 + 年老代大小 + 持久代大小。持久代一般固定大小為64m，所以增大年輕代後，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。

ü -Xss

是指設定每個線程的堆棧大小。這個就要依據你的程序，看一個線程 大約需要佔用多少內存，可能會有多少線程同時運行等。一般不易設置超過1M，要不然容易出現out ofmemory。

ü -XX:+AggressiveOpts

作用如其名（aggressive），啟用這個參數，則每當JDK版本升級時，你的JVM都會使用最新加入的優化技術（如果有的話）

ü -XX:+UseBiasedLocking

啟用一個優化了的線程鎖，我們知道在我們的appserver，每個http請求就是一個線程，有的請求短有的請求長，就會有請求排隊的現象，甚至還會出現線程阻塞，這個優化了的線程鎖使得你的appserver內對線程處理自動進行最優調配。

ü -XX:PermSize=128M-XX:MaxPermSize=256M

JVM使用-XX:PermSize設置非堆內存初始值，默認是物理內存的1/64；

在數據量的很大的文件導出時，一定要把這兩個值設置上，否則會出現內存溢出的錯誤。

由XX:MaxPermSize設置最大非堆內存的大小，默認是物理內存的1/4。

那麼，如果是物理內存4GB，那麼64分之一就是64MB，這就是PermSize默認值，也就是永生代內存初始大小；

四分之一是1024MB，這就是MaxPermSize默認大小。

ü -XX:+DisableExplicitGC

在程序代碼中不允許有顯示的調用」System.gc()」。看到過有兩個極品工程中每次在DAO操作結束時手動調用System.gc()一下，覺得這樣做好像能夠解決它們的out ofmemory問題一樣，付出的代價就是系統響應時間嚴重降低，就和我在關於Xms,Xmx裡的解釋的原理一樣，這樣去調用GC導致系統的JVM大起大落，性能不到什麼地方去喲！

ü -XX:+UseParNewGC

對年輕代採用多線程並行回收，這樣收得快。

ü -XX:+UseConcMarkSweepGC

即CMS gc，這一特性只有jdk1.5即後續版本才具有的功能，它使用的是gc估算觸發和heap佔用觸發。

我們知道頻頻繁的GC會造面JVM的大起大落從而影響到系統的效率，因此使用了CMS GC後可以在GC次數增多的情況下，每次GC的響應時間卻很短，比如說使用了CMS GC後經過jprofiler的觀察，GC被觸發次數非常多，而每次GC耗時僅為幾毫秒。

ü -XX:MaxTenuringThreshold

設置垃圾最大年齡。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概率。

這個值的設置是根據本地的jprofiler監控後得到的一個理想的值，不能一概而論原搬照抄。

ü -XX:+CMSParallelRemarkEnabled

在使用UseParNewGC 的情況下, 儘量減少 mark 的時間

ü -XX:+UseCMSCompactAtFullCollection

在使用concurrent gc 的情況下, 防止 memoryfragmention, 對live object 進行整理, 使 memory 碎片減少。

ü -XX:LargePageSizeInBytes

指定 Java heap的分頁頁面大小

ü -XX:+UseFastAccessorMethods

get,set 方法轉成本地代碼

ü -XX:+UseCMSInitiatingOccupancyOnly

指示只有在 oldgeneration 在使用了初始化的比例後concurrent collector 啟動收集

ü -XX:CMSInitiatingOccupancyFraction=70

CMSInitiatingOccupancyFraction，這個參數設置有很大技巧，基本上滿足**(Xmx-Xmn)*(100-CMSInitiatingOccupancyFraction)/100>=Xmn**就不會出現promotion failed。在我的應用中Xmx是6000，Xmn是512，那麼Xmx-Xmn是5488兆，也就是年老代有5488 兆，CMSInitiatingOccupancyFraction=90說明年老代到90%滿

的時候開始執行對年老代的並發垃圾回收（CMS），這時還 剩10%的空間是5488*10%=548兆，所以即使Xmn（也就是年輕代共512兆）裡所有對象都搬到年老代裡，548兆的空間也足夠了，所以只要滿 足上面的公式，就不會出現垃圾回收時的promotion failed；

因此這個參數的設置必須與Xmn關聯在一起。

ü -Djava.awt.headless=true

這個參數一般我們都是放在最後使用的，這全參數的作用是這樣的，有時我們會在我們的J2EE工程中使用一些圖表工具如：jfreechart，用於在web網頁輸出GIF/JPG等流，在winodws環境下，一般我們的app server在輸出圖形時不會碰到什麼問題，但是在linux/unix環境下經常會碰到一個exception導致你在winodws開發環境下圖片顯示的好好可是在linux/unix下卻顯示不出來，因此加上這個參數以免避這樣的情況出現。

上述這樣的配置，基本上可以達到：

ü 系統響應時間增快

ü JVM回收速度增快同時又不影響系統的響應率

ü JVM內存最大化利用

ü 線程阻塞情況最小化

2.3 Tomcat容器內的優化

前面我們對Tomcat啟動時的命令進行了優化，增加了系統的JVM可使用數、垃圾回收效率與線程阻塞情況、增加了系統響應效率等還有一個很重要的指標，我們沒有去做優化，就是吞吐量。

還記得我們在第三天的學習中說的，這個系統本身可以處理1000，你沒有優化和配置導致它默認只能處理25。因此下面我們來看Tomcat容器內的優化。

打開tomcat安裝目錄\conf\server.xml文件，定位到這一行：

```
<Connector port="8080" protocol="HTTP/1.1"
```

這一行就是我們的tomcat容器性能參數設置的地方，它一般都會有一個默認值，這些默認值是遠遠不夠我們的使用的，我們來看經過更改後的這一段的配置：

```
<Connector port="8080" protocol="HTTP/1.1"
        URIEncoding="UTF-8"  minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true" connectionTimeout="20000"
        acceptCount="300"  maxThreads="300" maxProcessors="1000" minProcessors="5"
        useURIValidationHack="false"
                                compression="on" compressionMinSize="2048"
                                compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"
            redirectPort="8443"
/>
```

好大一陀唉。。。。。。

沒關係，一個個來解釋

ü URIEncoding=」UTF-8」

使得tomcat可以解析含有中文名的文件的url，真方便，不像apache裡還有搞個mod_encoding，還要手工編譯

ü maxSpareThreads

maxSpareThreads 的意思就是如果空閒狀態的線程數多於設置的數目，則將這些線程中止，減少這個池中的線程總數。

ü minSpareThreads

最小備用線程數，tomcat啟動時的初始化的線程數。

ü enableLookups

這個功效和Apache中的HostnameLookups一樣，設為關閉。

ü connectionTimeout

connectionTimeout為網絡連接超時時間毫秒數。

ü maxThreads

maxThreads Tomcat使用線程來處理接收的每個請求。這個值表示Tomcat可創建的最大的線程數，即最大並發數。

ü acceptCount

acceptCount是當線程數達到maxThreads後，後續請求會被放入一個等待隊列，這個acceptCount是這個隊列的大小，如果這個隊列也滿了，就直接refuse connection

ü maxProcessors與minProcessors

在 Java中線程是程序運行時的路徑，是在一個程序中與其它控制線程無關的、能夠獨立運行的代碼段。它們共享相同的地址空間。多線程幫助程序員寫出CPU最 大利用率的高效程序，使空閒時間保持最低，從而接受更多的請求。

通常Windows是1000個左右，Linux是2000個左右。

ü useURIValidationHack

我們來看一下tomcat中的一段源碼：

```
security
        if (connector.getUseURIValidationHack()) {
            String uri = validate(request.getRequestURI());
            if (uri == null) {
                res.setStatus(400);
                res.setMessage("Invalid URI");
                throw new IOException("Invalid URI");
            } else {
                req.requestURI().setString(uri);
                // Redoing the URI decoding
                req.decodedURI().duplicate(req.requestURI());
                req.getURLDecoder().convert(req.decodedURI(), true);
            }
        }
```

可以看到如果把useURIValidationHack設成"false"，可以減少它對一些url的不必要的檢查從而減省開銷。

ü enableLookups="false"

為了消除DNS查詢對性能的影響我們可以關閉DNS查詢，方式是修改server.xml文件中的enableLookups參數值。

ü disableUploadTimeout

類似於Apache中的keeyalive一樣

ü 給Tomcat配置gzip壓縮(HTTP壓縮)功能

```
compression="on" compressionMinSize="2048"
compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"
```

HTTP 壓縮可以大大提高瀏覽網站的速度，它的原理是，在客戶端請求網頁後，從服務器端將網頁文件壓縮，再下載到客戶端，由客戶端的瀏覽器負責解壓縮並瀏覽。相對於普通的瀏覽過程HTML,CSS,Javascript , Text ，它可以節省40%左右的流量。更為重要的是，它可以對動態生成的，包括CGI、PHP , JSP , ASP , Servlet,SHTML等輸出的網頁也能進行壓縮，壓縮效率驚人。

1)compression="on" 打開壓縮功能

2)compressionMinSize="2048" 啟用壓縮的輸出內容大小，這裡面默認為2KB

3)noCompressionUserAgents="gozilla, traviata" 對於以下的瀏覽器，不啟用壓縮

4)compressableMimeType="text/html,text/xml" 壓縮類型

最後不要忘了把8443端口的地方也加上同樣的配置，因為如果我們走https協議的話，我們將會用到8443端口這個段的配置，對吧？

```
<!--enable tomcat ssl-->
   <Connector port="8443" protocol="HTTP/1.1"
              URIEncoding="UTF-8"  minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true" connectionTimeout="20000"
        acceptCount="300"  maxThreads="300" maxProcessors="1000" minProcessors="5"
        useURIValidationHack="false"
                compression="on" compressionMinSize="2048"
                compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"
             SSLEnabled="true"
         scheme="https" secure="true"
         clientAuth="false" sslProtocol="TLS"
         keystoreFile="d:/tomcat2/conf/shnlap93.jks" keystorePass="aaaaaa"
     />
```

好了，所有的Tomcat優化的地方都加上了。結闔第三天中的Apache的性能優化，我們這個架構可以「飛奔」起來了，當然這邊把有提及任何關於數據庫優化的步驟，但僅憑這兩步，我們的系統已經有了很大的提升。

舉個真實的例子：上一個項目，經過4輪performance testing，第一輪進行了問題的定位，第二輪就是進行了apache+tomcat/weblogic的優化，第三輪是做集群優化，第四輪是sql與codes的優化。

在到達第二輪時，我們的性能已經提升了多少倍呢？我們來看一個loaderrunner的截圖吧：

左邊第一列是第一輪沒有經過任何調優的壓力測試報告。

右邊這一列是經過了apache優化，tomcat優化後得到的壓力測試報告。

大家看看，這就提高了多少倍？這還只是在沒有改動代碼的情況下得到的改善，現在明白了好好的調優一

個apache和tomcat其實是多麼的重要了？如果加上後面的代碼、SQL的調優、數據庫的調優。。。。。。所以我在上一個工程中有單筆交易性能（無論是吞吐量、響應時間）提高了80倍這樣的極端例子的存在。


Average
9.01
12.805
8.661
9.439
12.483
3.802
0.672
1.229
2.003

Average
0.114
0.265
0.11
0.11
0.414
0.265
0.104
0.079
0.196