튜닝 및 모니터링

SUN JVM 튜닝 옵션

2013. 11. 01

TmaxSoft

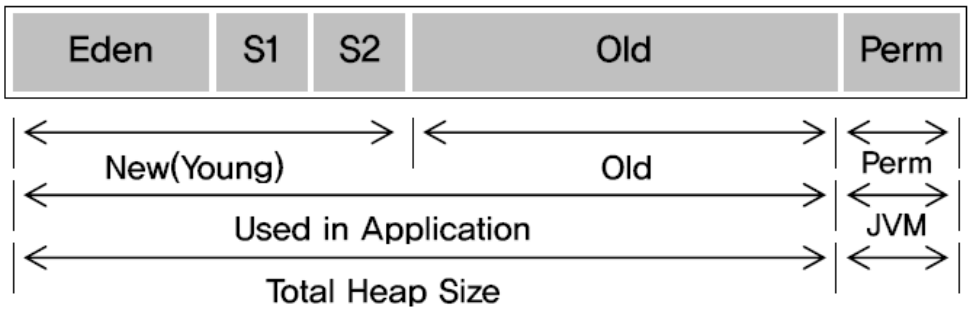
목차

1. 개요

SUN JVM 의 특징을 살펴보고, Trouble Shooting 방법과, 실제 Site 튜닝사례를 살펴보도록 한다.

2. JVM 특징 소개

JVM 메모리영역



영역	설명
New(Young)	이 영역은 Java객체가 처음 생성되어 저장되는 곳으로, 시간이 지나면 우선순위에 따라 Old 영역으로 옮겨진다.
Old	New(Young)영역에서 오래된 객체가 이동된다.
Perm	Class, Method등의 Code가 저장되는 영역

GC 알고리즘

Minor GC

New 영역은 Eden 과 Survivor 두 영역으로 나뉘어지고, Survivor 는 Survivor1 과 Survivor2 로 다시 나뉘어진다. Eden 영역은 객체가 생성되자마자 저장이 되는 곳이고, Minor GC 가 발생하면 Eden 과 Survivor1 에 사용한지 오래 되지 않은 객체를 Suvivor2 로 복사한다. 그리고 Survivor1 과 Eden 영역을 Clear 한다. 다음 번 Minor GC 가 발생하면 같은 원리로 Eden 과 Survivor1 영역에서 사용하던 객체를 Survivor2 에 복사한다. 계속 이런 방법을 반복적으로 수행하면서 Minor GC 를 수행한다.

이렇게 Minor GC 를 수행하다가, Survivor2 영역에서 오래된 객체는 지정된 비율에 다다르면 Old 영역으로 옮겨지게 된다. 객체 생성 시간이 오래되면 Eden 영역이나 Survivor 영역에서 Old 영역으로 이동을 하게 된다.

Full GC

Old 영역 GC 를 Full GC 라고 부르며, Mark & Compact 알고리즘을 이용한다. 이 알고리즘은 전체 객체들의 reference tree 를 찾아가면서 연결되지 않는 객체를 Mark 한다. 이때 사용되지 않는 객체를 모두 Mark 가 되고, 그 다음으로 객체를 Compact 기법으로 사용하지 않도록 변경해버린다.

Full GC 는 속도가 느리기 때문에, Full GC 가 일어나는 도중에는 thread 들의 순간적으로 AP 가 멈춰 버리기 때문에, 성능과 안정성에 큰 영향을 준다. 다음 장에서는 Full GC 시간을 줄이기 위한 방법을 알아보도록 한다.

3. JVM 주요 옵션 소개

일반 주요옵션

-Xnoclassgc

Class Garbage Collection을 수행하지 않는다.

-Xloggc:<file>

GC Log를 기록할 파일명을 지정한다. 파일명을 지정하지 않으면 Standard Out이나 Standard Error 콘솔에 출력된다.

주로 -XX:+PrintGCTimeStamps, -XX:+PrintGCDetails 옵션과 같이 사용된다.

-Xint

Intepreter 모드로만 ByteCode를 실행한다. Interpreter 모드로 실행될 경우에는 JIT Compile 기능이 동작하지 않는다. 이 옵션을 활성화하면 실행 속도이 다소 저하될 수 있다. 따라서 HotSpot Compiler의 버그로 문제가 생길 때만 사용이 권장된다.

-Xmixed

Mixed 모드로 ByteCode를 실행한다. HotSpot JVM의 Default 동작 방식이며, -Xint 옵션과 상호 배타적인 옵션이다

-Xms<size>

Java Heap의 최초 크기(Start Size)를 지정한다.

-Xmx<size>

Java Heap의 최대 크기(Maximum Size)를 지정한다.

-Xmn

Young Generation이 거주하는 New Space의 크기를 지정한다. 대개의 경우 이 옵션보다는 -XX:NewRatio 옵션이나 -XX:NewSize 옵션을 많이 사용한다

-Xss<size>

개별 Thread의 Stack Size를 지정한다. 예를 들어 Thread Stack Size가 1M이고, Thread가 최대 100개 활성화된다면, 최대 100M의 메모리를 사용하게 된다. 대부분의 경우 기본값(Default)을 그대로 사용하는 것이 바람직하다. 많은 수의 Thread를 사용하는 Application의 경우 Thread Stack에 의한 메모리 요구량이 높아지며 이로 인해 Out Of Memory Error가 발생할 수 있다. 이런 경우에는 -Xss 옵션을 이용해 Thread Stack Size를 줄여주어

야 한다.(1.4이하에서는 Thread의 Native Stack Size를 의미)

-Xoss<size>

개별 Thread의 java stack Size를 지정한다. JDK5.0부터는 Xss로 통합이 되었으므로 의미가 없다.

-XX:SurvivorRatio=8

Survivor Space와 Eden Space의 비율을 지정한다. 만일 이 값이 6이면, To Survivor Ratio:From Survivor Ratio:Eden Space = 1:1:6이 된다. 즉, 하나의 Survivor Space의 크기가 Young Generation의 1/8이 된다. Survivor Space의 크기가 크면 Tenured Generation으로 옮겨가기 전의 중간 버퍼 영역이 커지는 셈이다. 따라서 Full GC의 빈도를 줄이는 역할을 할 수 있다. 반면 Eden Space의 크기가 줄어들므로 Minor GC가 자주 발생하게 된다.

-XX:MaxHeapFreeRatio=70

Heap Shrinkage를 수행하는 임계치를 지정한다. 예를 들어 이 값이 70이면 Heap의 Free 공간이 70% 이상이 되면 Heap 크기가 축소된다. MinHeapFreeRatio 옵션과 함께 Heap의 크기 조정을 담당한다.

-XX:MinHeapFreeRatio=40

Heap Expansion을 수행하는 임계치를 지정한다. 예를 들어 이 값이 40이면 Heap의 Free 공간이 40% 미만이 되면 Heap 크기가 확대된다. MaxHeapFreeRatio 옵션과 함께 Heap의 크기 조정을 담당한다.

-XX:NewSize=2.125m

Young Generation의 시작 크기를 지정한다. Young Generation의 크기는 NewSize 옵션(시작 크기)과 MaxNewSize 옵션(최대 크기)에 의해 결정된다.

-XX:MaxNewSize=size

Young Generation의 최대 크기를 지정한다. Young Generation의 시작 크기는 NewSize 옵션에 의해 지정된다.

-XX:PermSize

Permanent Generation의 최초 크기를 지정한다. Permanent Generation의 최대 크기는 MaxPermSize 옵션에 의해 지정된다. 많은 수의 Class를 로딩하는 Application은 큰 크기의 Permanent Generation을 필요로 하며, Permanent Generation의 크기가 작아서 Class를 로딩하는 못하면 Out Of Memory Error가 발생한다.

-XX:MaxPermSize=64m

Permanent Generation의 최대 크기를 지정한다. Permanent Generation의 시작 크기는 PermSize 옵션에 의해 지정된다. 많은 수의 Class를 Loading하는 Application은 PermSize와 MaxPermSize 옵션을 이용해 Permanent Generation의 크기를 크게 해주는 것이 좋다. Permanent Generation의 크기가 작을 경우에는 Out Of Memory Error가 발생할 수 있다.

-XX:NewRatio=2

Young Generation과 Old Generation의 비율을 결정한다. 예를 들어 이 값이 2이면 Young:Old = 1:2 가 되고, Young Generation의 크기는 전체 Java Heap의 1/3이 된다

[sparc -server: 2, sparc -client: 4 (1.3) 8 (1.3.1+), intel: 12]

-XX:+DisableExplicitGC

System.gc 호출에 의한 Explicit GC를 비활성화한다. RMI에 의한 Explicit GC나 Application에서의 Explicit GC를 원천적으로 방지하고자 할 경우에 사용된다.

GC종류에 따른 옵션

-XX:+UseParallelGC

Throughput Garbage Collector는 New Generation에 의한 Minor GC 작업을 여러 개의 Thread(CPU의 개수만큼)를 이용해 Parallel하게 진행한다

Server 급 머신 에서는 이 Collector가 기본 설정이라는 것은 의미하는 바가 크다. 특히 CPU 자원이 풍부한 환경에서는 큰 효과를 얻을 수 있다.

-XX:ParallelGCThreads 옵션을 이용하면 Parallel Collector Threads의 개수를 조절할 수 있다.

-XX:+UseConcMarkSweepGC

Throughput Garbage Collector와는 달리 Response Time를 최적화하게끔 동작한다. 정확하게 말하면 Response Time을 최소화하기 위해 Full GC 과정을 되도록이면 Thread 블로킹 없이 진행한다. Thread의 정지 시간(Pause Time)이 최소화되기 때문에 Low Pause Garbage Collector라는 이름이 붙었다.

-XX:+UseParNewGC

CMS를 사용할 경우 Minor GC를 Parallel하게 할 지의 여부를 지정한다. CPU가 복수 개일 경우는 기본적으로 이 모드를 사용한다.

-XX:+CMSParallelRemarkEnabled

-XX:+UseParNewGC 와 함께 사용할 경우 GC대상을 표시하기 위하여 잠시 멈추는 현상을 줄인다

-XX:CMSInitiatingOccupancyFraction

만일 이 값이 60이면 Old 영역의 60%가 사용되는 시점에 Concurrent GC 작업이 시작된다. 만일 이 값이 너무 크면 제 때 GC가 이루어지지 못해 Mark and Sweep GC가 작동할 확률이 높아진다. 반면 이 값이 너무 작으면 Concurrent GC 작업이 너무 일찍 시작해서 Application Thread의 CPU 점유를 방해한다. 따라서 처리량(Throughput)이 낮아질 수 있다.

-XX:MaxTenuringThreshold

default값은 31로, eden 영역의 객체가 survivor영역으로 옮겨져서 최종적으로 old영역으로 옮겨지기까지 최대로 복사를 31번까지 할 수 있다는 것을 의미한다. 0으로 하게 되면 바로 old영역으로 옮겨가게 되어 복사되는 시간을 줄이게 된다.

디버그옵션

-XX:-PrintGCDetails

GC 발생시 Heap 영역에 대한 비교적 상세한 정보를 추가적으로 기록한다. 추가적인 정보는 {GC 전 후의 Young/Old Generation의 크기, GC 전 후의 Permanent Generation의 크기, GC 작업에 소요된 시간} 등이다.
Minor GC가 발생한 경우 PrintGCDetails 옵션의 적용 예는 아래와 같다.
[GC [DefNew: 64575K->959K(64576K), 0.0457646 secs] 196016K->133633K(261184K), 0.0459067 secs]]
위의 로그가 의미하는 바는 다음과 같다.
GC 전의 Young Generation Usage = 64M, Young Generation Size = 64M
GC 전의 Total Heap Usage = 196M, Total Heap Size = 260M
GC 후의 Young Generation Usage = 9.5M
GC 후의 Total Heap Usage = 133M
Minor GC 소요 시간 = 0.0457646 초
Major GC가 발생한 경우 PrintGCDetails 옵션의 적용 예는 아래와 같다.
111.042: [GC 111.042: [DefNew: 8128K->8128K(8128K), 0.0000505 secs]
111.042: [Tenured: 18154K->2311K(24576K), 0.1290354 secs] 26282K->2311K(32704K), 0.1293306 secs]
위의 로그는 Minor GC 정보 외에 다음과 같은 Major GC 정보를 제공한다.
GC 전의 Tenured Generation Usage = 18M, Tenured Generation Size = 24M
GC 후의 Tenured Generation Usage = 2.3M
Major GC 소요시간 = 0.12초
(참고) PrintGCDetails + PrintGCTimeStamps 옵션의 조합이 가장 보편적으로 사용된다.

-XX:-PrintGCTimeStamps

GC가 발생한 시간을 JVM의 최초 구동 시간 기준으로 기록한다.

-XX:+PrintHeapAtGC

GC 발생 전후의 Heap에 대한 정보를 상세하게 기록한다

-XX:HeapDumpPath=./java_pid<pid>.hprof

HeapDump에 대한 파일명과 경로를 지정한다.
(1.4.2_12, 5.0 update 7 이상에서 사용 가능하다.)

-XX:-HeapDumpOnOutOfMemoryError
OutOfMemory가 났을 경우 HeapDump를 생성한다.
(1.4.2_12, 5.0 update 7 이상에서 사용 가능하다.)

-XX:OnOutOfMemoryError="<cmd args>"
OutOfMemory가 발생했을 경우 특정 command를 수행한다.
(1.4.2_12 이상과 jdk6에서 사용 가능하다.)

기타옵션

-Xrs
OS Signal사용을 최소화한다. 가령 이 옵션을 켜면 kill -3 [pid] 명령을 수행해도 Thread dump가 발생하지 않는다.

-XX:MaxGCPauseMillis
GC에 의한 Pause Time을 <value>ms 이하가 되게끔 Heap 크기와 기타 옵션들을 자동으로 조정하는 기능을 한다.

-XX:+MaxFDLimit
JVM의 File Descriptoer값을 OS의 max값으로 설정한다.

-XX:+ScavengeBeforeFullGC
Full GC 하기 이전에 Minor GC를 수행한다.

-XX:+UseAltSigs
Solaris 운영체제에서 VM은 디폴트로 SIGUSR1을 사용한다.
이것은 SIGUSR1을 시그널 체인(signal-chain) 하는 애플리케이션 과 충돌하는 경우 이 옵션은 default로 SIGUSR2 사용하도록 할 수 있다.

-XX:+UseBoundThreads
Solaris에서 제공하는 kernel단의 thread를 사용하는 옵션으로, CPU resource가 부족하여 thread들이 더 이상 진행되지 못하는 thread 부족현상을 감소시킬 수 있다.

-XX:-UseParallelOldGC
Old영역에 대해 ParallelGC를 수행한다. (JDK6에서 포함)

-XX:-UseSerialGC

JDK 1.4에서는 이 Collector가 기본 설정이다. JDK 5.0에서는 만일 JVM이 실행되는 환경이 Server 급이라면 Throughput Garbage Collector가 기본 설정이 된다.

-XX:ThreadStackSiz

Solaris Thread Stack Size를 설정한다.(kbyte)

-XX: TargetSurvivorRatio=50

객체가 survivor영역에서 Old 영역으로 이동하기 전까지 survivor영역이 사용되어야 하는 퍼센트 수치

4. 분석 기술

GC알고리즘의 이해

SUN Hot-Spot JDK에서는, 아래와 같이 주요 4가지 Garbage Collection 옵션을 제공한다.
이들은 아래와 같은 설정으로 설정이 가능하다.

1. Default garbage collector

구분	설명
옵션	-XX:+UseSerialGC
설명	JDK1.4에서는 default이다. JDK 5에서는 서버급(2CPU,2G메모리)일 경우에는 Parallel GC가 선택된다. 전통적인 GC방법으로 Minor GC에 Scavenge를, Full GC에 Mark & compact 알고리즘을 사용하는 방법이다

2. Parallel garbage collector

구분	설명
옵션	-XX:+UseParallelGC
설명	JDK 1.4 부터 지원되는 Parallel GC 는 Minor GC 를 동시에 여러 개의 Thread 를 이용해서 GC 를 수행하는 방법으로 하나의 Thread 를 이용하는 것보다 훨씬 빨리 GC 를 수행할 수 있다. 이때 반드시 고려되어야 하는 것이 CPU 개수이다. 시스템적으로 계산하기 위해 CPU 를 사용하기 때문에 최소 4CPU 이상에서만 최적의 성능을 보인다. Parallel GC 는 두 가지로 나뉘는데,Low-pause 방식과 Throughput 방식이다. Solaris 에서는 Low-pause Parallel GC 는 -XX:+UseParNewGC 옵션을 사용한다. Old 영역을 GC 할 때 Concurrent GC 방법과 함께 사용할 수 있다. 이 방법은 GC 가 일어날 때 빨리 GC 하는 것이 아니라 GC 가 발생할 때 Application 이 멈추는 현상을 줄이는데 초점이 맞춰져 있다. Throughput 방식의 Parallel GC 는 -XX:+UseParallelGC (Solaris 기준) 옵션을 이용하며 Old 영역을 GC 할 때는 Default GC (Mark and compact)방법만을 사용하도록 되어 있다.Minor GC 가 발생했을 때, 되도록이면 빨리 수행하도록 하는데 초점이 있다. 또한 동시에 몇 개의 Thread 를 이용하여 Minor 영역을 Parallel GC 할지를 결정할 수 있는데, -XX:ParallelGCThreads= 옵션을 사용하여 수를 지정할 수 있다.

3. Concurrent garbage collector

구분	설명
옵션	-XX:+UseConcMarkSweepGC
설명	Full GC 즉 Old 영역을 GC 하는 경우에는 그 시간이 길고 Application 이 순간적으로 멈춰버리기 때문에, 시스템 운용에 문제가 된다. 그래서 JDK1.4 부터 제공하는 Concurrent GC 는 기존의 이런 Full GC 의 단점을 보완하기 위해서 Full GC 에 의해서 Application 이 멈추어 지는 현상을 최소화 하기 위한 GC 방법이다. Full GC 에 소요되는 작업을 Application 을 멈추고 진행하는 것이 아니라, 일부는 Application 이 돌아가는 단계에서 수행하고, 최소한의 작업만을 Application 이 멈췄을 때 수행하는 방법으로 Application 이 멈추는 시간을 최소화한다. Application 이 수행 중 일 때 Full GC 를 위한 작업을 수행한다. (Sweep,mark) Application 을 멈추고 수행하는 작업은 일부분 (initial-mark, remark 작업)만을 수행하기 때문에, 기존 Default GC 의 Mark & Sweep Collector 에 비해서 Application 이 멈추는 시간이 현저하게 줄어든다. Solaris JVM 에서는 -XX:+UseConcMarkSweepGC Parameter 를 이용해 세팅한다.

GC 로그 출력 과 이해

-XX:+PrintGCDetails -XX:+PrintGCTimeStamps
-XX:+PrintGCApplicationStoppedTime 옵션을 적용하여 Collector 별 로그를 분석해 보도록 한다.

일반적으로 GC 로그는 다음과 같다.
"GC 종류종류: 이전사용메모리 -> GC 이후 사용메모리 (전체사이즈), 시간"
"DefNew: 18175K->1983K(18176K), 0.0357371 secs" 는 다음과 같은 사실을 의미한다.

New Generation 에 대해 Serial GC 를 수행했으며 GC 전에는 18M(18175K)
를 쓰고 있었고, GC 이후 1.9M(1983K)를 쓰고 있다.
New Generation 의 전체 크기는 18M(18176K) 이다.
GC 를 수행하는 데는 0.035 초 정도 걸렸다.

1. Default GC Log

0.109: [GC 0.110: [DefNew: 104K->64K(2112K), 0.0042610 secs]0.114: [Tenured: 18K->82K(1408K), 0.0088914 secs] 104K->82K(3520K), 0.0133819 secs]
Total time for which application threads were stopped: 0.0147430 seconds
JDK1.4 에서는 default 로 사용하고 있는 GC 로 위와 같은 형태로 나오게 된다.

2. Parallel GC Log

3.303: [GC [PSYoungGen: 15168K->3712K(15744K)] 251928K->244204K(257728K), 0.0386683 secs]
3.341: [Full GC [PSYoungGen: 3712K->0K(15744K)] [PSOldGen: 240492K->95306K(241984K)] 244204K->95306K(257728K) [PSPermGen: 1836K->1836K(16384K)], 0.4015217 secs]
Serial Collector 에서 DefNew 로 표기되었던 것이 PSYoungGen 으로 표기된다. Throughput Collector 의 개념을 이해했다면 그 차이를 정확하게 알 수 있을 것이다.

3. Concurrent GC Log

9.936: [GC [1 CMS-initial-mark: 173808K(253952K)] 174918K(262080K), 0.0011353 secs]
Total time for which application threads were stopped: 0.0029149 seconds
9.937: [CMS-concurrent-mark-start]
9.956: [GC 9.956: [ParNew: 8064K->0K(8128K), 0.0216583 secs] 181872K->176390K(262080K), 0.0217720 secs]
...
10.036: [GC 10.036: [ParNew: 8064K->0K(8128K), 0.0200405 secs] 187044K->181580K(262080K), 0.0201724 secs]
10.060: [CMS-concurrent-mark: 0.057/0.123 secs]
10.060: [CMS-concurrent-preclean-start]
10.061: [CMS-concurrent-preclean: 0.000/0.000 secs]
...
10.189: [GC[YG occupancy: 8064 K (8128 K)]10.190: [Rescan (parallel) , 0.0111151 secs]
10.201: [weak refs processing, 0.0000221 secs] [1 CMS-remark: 191976K(253952K)] 200040K(262080K), 0.0113344 secs]
...
10.663: [CMS-concurrent-sweep: 0.256/0.461 secs]
10.663: [CMS-concurrent-reset-start]
10.670: [CMS-concurrent-reset: 0.007/0.007 secs]

Serial Collector 에서 DefNew 로 표기되었던 것이 ParNew 로 표기되는 것에 주의한다.
-XX:+UseParNewGC 옵션이 작동한 결과이다. CMS 로 시작하는 부분은 initial mark->concurrent mark -> remark -> concurrent sweep 으로 이어지는 Full GC 의 활동 상황이 기록된 것이다.

jstat 을 이용한 실시간 메모리 영역 모니터링

JDK5 에서 추가된 jstat 를 사용하여 다른 jvm 의 실시간 메모리 상태를 보는 방법이 있는데 따로 GC 옵션을 걸지 않았을 경우에 유용하게 사용될 수 있다.
우선 jps 명령어로 pid 를 알아낸다.

```
jps |grep EngineContainerBootstrapper
21778 EngineContainerBootstrapper
jstat -gc -h3 21778 1s
S0C S1C S0U S1U EC EU OC OU PC PU YGC YGCT FGC FGCT GCT
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
S0C S1C S0U S1U EC EU OC OU PC PU YGC YGCT FGC FGCT GCT
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
384.0 448.0 0.0 0.0 1047680.0 66763.1 1024000.0 18322.2 45056.0 42329.6 399 105.318 125 91.938 197.256
```

-gc 옵션 이외에도 상세상태를 볼 수 있는 옵션을 제공한다.
-h3 는 header 를 3 번마다 출력하는 옵션이고 1s 는 1 초마다 출력을 하는 옵션이다.

헤더별 의미는 다음과 같다.
S0, S1: survivor 의 두 영역
E: Eden 영역
O: Old 영역
P: Perm 영역
YGC: 마지막 Young GC Count(횟수)
YGCT: 마지막 Young GC Time(시간)
FGC: 마지막 Full GC Count(횟수)
FGCT: 마지막 Full GC Time(시간)
GCT: (YoungGC + FullGC)마지막 GC Time

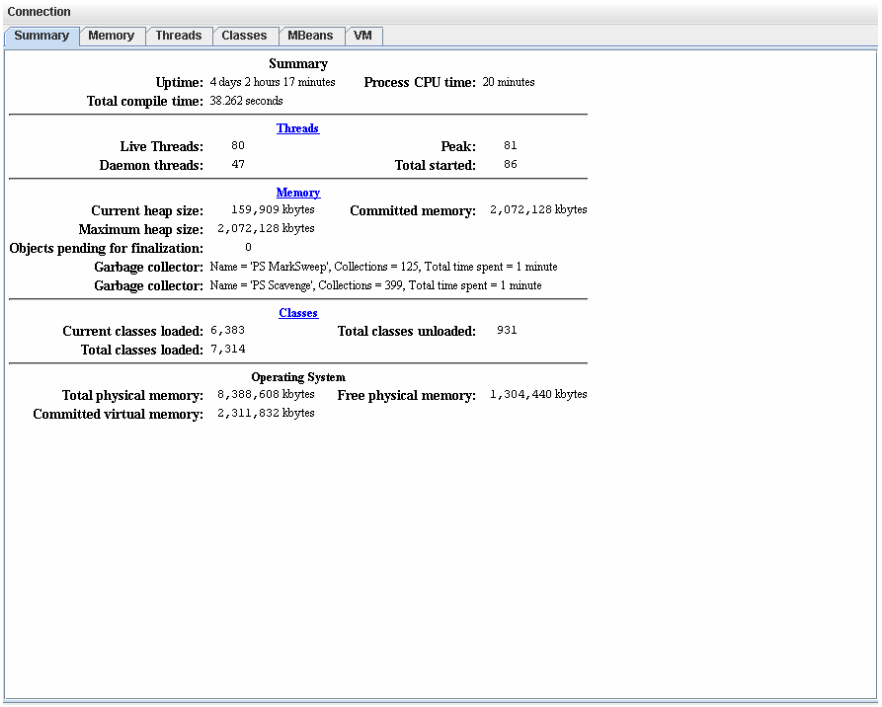
jconsole 을 이용한 실시간 GUI 모니터링

이번에도 JDK5 에서 제공하는 jconsole 을 이용하여 메모리 상태를 실시간으로 보는 방법에 대해 알아본다.
jconsole 은 jmxremote 로 통신을 하므로 JEUSMain.xml 의 <command-option>에 -Dcom.sun.management.jmxremote 가 추가되어야 한다.

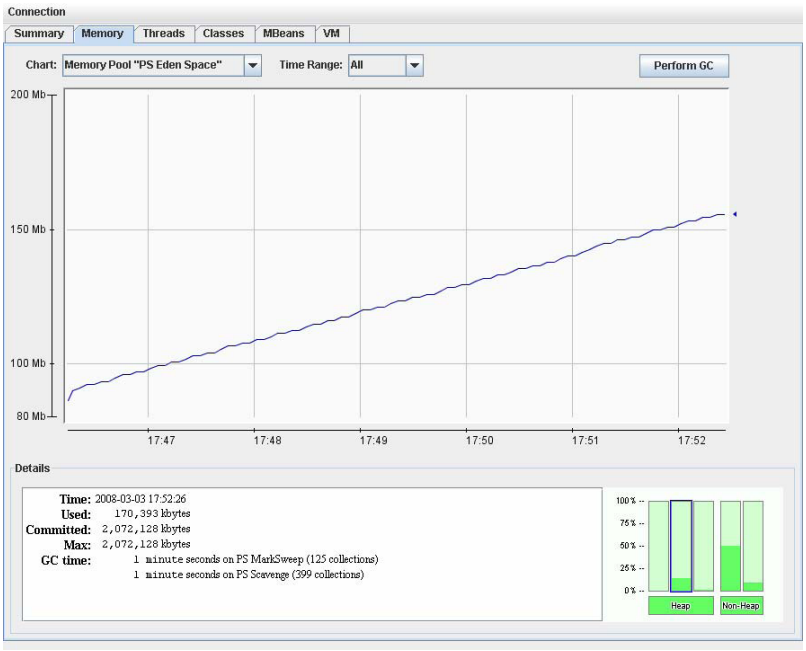
windows 에서는 직접 사용이 가능하나, Unix 환경에서는 XManager 를 사용하여야 구동이 가능하다. 위 그림과 같이 XManager 의 XStart 를 구동한다.

구동을 한 후 다음과 같이 display 세팅을 해준다.

export DISPLAY=로컬 PC 아이피:0.0



Memory tab 을 누르면 아래와 같이 Eden, Survivor, Old, Perm 영역들의 세부 내역을 GUI 로 볼 수 있게 된다. 또한 Thread 탭에서는 덤프를 생성하지 않고서도, 각 Thread 의 실시간 Trace 를 확인할 수 있다. 단 속도가 느리다는 것이 단점이다.



Heap Dump 분석

1. Heap Dump 생성

HeapDump 를 생성하는 방법은 다음의 4 가지가 있다.

1. JDK1.4.2_12,JDK5.0 update7 에서부터 가능한 -XX:+HeapDumpOnOutOfMemoryError 옵션을 적용하게 되면 OutOfMemory 가 발생했을 경우 HeapDump 를 생성하게 된다. default 로는 java_pid<pid>.hprof 이고 -XX:HeapDumpPath 로 경로를 지정할 수 있다.

2. JDK1.4.2_12,JDK5.0 update14 에서부터 가능한 -XX:+HeapDumpOnCtrlBreak 을 적용하면 Ctrl+ Break 나 SIGQUIT(kill -3) signal 을 받았을 경우 HeapDump 를 생성한다.

3. JDK5.0 에서 추가된 옵션으로 -agentlib:hprof=file=dump.hprof,format=b 를 JVM 실행 시에 추가한다.

4. JDK1.4.2_09 에서 추가된 jmap 을 이용하여 다음과 같이 덤프를 생성한다.

jmap -heap:format=b PID_OF_PROCESS (jdk5)

jmap -dump:format=b,file=snapshot2.jmap PID_OF_PROCESS (jdk6)

2. hat 으로 **Heap Dump** 분석

이렇게 생성된 파일을 분석하기 위해서는 jdk6 에 포함되어있는 jhat 를 사용하여야 하거나, jdk5 는 별도로 download(http://hat.dev.java.net)를 받아야 한다.

우선 위의 옵션들 중 하나를 선택하여 dump 를 생성한다. 로컬이나 서버에서 hat 을 다운받아 설치하거나 jdk6 에 있는 jhat 을 이용하여 실행한다.

jhat dump.hprof

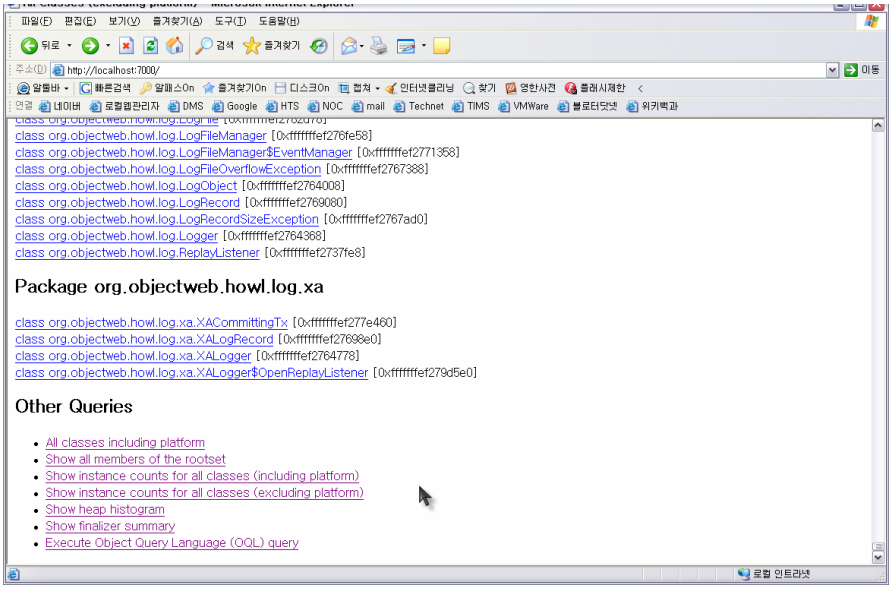
...

...

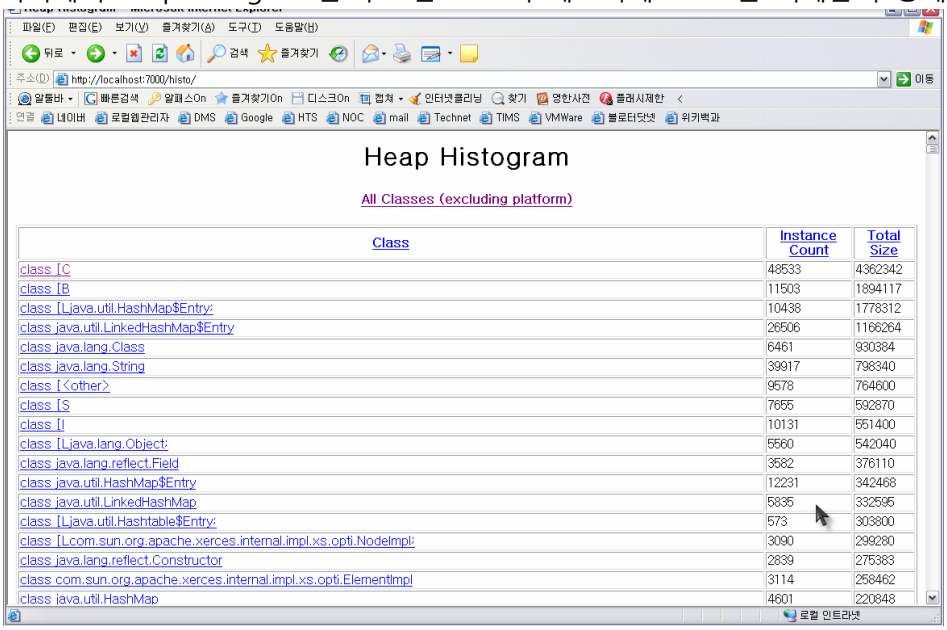
Started HTTP server on port 7000

Server is ready.

위와 같이 하면 분석은 모두 끝난 상태이고, 결과는 위의 7000 번 포트를 웹브라우저로 접근하여 볼 수 있다.



여기에서 heap histogram 을 누르면 JVM 의 메모리에 로드 된 객체들의 통계를 볼 수 있다.



Class	Instance Count	Total Size
class [C	48533	4362342
class [B	11503	1894117
class [Ljava.util.HashMap$Entry;	10438	1778312
class java.util.LinkedHashMap$Entry	26506	1166264
class java.lang.Class	6461	930384
class java.lang.String	39917	798340
class [<other>	9578	764600
class [S	7655	592870
class [I	10131	551400
class [Ljava.lang.Object;	5560	542040
class java.lang.reflect.Field	3582	376110
class java.util.HashMap$Entry	12231	342468
class java.util.LinkedHashMap	5835	332595
class [Ljava.util.Hashtable$Entry;	573	303800
class [Lcom.sun.org.apache.xerces.internal.impl.xs.opti.NodeImpl;	3090	299280
class java.lang.reflect.Constructor	2839	275383
class com.sun.org.apache.xerces.internal.impl.xs.opti.ElementImpl	3114	258462
class java.util.HashMap	4601	220848

3. IBM ha 로 Heap Dump 분석

IBM 의 HA 로도 분석이 가능한데, 아래의 주소에서 다운로드 받을 수 있다.

http://www.alphaworks.ibm.com/tech/heapanalyzer/download/

Copyright © 2013 TmaxSoft Co., Ltd. All Rights Reserved. TmaxSoft Co., Ltd.

Trademarks
Tmax, WebtoB, WebT, JEUS, ProFrame, SysMaster and OpenFrame are registered trademarks of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Contact Information
TmaxSoft can be contacted at the following addresses to arrange for a consulting team to visit your company and discuss your options for legacy modernization.

Korea – TmaxSoft Co., Ltd.
Corporate Headquarters
272-6 Seohyeon-dong, Bundang-gu, Seongnam-si, South Korea, 463-824
Tel : (+82) 31-8018-1708 Fax : (+82) 31-8018-1710
Website : http://tmaxsoft.com

U.S.A. – TmaxSoft Inc.
560 Sylvan Avenue Englewood Cliffs, NJ 07632, USA
Tel : (+1) 201-567-8266 Fax : (+1) 201-567-7339
Website : http://us.tmaxsoft.com

Japan – TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073 Japan
Tel : (+81) 3-5765-2550 Fax: (+81) 3-5765-2567
Website : http://jp.tmaxsoft.com

China – TmaxSoft China Co., Ltd.
Room 1101, Building B, Recreo International Center, East Road Wang Jing, Chaoyang District, Beijing, 100102, P.R.C
Tel : (+86) 10-5783-9188 Fax: (+86) 10-5783-9188(#800)
Website : http://cn.tmaxsoft.com

China(JV) – Upright(Beijing) Software Technology Co., Ltd
Room 1102, Building B, Recreo International Center, East Road Wang Jing, Chaoyang District, Beijing, 100102, P.R.C
Tel : (+86) 10-5783-9188 Fax: (+86) 10-5783-9188(#800)
Website : www.uprightsoft.com

TD-JSTN-C1101016