**1) JVM 메모리 운영**

  - Heap = Young Generation + Old Generation + Permanent Generation(PermSize=MethodArea)

    + Young 에서 신규 생성되어 오래된 객체가 Old로 넘어감

    + Young 에서 바로 사용되지 않으면 Minor GC

    + Old로 넘어간 후 Old에서 더 이상 사용되지 않으면 Full GC

    + Perm 영역은 상수영역으로 GC 대상은 아니나 간혹 일어남

  - Young Generation = Eden + Survivor (2개)  로 구성

    + Eden 에서 사용되어진 것은 Survivor로 넘어가고 다시 Old 영역으로 넘겨진다

**2) jstat 명령을 통한 GC 모니터링**

  - $JDK\_HOME/bin/jstat 명령어 이용

  - jstat 옵션들 : 출처 (<http://helloworld.naver.com/helloworld/helloworld/6043>)

|  |  |
| --- | --- |
| **옵션** | **기능** |
| gc | 각 힙(heap) 영역의 현재 크기와 현재 사용량(Eden 영역, Survivor 영역, Old 영역등), 총 GC 수행 횟수, 누적 GC 소요 시간을 보여 준다. |
| gccapactiy | 각 힙 영역의 최소 크기(ms), 최대 크기(mx), 현재 크기, 각 영역별 GC 수행 횟수를 알 수 있는 정보를 보여 준다. 단, 현재 사용량과 누적 GC 소요 시간은 알 수 없다. |
| gccause | -gcutil 옵션이 제공하는 정보와 함께 마지막 GC 원인과 현재 발생하고 있는 GC의 원인을 알 수 있는 정보를 보여 준다. |
| gcnew | New 영역에 대한 GC 수행 정보를 보여 준다. |
| gcnewcapacity | New 영역의 크기에 대한 통계 정보를 보여 준다. |
| gcold | Old 영역에 대한 GC 수행 정보를 보여 준다. |
| gcoldcapacity | Old 영역의 크기에 대한 통계 정보를 보여 준다. |
| gcpermcapacity | Permanent 영역에 대한 통계 정보를 보여 준다. |
| gcutil | 각 힙 영역에 대한 사용 정도를 백분율로 보여 준다. 아울러 총 GC 수행 횟수와 누적 GC 시간을 알 수 있다. |

  - gcutil / gccapcity 사용 예

    + gcutil : WAS 기동후 FGC(Full GC 건수)는  230회 발생하였고, FGCT(Full GC 수행시간)은 2302초로 거의 10초가 소요되었다

                 Full GC 수행마다 10초의 시간

    + gccapacity : New 영역 사용크기(NGCMN) 이 2,097,152 (즉 2Gbytes)이고 Old 영역 사용크기(OGCMN)가 3,145,728 (즉 3GBytes) 가량이 된다

[jboss@dowon ~]$ ps -ef |grep java

jboss    13090 13053  8 Mar27 ?        04:28:28 /usr/java/jdk1.6.0\_31/bin/java -Dprogram.name=run.sh -server .. 중략..

[jboss@dowon ~]$ **jstat -gcutil 13090**

  S0     S1     E        O       P     **YGC   YGCT    FGC   FGCT**      GCT

  0.76   0.00  85.57  86.24  26.88   **1260  644.013   230   2302.259**   2946.273

[jboss@dowon ~]$ **jstat -gccapacity 13090**

**NGCMN**NGCMX     NGC     S0C   S1C       EC      **OGCMN**OGCMX       OGC         OC      PGCMN    PGCMX     PGC       PC     YGC    FGC

**2097152.0** 2097152.0 2097152.0 430656.0 412096.0 1254400.0  **3145728.0**3145728.0  3145728.0  3145728.0 524288.0 524288.0 524288.0 524288.0   1260   230

  - 간단한  GC Check Shell Script 작성해 보았다

    + weblogic.Server (jboss: org.jboss.Main) 부분을 WAS나 JAVA Process 환경에 맞게 변경하여 사용한다

    + **WAS가 기동된 이후 Full GC 평균 시간 = FGCT/FGC**  : 만일 3초이상이면 점검 대상으로 지정

#!/bin/sh

# [GC Statistics after WAS started]

# config

#   interval : 3 sec

#   count : 2

echo ""

echo "[GC statistics after WAS is started]"

echo "----------------------------------------"

echo " S0/S1: Survivor 0/1 (KB)"

echo " E    : Eden (KB)"

echo " O    : Old area size (KB)"

echo " P    : Permanent area size (KB)"

echo " YGC  : Young Generation GC Count"

echo " YGCT : Young Generation GC Total Time"

echo " FGC  : Full GC Count"

echo " FGCT : Full GC Total Time"

echo " GCT  : GC Total Time"

echo "----------------------------------------"

echo ""

echo " HostName : " $(hostname)

echo ""

for PID in  **`ps -ef | grep java | egrep -v grep | grep weblogic.Server | awk -F" " '{print $2}'`**

do

echo " WAS PID  : " $PID

echo "----------------------------------------------------------------------------"

jstat -gcutil $PID 3000 2

echo "----------------------------------------------------------------------------"

echo ""

done

echo ""

결과

===========================================

 HostName :  jupitorDowon

 WAS PID  :  355343

----------------------------------------------------------------------------

  S0     S1     E      O      P     YGC     YGCT    FGC    FGCT     GCT

  0.00   0.00  52.68  13.52  31.58   3764   30.408  1878 1148.863 1179.271

  0.00   0.00  52.69  13.52  31.58   3764   30.408  1878 1148.863 1179.271

----------------------------------------------------------------------------

  - 컬럼 출력 내용 : 출처 (<http://helloworld.naver.com/helloworld/helloworld/6043>)

|  |  |  |
| --- | --- | --- |
| **칼럼** | **설명** | **jstat 옵션** |
| S0C | Survivor 0 영역의 현재 크기를 KB 단위로 표시 | -gc  -gccapacity  -gcnew  -gcnewcapacity |
| S1C | Survivor 1 영역의 현재 크기를 KB 단위로 표시 | -gc  -gccapacity  -gcnew  -gcnewcapacity |
| S0U | Survivor 0 영역의 현재 사용량을 KB 단위로 표시 | -gc  -gcnew |
| S1U | Survivor 1 영역의 현재 사용량을 KB 단위로 표시 | -gc  -gcnew |
| EC | Eden 영역의 현재 크기를 KB 단위로 표시 | -gc  -gccapacity  -gcnew  -gcnewcapacity |
| EU | Eden 영역의 현재 사용량을KB 단위로 표시 | -gc  -gcnew |
| OC | Old 영역의 현재 크기를 KB 단위로 표시 | -gc  -gccapacity  -gcold  -gcoldcapacity |
| OU | Old 영역의 현재 사용량을KB 단위로 표시 | -gc  -gcold |
| PC | Permanent영역의 현재 크기를 KB 단위로 표시 | -gc  -gccapacity  -gcold  -gcoldcapacity  -gcpermcapacity |
| PU | Permanent영역의 현재 사용량을KB 단위로 표시 | -gc  -gcold |
| YGC | Young Generation의 GC 이벤트 발생 횟수 | -gc  -gccapacity  -gcnew  -gcnewcapacity  -gcold  -gcoldcapacity  -gcpermcapacity  -gcutil  -gccause |
| YGCT | Yong Generation의 GC 수행 누적 시간 | -gc  -gcnew  -gcutil  -gccause |
| FGC | Full GC 이벤트가 발생한 횟수 | -gc  -gccapacity  -gcnew  -gcnewcapacity  -gcold  -gcoldcapacity  -gcpermcapacity  -gcutil  -gccause |
| FGCT | Full GC 수행 누적 시간 | -gc  -gcold  -gcoldcapacity  -gcpermcapacity  -gcutil  -gccause |
| GCT | 전체 GC 수행 누적 시간 | -gc  -gcold  -gcoldcapacity  -gcpermcapacity  -gcutil  -gccause |
| NGCMN | New Generation의 최소 크기를 KB단위로 표시 | -gccapacity  -gcnewcapacity |
| NGCMX | New Generation의 최대 크기를 KB단위로 표시 | -gccapacity  -gcnewcapacity |
| NGC | New Generation의 현재 크기를 KB단위로 표시 | -gccapacity  -gcnewcapacity |
| OGCMN | Old Generation의 최소 크기를 KB단위로 표시 | -gccapacity  -gcoldcapacity |
| OGCMX | Old Generation의 최대 크기를 KB단위로 표시 | -gccapacity  -gcoldcapacity |
| OGC | Old Generation의 현재 크기를 KB단위로 표시 | -gccapacity  -gcoldcapacity |
| PGCMN | Permanent Generation의 최소 크기를 KB단위로 표시 | -gccapacity  -gcpermcapacity |
| PGCMX | Permanent Generation의 최대 크기를 KB단위로 표시 | -gccapacity  -gcpermcapacity |
| PGC | 현재 Permanent Generation의 크기를 KB단위로 표시 | -gccapacity  -gcpermcapacity |
| PC | Permanent 영역의 현재 크기를 KB단위로 표시 | -gccapacity  -gcpermcapacity |
| PU | Permanent 영역의 현재 사용량을 KB단위로 표시 | -gc  -gcold |
| LGCC | 지난 GC의 발생 이유 | -gccause |
| GCC | 현재 GC의 발생 이유 | -gccause |
| TT | Tenuring threshold. Young 영역 내에서 이 횟수만큼 복사되었을 경우(S0 ->S1, S1->S0) Old 영역으로 이동 | -gcnew |
| MTT | 최대 Tenuring threshold. Yong 영역 내에서 이 횟수만큼 복사되었을 경우 Old 영역으로 이동 | -gcnew |
| DSS | 적절한 Survivor 영역의 크기를 KB단위로 표시 | -gcnew |

**3) verbose:gc 옵션을 통한 GC 모니터링**

  - WAS 기동 Shell에 verbose:gc 옵션을 준다

    + GC에 대해 자세히 시간을 찍도록 한다

    + 로그위치와 명칭을 지정한다

-verbose:gc

-Xloggc:/applog/cmsmDomain/gclog/gc.log

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintHeapAtGC

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=/applog/cmsmDomain/gclog/java\_pid.hprof

  - 로그 출력 예

**// Full GC**

7210.826: [Full GC (System) [PSYoungGen: 23616K->0K(1947520K)] [PSOldGen: 886563K->856929K(3145728K)] 910179K->856929K(5093248K) [PSPermGen: 138064K->138064K(524288K)], 2.9391980 secs] [Times: user=2.98 sys=0.00, real=2.94 secs]

Heap after GC invocations=52 (full 3):

 PSYoungGen      total 1947520K, used 0K [0x0000000780000000, 0x0000000800000000, 0x0000000800000000)

  eden space 1801536K, 0% used [0x0000000780000000,0x0000000780000000,0x00000007edf50000)

  from space 145984K, 0% used [0x00000007edf50000,0x00000007edf50000,0x00000007f6de0000)

  to   space 141376K, 0% used [0x00000007f75f0000,0x00000007f75f0000,0x0000000800000000)

 PSOldGen        total 3145728K, used 856929K [0x00000006c0000000, 0x0000000780000000, 0x0000000780000000)

  object space 3145728K, 27% used [0x00000006c0000000,0x00000006f44d8428,0x0000000780000000)

 PSPermGen       total 524288K, used 138064K [0x00000006a0000000, 0x00000006c0000000, 0x00000006c0000000)

  object space 524288K, 26% used [0x00000006a0000000,0x00000006a86d41d8,0x00000006c0000000)

}

**// Minor GC**

10500.769: [GC [PSYoungGen: 1769535K->35007K(1928320K)] 2645761K->912640K(5074048K), 0.0958040 secs] [Times: user=0.17 sys=0.00, real=0.09 secs]

Heap after GC invocations=67 (full 3):

 PSYoungGen      total 1928320K, used 35007K [0x0000000780000000, 0x0000000800000000, 0x0000000800000000)

  eden space 1754816K, 0% used [0x0000000780000000,0x0000000780000000,0x00000007eb1b0000)

  from space 173504K, 20% used [0x00000007f5690000,0x00000007f78bff00,0x0000000800000000)

  to   space 168832K, 0% used [0x00000007eb1b0000,0x00000007eb1b0000,0x00000007f5690000)

 PSOldGen        total 3145728K, used 877632K [0x00000006c0000000, 0x0000000780000000, 0x0000000780000000)

  object space 3145728K, 27% used [0x00000006c0000000,0x00000006f59101d8,0x0000000780000000)

 PSPermGen       total 524288K, used 138386K [0x00000006a0000000, 0x00000006c0000000, 0x00000006c0000000)

  object space 524288K, 26% used [0x00000006a0000000,0x00000006a8724880,0x00000006c0000000)

}

**4) 튜닝하기**

  - WAS 기동 Shell의 기본옵션

    + -Xms -Xmx 힙사이즈 지정

    + -server 옵션 지정

**GC 튜닝 시 기본적으로 학인해야 하는 JVM 옵션 (출처:**<http://helloworld.naver.com/helloworld/37111>)

|  |  |  |
| --- | --- | --- |
| **구분** | **옵션** | **설명** |
| 힙(heap) 영역 크기 | -Xms | JVM 시작 시 힙 영역 크기 |
| -Xmx | 최대 힙 영역 크기 |
| New 영역의 크기 | -XX:NewRatio | New영역과 Old 영역의 비율 |
| -XX:NewSize | New영역의 크기 |
| -XX:SurvivorRatio | Eden 영역과 Survivor 영역의 비율 |

  - 설정 예

/usr/java/jdk1.6.0\_31/bin/java

-server

-Xms5120m -Xmx5120m

-XX:NewSize=2048m

-XX:MaxNewSize=2048m

-XX:PermSize=512m

-XX:MaxPermSize=512m

-Xss5m

-verbose:gc

-Xloggc:/applog/cmsmDomain/gclog/gc.log

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintHeapAtGC

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=/applog/cmsmDomain/gclog/java\_pid.hprof

  - 다음의 경우 튜닝이 필요없다. 단 이것은 서비스의 특성에 따라 틀리다. 해당 시간 이상이면 튜닝 대상

* Minor GC의 처리 시간이 빠르다(50ms내외).
* Minor GC 주기가 빈번하지 않다(10초 내외).
* Full GC의 처리 시간이 빠르다(보통1초 이내).
* Full GC 주기가 빈번하지 않다(10분에 1회).

  - Full GC 수행 시간과 수행 간격 조사 => Minor GC 수행 시간과 수행 간격 조사 => 전체 평균값 및 수행 횟수등을 조사하여 튠이 필요성을 판단한다

  - 메모리 설정 옵션을 조정하면서 결정한다

    + 힙사이즈가 크면 GC 시간도 올래 걸린다 : 그만큼 많은 객체가 쌓여 있을 것이므로

    + 따라서 무조건 크게 잡는다고 해서 좋은 것은 아니므로 적정사이즈를 찾아가는 것이 중요하다

    + NewRatio=2 는 New Generation:Old Generation = 1:2 비율 사이즈로 사용크기를 지정하는 것임 (-Xmx 사이즈에 대해서)

      즉, NewRatio=4 이면 New:Old=1:4 비율의 크기임

  - 튜닝 예 1)

    + jstat -gccapacity의 New영역 사용크기(NGCMN)과 Old영역 사용크기(OGCMN) 값의 비율을 체크한다

      NGCMN:OGCMN = 2:3 (Gbytes)

    + NewRatio=2 => 3 => 4 로 바꾸어 가면서 Full/Minor GC 평균응답시간을 구한다 (jstat -gctuil로 구함)

  - 튜닝 예 2)

    + verbose:gc 로그가 남으면 HP JMeter를 통하여 분석한다

    + JMeter를 통하여 Duration 발생 간격을 측정하거나 GC 소요 시간을 그래프 형태로 볼 수 있다.

    + 여러 옵션을 차례로 처음 몇개에만 적용해 보고 jstat -gcutil 을 통하여 통계값을 얻고 최적치를 측정해 나간다

* Case1 : -XX:+UseParallelGC -Xms1536m -Xmx1536m -XX:NewRatio=2
* Case2 : -XX:+UseParallelGC -Xms1536m -Xmx1536m -XX:NewRatio=3
* Case3 : -XX:+UseParallelGC -Xms1g -Xmx1g -XX:NewRatio=3
* Case4 : -XX:+UseParallelOldGC -Xms1536m -Xmx1536m -XX:NewRatio=2
* Case5 : -XX:+UseParallelOldGC -Xms1536m -Xmx1536m -XX:NewRatio=3
* Case6 : -XX:+UseParallelOldGC -Xms1g -Xmx1g -XX:NewRatio=3

출처: <http://mobicon.tistory.com/254> [Mobile Convergence]