

# [힙메모리 영역별 기준사용량 초과시 경고발생]

## 부 제 : Out of Memory 발생시 HeapDump생성

작 성: 김지훈([kjh@jennifersoft.com](mailto:kjh@jennifersoft.com))

제니퍼소프트 기술지원부

### ■ 문서개요

본 문서는 JAVA5에서부터 추가된 JVM레벨 모니터링 JMX인터페이스와 제니퍼4.0에서 새로 선보인 기능인 Extra Agent기능을사용해서 WAS(Web Application Server) 혹은 자바데몬(JAVA DAEMON)이 사용하는 힙메모리(각 메모리 영역별)가 지정된 Threshold를 초과했을 때 경고메세지를 발생하는 방법에 대해 소개합니다. (OutOfMemory발생시 힙덤프 생성 기능 소개 포함)

### ■ 순 서

1. Extra Agent 인터페이스 소개
2. Extra Agent 인터페이스 구현 예제소개
3. 구현 클래스 등록 및 설정방법
4. 데모실행 및 결과확인
5. -XX: HeapDumpOnOutOfMemoryError 옵션 소개 및 Memory Analysis 툴 소개
6. 맺음말

#### 1. Extra Agent 인터페이스 소개

Extra agent인터페이스는 제니퍼4.0에 새롭게 추가된 인터페이스로서 제니퍼에이전트가 기본적으로 제공하는 모니터링 데이터 이외에 사용자가 추가로 사용자 환경에 필요한 데이터를 수집, 사용자정의형 모니터링 화면을 구축할 수 있도록 합니다.

```
package com.javaservice.jennifer.agent;
```

```
import com.javaservice.jennifer.protocol.RmPacket;
```

```
public interface ExtraAgent {
```

```
    // process 메소드를 구현해야 합니다. Rmpacket을 주기적으로 리턴
```

```
    public RmPacket process(String agent);
```

```
    //static method인 ExtraAgentUtil.sendAlert()인 이벤트 발생시 필요에 따라 사용
```

```
    이번 예제는 ExtraAgentUtil.sendAlert()메소드를 통해 생성된 에러메세지를 제니퍼서버로 전달함
```

```
}
```

[Extra Agent인터페이스 소개]

## 2. Extra Agent 인터페이스 구현 예제

```
package jennifersoft.tech.support;

import java.io.IOException;
import java.lang.management.ManagementFactory;
import java.lang.management.MemoryMXBean;
import java.lang.management.MemoryNotificationInfo;
import java.lang.management.MemoryPoolMXBean;
import java.lang.management.MemoryUsage;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

import javax.management.MBeanServer;
import javax.management.Notification;
import javax.management.NotificationBroadcaster;
import javax.management.NotificationListener;
import javax.management.openmbean.CompositeData;

import com.javaservice.jennifer.agent.Config;
import com.javaservice.jennifer.agent.ExtraAgent;
import com.javaservice.jennifer.agent.ExtraAgentUtil;
import com.javaservice.jennifer.protocol.RmPacket;
import com.sun.management.HotSpotDiagnosticMXBean;

public class HeapMemAlert implements ExtraAgent {

    private final Map<String, MemoryPoolMXBean> POOLS = new HashMap<String,
MemoryPoolMXBean>();
    HotSpotDiagnosticMXBean bean = null;
    String enable_heapdump = "false";
    double heapsize = 0;

    Properties p = null;
```

```

public HeapMemAlert(){

    init();
}

public void init(){

    //구성파일에서 메모리 사용량 Threshold 값을 읽어옵니다.
    p = Config.getProperties();
    heapsize = Double.parseDouble(p.getProperty("heap_warning_rate"));

    MBeanServer server = ManagementFactory.getPlatformMBeanServer();
    try {
        bean =
            ManagementFactory.newPlatformMXBeanProxy(server,
                "com.sun.management:type=HotSpotDiagnostic",
HotSpotDiagnosticMXBean.class);
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }

    MemoryMXBean mem = ManagementFactory.getMemoryMXBean();
    ((NotificationBroadcaster)mem).addNotificationListener(listener, null, null);
    for (MemoryPoolMXBean memPool : ManagementFactory.getMemoryPoolMXBeans()) {
        if (memPool.isUsageThresholdSupported()) {
            POOLS.put(memPool.getName(), memPool);

            //각 메모리 영역별 Threshold 값을 설정합니다.
            setUsageThreshold(memPool, heapsize);
        }
    }
}

//이벤트 리스터를 등록합니다.
private final NotificationListener listener = new NotificationListener() {
    public void handleNotification(Notification notification, Object handback) {
        if
(MemoryNotificationInfo.MEMORY_THRESHOLD_EXCEEDED.equals(notification.getType())) {

```

```

        CompositeData cd = (CompositeData)notification.getUserData();
        MemoryNotificationInfo info = MemoryNotificationInfo.from(cd);
        MemoryUsage memUsage = info.getUsage();
        String poolName = info.getPoolName();

        /*
        메세지 헤더부분에 [WARNING, ERROR, FATAL로서 메세지
경고레벨을 설정합니다. 아무것도 사용하지 않을때는 정보로서 제니퍼가
인식하게 됩니다.
        */

        String message = "WARNING " + poolName.replace(" ", "_") + "영역
사용률" + (heapsize * 100) + "%를 초과하였습니다. " +
        "현재 사용률은 " + (memUsage.getUsed()/1024/1024) + "MB입니다." ;
        System.out.println(message);

        //제니퍼서버로 경고메세지 전달
        ExtraAgentUtil.sendAlert(message);

    }
}
};

public void setHeapDump(String enable_heapdump){

    //OutOfMemory발생시 힙덤프를 생성하도록 설정합니다.
    if(enable_heapdump.equals("true")){
        bean.setVMOption("HeapDumpOnOutOfMemoryError", "true");
        //System.out.println(bean.getVMOption("HeapDumpOnOutOfMemoryError"));

        //OutOfMemory발생시 힙덤프를 생성하지 않습니다.
    }else if(enable_heapdump.equals("false")){
        bean.setVMOption("HeapDumpOnOutOfMemoryError", "false");
        System.out.println(bean.getVMOption("HeapDumpOnOutOfMemoryError")
        );
    }
}
}

```

```

public void setUsageThreshold(MemoryPoolMXBean memPool, double percentage) {
    MemoryUsage memUsage = memPool.getUsage();
    long max = memUsage.getMax();
    memPool.setUsageThreshold((long)(max * percentage));
}

public RmPacket process(String agent) {

    //구성파일에서 힙덤프생성여부
    enable_heapdump = p.getProperty("enable_heapdump");

    if(enable_heapdump.equals("true") &&
bean.getVMOption("HeapDumpOnOutOfMemoryError").getValue().equals("false")){

        setHeapDump("true");

    }else if(enable_heapdump.equals("false") &&
bean.getVMOption("HeapDumpOnOutOfMemoryError").getValue().equals("true")){

        setHeapDump("false");

    }

    return null;
}
}

```

[구현예제:HeapMemAlert.java]

### 3. 구현 클래스 등록 및 설정방법

#### ① 구현클래스 등록

생성된 클래스를 heapalert.jar로 묶어 jennifer.jar클래스 위치와 동일한 디렉토리 혹은 PATH상에 복사 혹은 설정합니다.

#### ② 제니퍼서버 설정

**extra\_enable=true**

→ Extar Agent기능을 활성화 합니다.

**extra\_agent\_class=jennifersoft.tech.support.HeapMemAlert**

→ 앞서 작성된 클래스를 등록합니다.

**#extra\_data\_interval=2000**

→ 메시지 전달주기를 설정합니다. 그러나 이번 예제는 이벤트 발생시 경고메세지를 전달하기 때문에 해당 옵션을 비활성화 했습니다.

**enable\_heapdump=true**

→ **Threshold**초과시 힙덤프를 생성할 수 옵션이ダイナミック하게 반영되도록 설정합니다.

**heap\_warning\_rate=0.9**

→ 각 메모리영역별 **Threshold**값을 설정합니다.(이번 예제는 **90%**을 초과했을 경우 메시지를 발생시킵니다.)

**#extra\_data\_send\_target=127.0.0.1:7701**

→ 레몬서버를 사용하지 않으면(비활성화) 직접 제니퍼서버로 해당 데이터를 전송합니다.

③ 제니퍼에이전트가 설치된 **WAS(Web Application Server)** 혹은 **JAVA DAEMON**을 새로 시작합니다.

#### 4. 데모실행 및 결과 확인

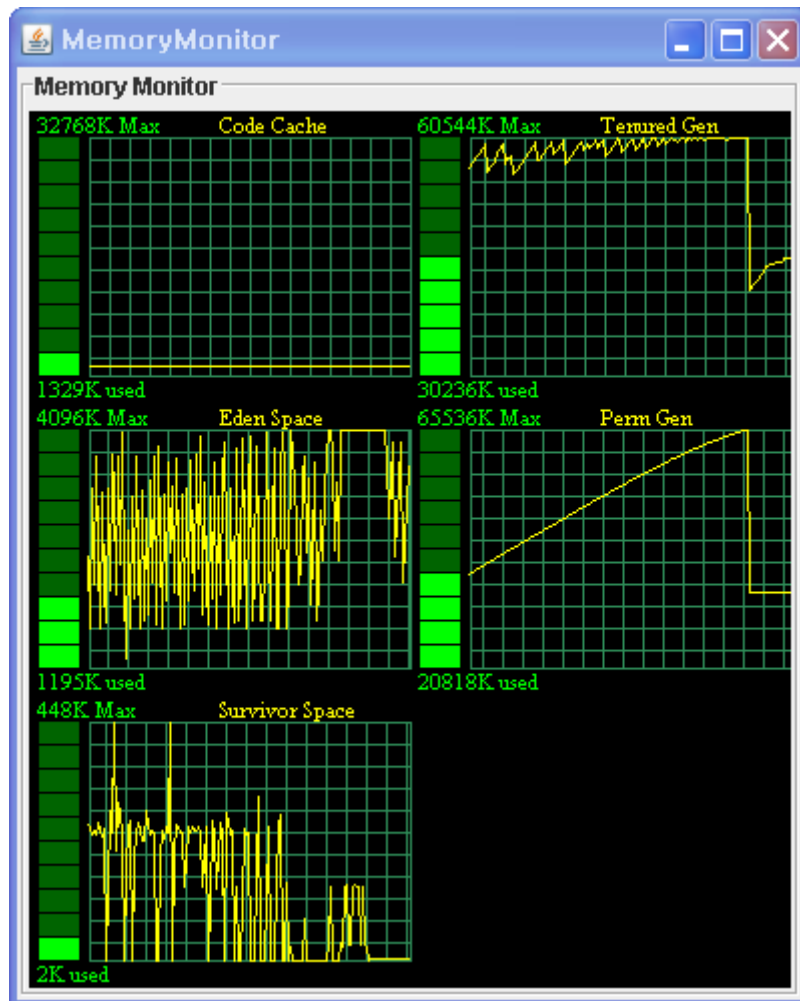
##### ① 데모 대상 프로그램 소개

수집되는 데이터 값의 비교를 위해 제니퍼에이전트를 JAVA 6버전의 데모 프로그램인 MemoryMonitor에 설치해서 진행했습니다.

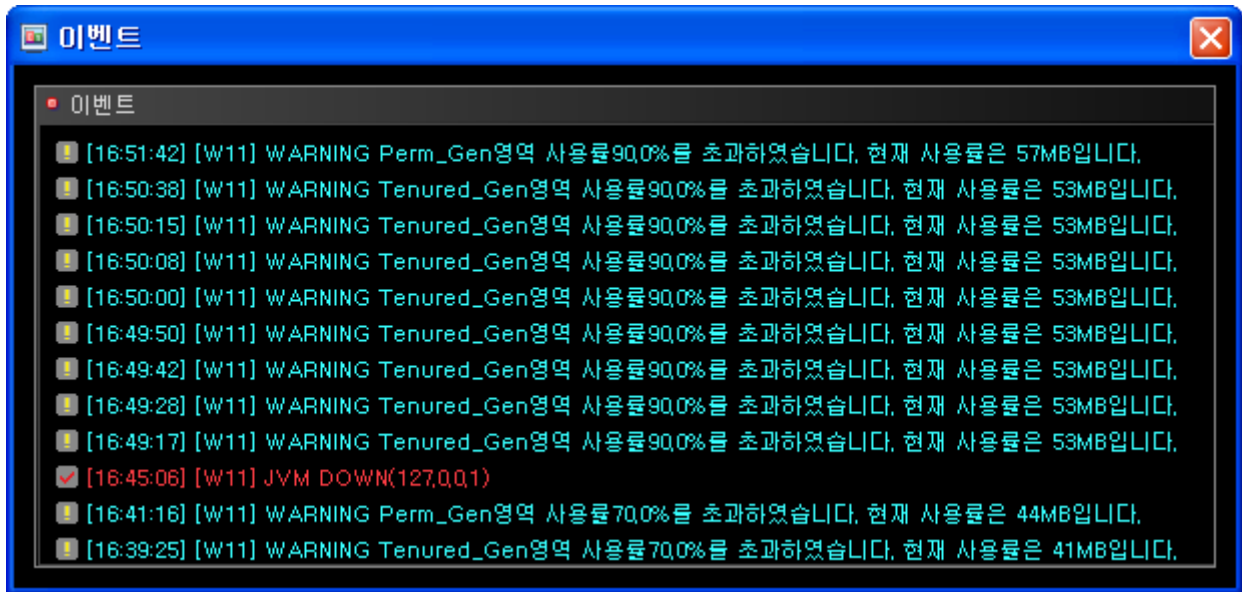
([JAVA\_HOME]/demo/management/MemoryMonitor/MemoryMonitor.jar

##### [주의]

-server 옵션을 주고 실행했습니다. Default인 client와 생성되는 메모리 구조가 일부 다른 Client를 기동할 경우 Perm Gen 영역에서 [shared-ro]와 [shared-rw]영역을 확인할 수 있습니다.



[그림 1 : Tenured Gen 및 Perm Gen영역이 증가되는 것 확인 - Out of Memory발생]



[그림 2: 지정된 Threshold를 초과함에 따라 경고메세지 발생(Tenured\_Gen, Perm\_Gen)]

[SystemOut 로그내용]

```

java.lang.OutOfMemoryError: PermGen space
Dumping heap to java_pid6716.hprof ...
Heap dump file created [82258354 bytes in 4.309 secs]
  
```

: PermGen 영역 부족과 함께 OutofMemory발생

HeapDumpOnOutOfMemoryError설정에 따라 Out of Memory발생시 힙 덤프 생성

## 5. -XX: HeapDumpOnOutOfMemoryError 옵션 소개 및 Eclipse Memory Analyzer 툴 소개

### ① HeapDumpOnOutOfMemoryError 소개

[Sun JDK 1.4.2\_12, 1.5.0\_7, 그리고 HP 1.4.2\_11] 이후 버전부터는 JVM OPTIONS에 HeapDumpOnOutOfMemoryError를 설정하면 Out of Memory발생시 힙덤프를 생성할 수 있다.

(추가옵션 : - -XX:HeapDumpPath=/path 힙덤프 생성 위치 지정)

※ 참고로 이번 예제에서는 JAVA6버전에서 소개된 HotSpotDiagnostic MBean을 사용하여ダイナミック하게 (JVM 재시작없이..) -XX와 관련한 옵션을 활성화/비활성화하는 것을 소개하고자

HeapDumpOnOutOfMemoryError 설정하는 것을 소스상에 포함해 보았다.

### ② Eclipse Memory Analyzer 소개

최근 SUN, HP, SAP JVM덤프를 쉽게 분석해 주는 Eclipse 툴이 소개되었다.

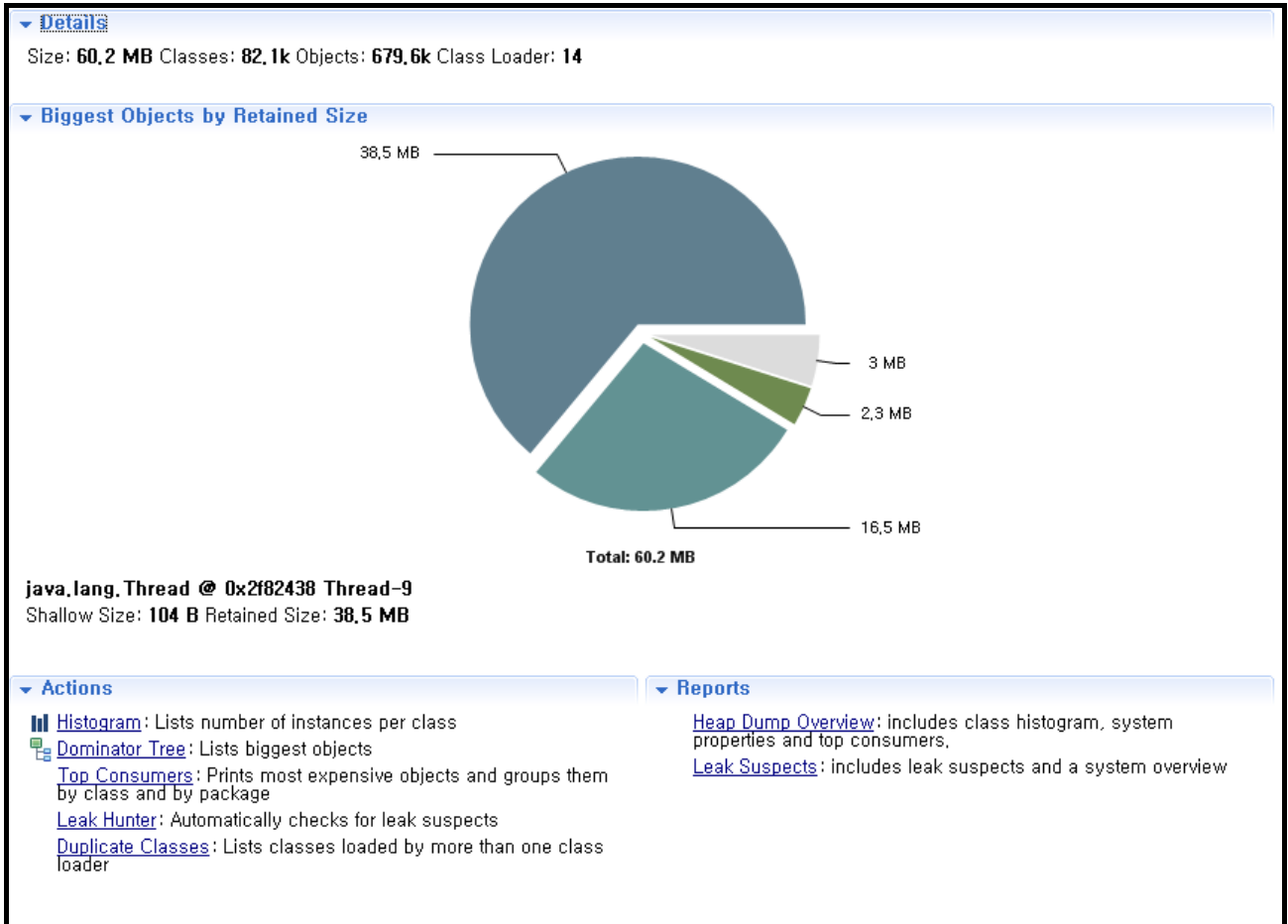
(참고 : <http://www.eclipse.org/mat/>)

이클립스 기반의 UI와 강력한 분석 기능을 가지고 있어 쉽게 그동안 IBM 힙덤프 분석툴과 같은 것을

기대했던 Sun, HP JAVA사용자에게는 기쁜 소식이 될 것 같다.

### [Mat 사용 예 및 주요기능 소개]

- ① 파일선택 → Oepn heap dump → 데모에서 생성된 힙덤프선택(java\_pid6716.hprof)
- ② 분석이 진행(덤프파일사이즈에 따라 분석시간에 차이가 날 수 있습니다.)
- ③ Overview화면

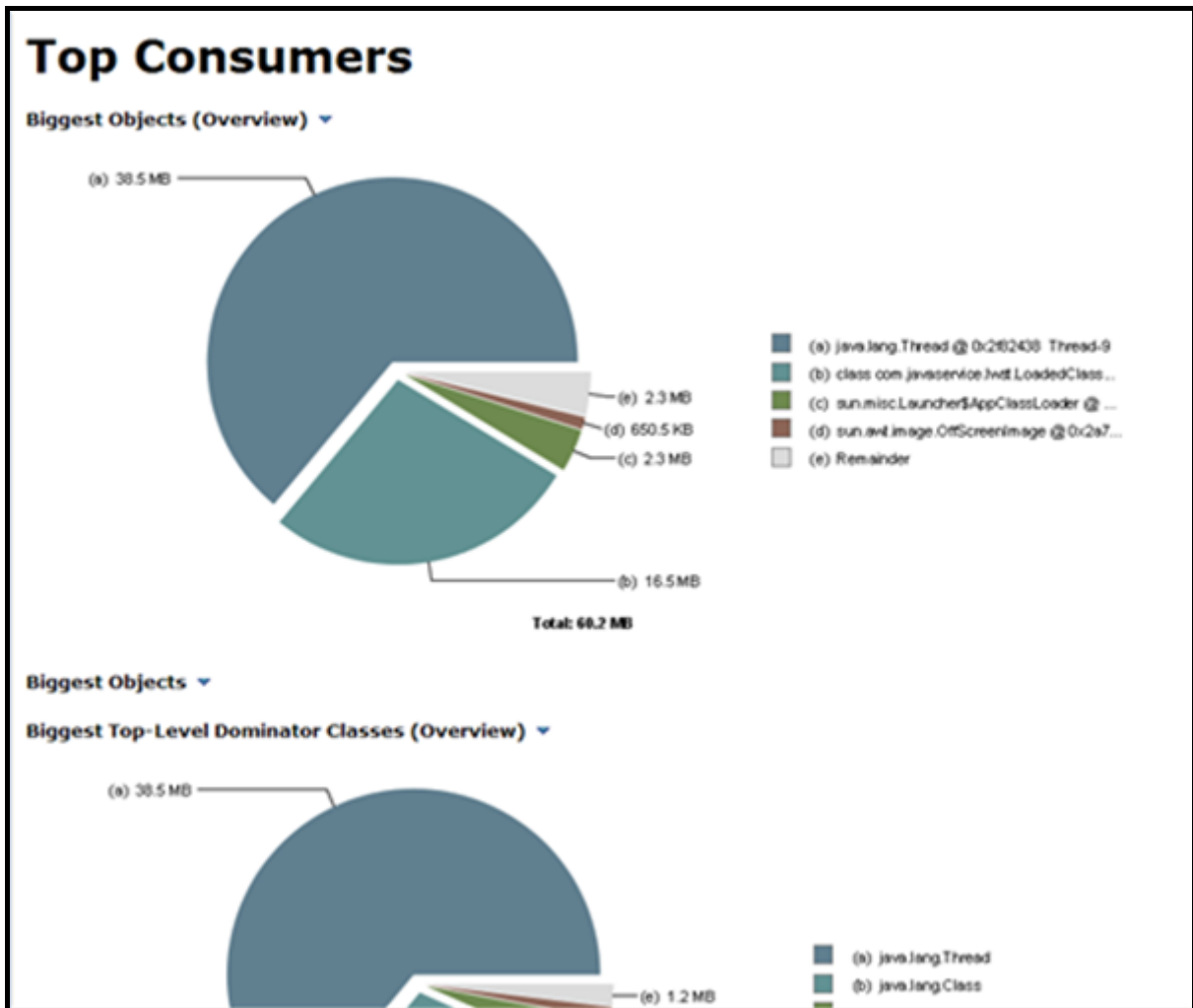


④ Histogram화면

Class Name	Objects	Shallow Heap	Retained Heap
<Regex Filter>	<Numeric Filter>	<Numeric Filter>	<Numeric Filter>
java.lang.Object[]	814	40,369,408	
char[]	166,740	6,928,872	
java.lang.String	166,740	4,001,760	
com.javaservice.lwst.LoadedClasses\$ClassState	79,868	3,194,720	
byte[]	567	2,271,752	
com.javaservice.lwst.util.SimpleSet\$Entry	79,907	1,917,768	
java.lang.String[]	80,379	1,295,688	
com.javaservice.lwst.util.SimpleSet\$Entry[]	16	843,648	
int[]	112	716,008	
float[]	16	403,456	
com.javaservice.lwst.util.SimpleStringSet\$Entry[]	2	400,080	
java.lang.Object	8,155	65,240	
java.util.HashMap\$Entry	2,061	49,464	
com.javaservice.lwst.LoadedClasses\$CapStackCtr[]	1	40,016	
com.javaservice.lwst.LiveObjectTrace\$LiveObjectState[]	1	40,016	
com.javaservice.jennifer.core.AppObj[]	1	39,904	
long[]	34	33,696	
java.util.Hashtable\$Entry	1,217	29,208	
java.util.HashMap\$Entry[]	229	29,104	
java.lang.reflect.Method	325	28,600	
com.javaservice.jennifer.core.b\$Thinktime[]	1	28,024	
java.util.TreeMap\$Entry	835	26,720	
java.lang.Class	82,108	26,688	
short[]	301	17,736	
java.net.DatagramPacket	530	16,960	
java.util.Hashtable\$Entry[]	68	14,744	
com.javaservice.jennifer.core.TxObj[]	3	14,328	
com.javaservice.lwst.util.SimpleHashMap\$Entry[]	14	13,000	
boolean[]	26	11,856	
sun.font.TrueTypeFont\$DirectoryEntry	446	10,704	
java.util.HashMap	224	8,960	
com.javaservice.jennifer.agent.c[]	1	8,064	
com.javaservice.jennifer.util.StringWeekHashtable\$Entry[]	1	8,024	
sun.java2d.loops.Blit	199	7,960	
sun.java2d.loops.ScaledBlit	163	6,520	
java.lang.Short	405	6,480	
java.lang.Integer	393	6,288	
java.util.WeakHashMap\$Entry	139	5,560	
<b>Total: 38 of 82,099 entries displayed</b>	<b>679,643</b>	<b>63,159,136</b>	



⑥ Top Consumer



⑦ Leak Hunter

Home Table Of Contents Up

**Description** ▾

The thread **java.lang.Thread @ 0x2f82438 Thread-9** keeps local variables with total size **40,392,768 (63.95%)** bytes.

The memory is accumulated in one instance of "**java.lang.Object[]**" loaded by "<system class loader>".

**Keywords**  
java.lang.Object[]

**Shortest Paths To the Accumulation Point** ▾

Class Name	Shallow Heap	Retained Heap
java.lang.Object[100000001] @ 0x2f90f40	40,000,016	40,063,904
jennifersoft.tech.support.MemoryMonitor\$Memeater @ 0x2f824a8	64	40,392,200
<Java Local>, target java.lang.Thread @ 0x2f82438 Thread-9 Thread	104	40,392,768
classloader java.security.ProtectionDomain @ 0x55b6950 *	32	80
Σ Total: 2 entries		

**Accumulated Objects** ▾

Class name	Shallow Heap	Retained Heap	Percentage
java.lang.Thread @ 0x2f82438 Thread-9	104	40,392,768	63.95%
jennifersoft.tech.support.MemoryMonitor\$Memeater @ 0x2f824a8	64	40,392,200	63.95%
java.lang.Object[100000001] @ 0x2f90f40	40,000,016	40,063,904	63.43%
java.lang.Object @ 0x2a730d0	8	8	0.00%
java.lang.Object @ 0x2a738f8	8	8	0.00%
java.lang.Object @ 0x2a74120	8	8	0.00%
java.lang.Object @ 0x2a74948	8	8	0.00%

5. 맺음말

Extra Agent 기능은 제니퍼에이전트 모니터링 기능을 손쉽게 확대할 수 있는 인터페이스를 제공합니다. Extra Agent기능 통해 최근 그 기능이 확대되고 있는 JMX기능을 손쉽게 제니퍼와 결합함으로써 사용자 시스템 환경에 적합한 추가적인 모니터링 데이터 수집 및 경고시스템 구축을 좀더 용이하게 할 수 있습니다.

## ■ 참고자료

- ManagementFactory

<http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/ManagementFactory.html>

- Memory notifications in Java

<http://recursor.blogspot.com/2006/10/memory-notifications-in-java.html>

- Detecting Memory Problems Before they Hurt

[http://www.theserverside.com/news/thread.tss?thread\\_id=27481](http://www.theserverside.com/news/thread.tss?thread_id=27481)

- Java SE 6 Monitoring, Management, Diagnosability

[http://weblogs.java.net/blog/mandychung/archive/2006/12/java\\_se\\_6\\_monit\\_1.html](http://weblogs.java.net/blog/mandychung/archive/2006/12/java_se_6_monit_1.html)

- Memory Analyzer (MAT)

<http://www.eclipse.org/mat/>

## ■ 문서에서 사용된 예제파일

heapalert.jar: Extra Agent인터페이스 구현 예제(소스파일 포함)