Java GC系列（4）：垃圾回收監視和分析

importnew.com/13838.html

本文由 ImportNew - lomoxy 翻譯自 javapapers。歡迎加入翻譯小組。轉載請見文末要求。

目錄

在這個Java GC系列教程中，讓我們學習用於垃圾回收監視和分析的工具。然後，選用一種工具來監視一個Java示例程序的垃圾回收過程。如果你是一名初學者，你最好仔細閱讀該系列教程。你可以從這裡（垃圾回收介紹）開始。

Java GC監視和分析工具

下面是一些可用的工具，每個都有自己的優勢和缺點。我們可以通過選擇正確的工具並分析，來提升應用程序的性能。這篇教程中，我們選用Java VisualVM。

- Java VisualVM
- Naarad
- GCViewer
- IBM Pattern Modeling and Analysis Tool for Java Garbage Collector
- HPjmeter
- IBM Monitoring and Diagnostic Tools for Java-Garbage Collection and Memory
- Visualizer
- Verbose GC Analyzer

Java VisualVM

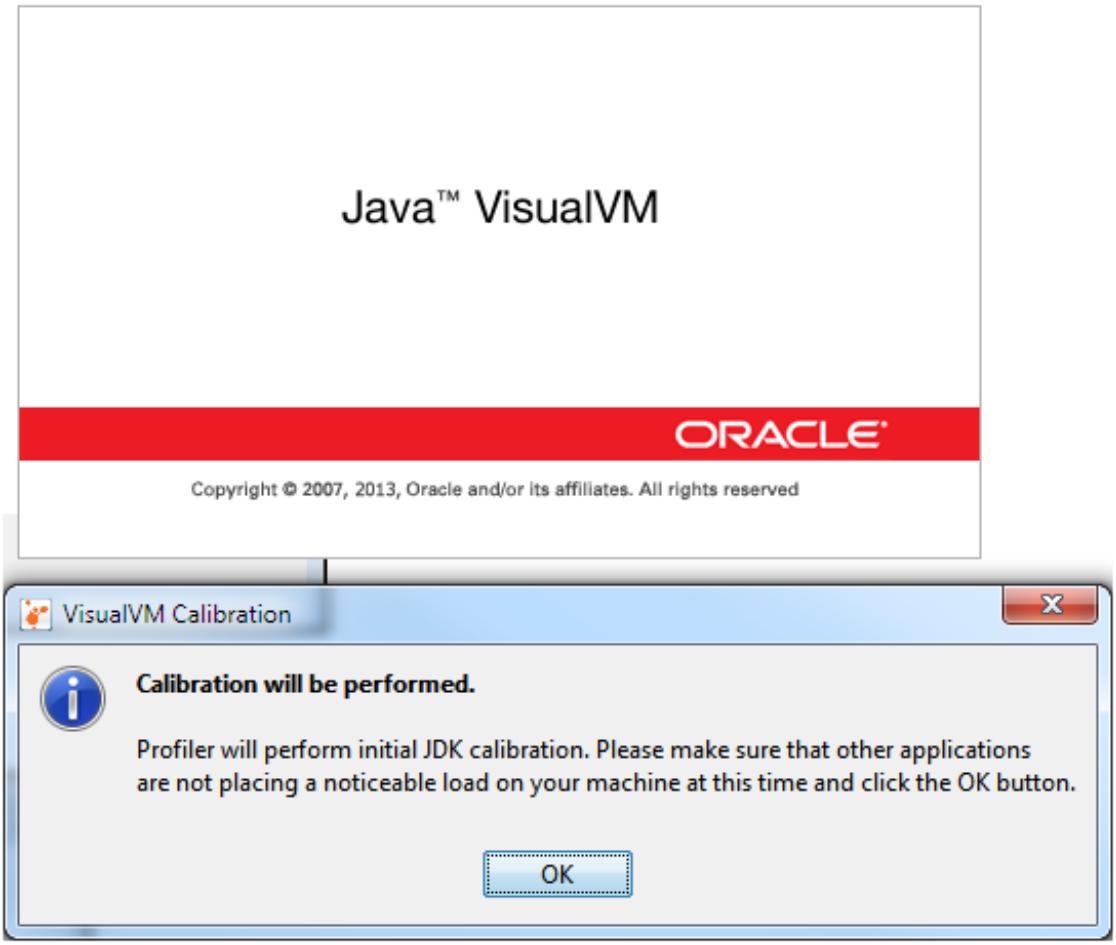
Java VisualVM使用是免費的，其需要安裝Java SE SDK。看一下Java JDK的bin文件夾中（路徑：\Java\jdk1.8.0\bin），這裡面有很多javac和java工具，jvisualvm就是其中之一。

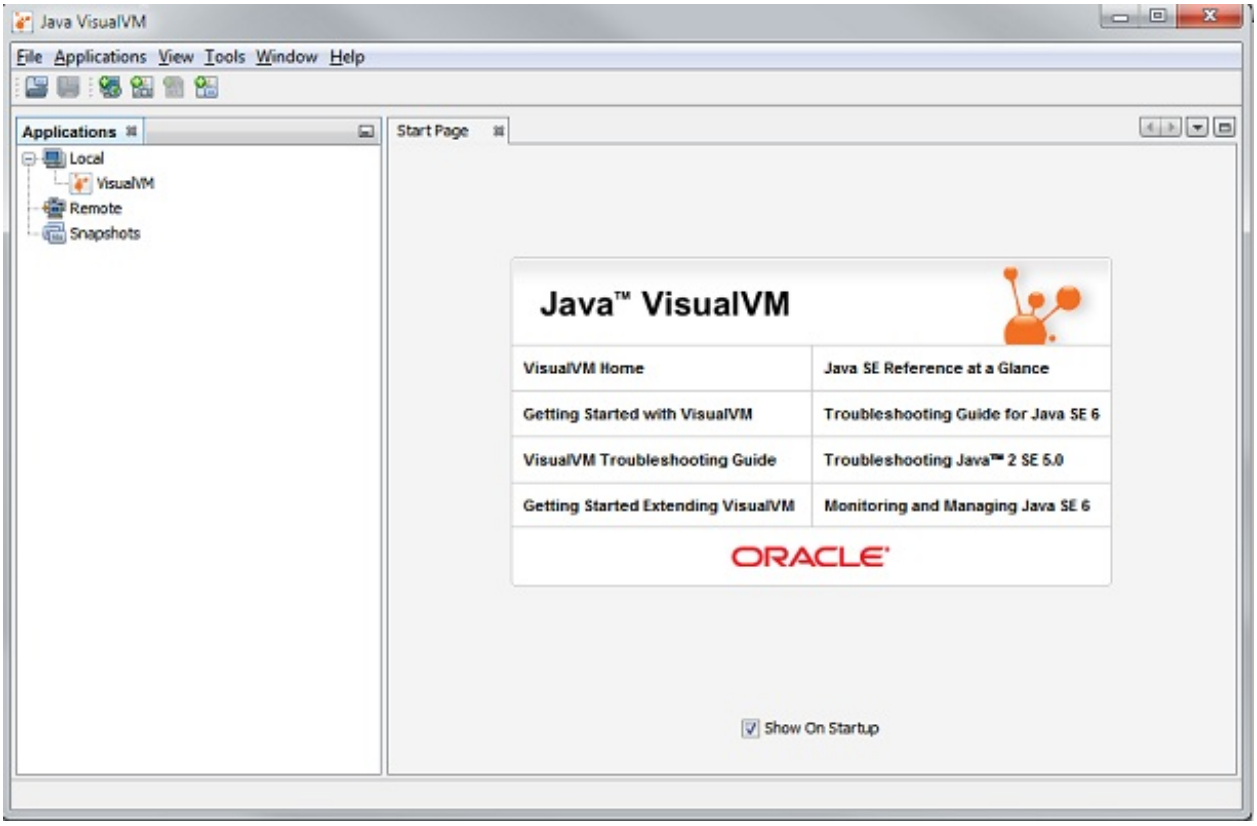
Java VisualVM能夠被用於：

- 生成並分析堆的內存轉儲；
- 在MBeans上觀察並操作；
- 監視垃圾回收；
- 內存和CPU性能分析；

1、啟動VisualVM

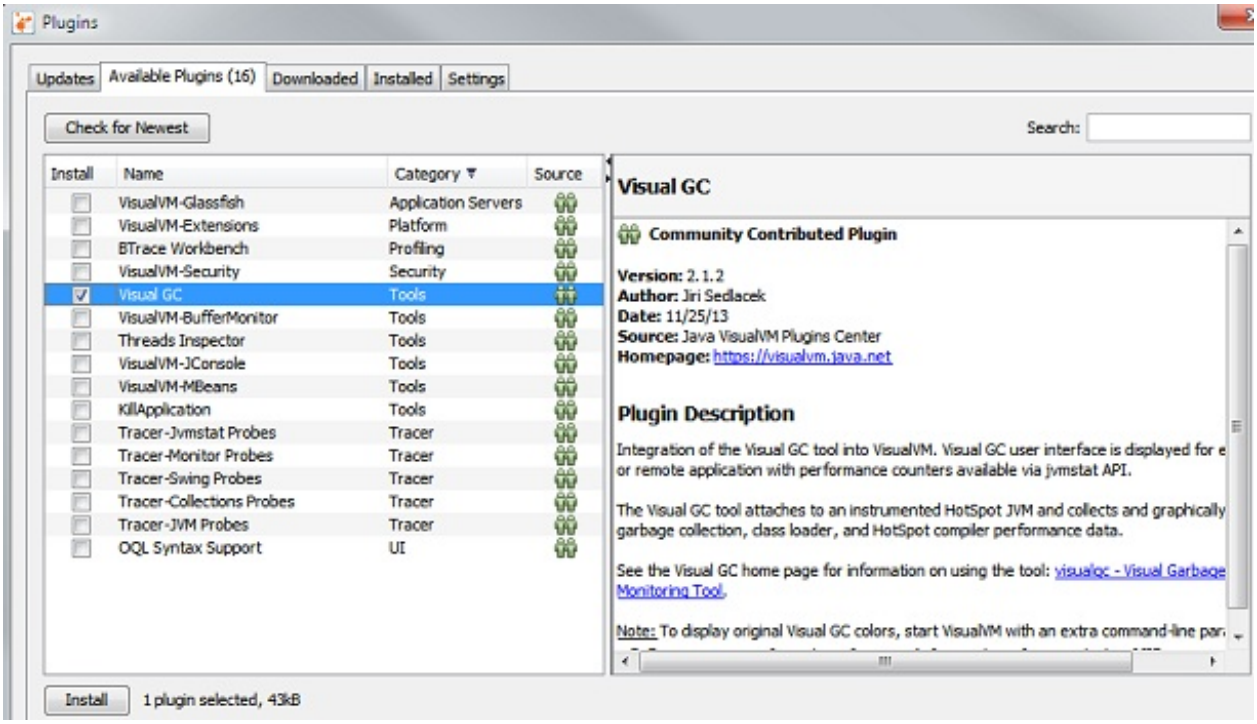
jvisualvm位於JDK bin文件夾下，直接點擊就可以。


Java™ VisualVM
ORACLE
Copyright © 2007, 2013, Oracle and/or its affiliates. All rights reserved
VisualVM Calibration
Calibration will be performed.
Profiler will perform initial JDK calibration. Please make sure that other applications are not placing a noticeable load on your machine at this time and click the OK button.
OK


Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
Remote
Snapshots
Start Page
Java™ VisualVM
VisualVM Home
Java SE Reference at a Glance
Getting Started with VisualVM
Troubleshooting Guide for Java SE 6
VisualVM Troubleshooting Guide
Troubleshooting Java™ 2 SE 5.0
Getting Started Extending VisualVM
Monitoring and Managing Java SE 6
ORACLE
Show On Startup

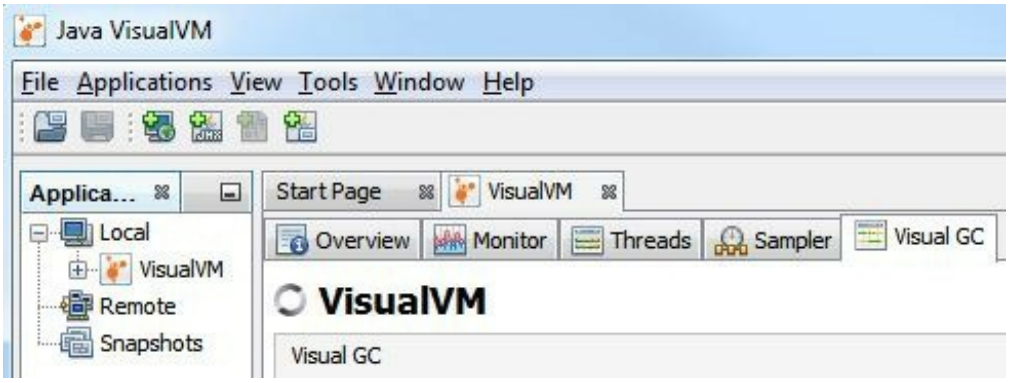
2、安装可视化GC插件

我們需要安裝可視化GC插件，以便在Java GC過程中有良好的視覺感受。



3、監視GC

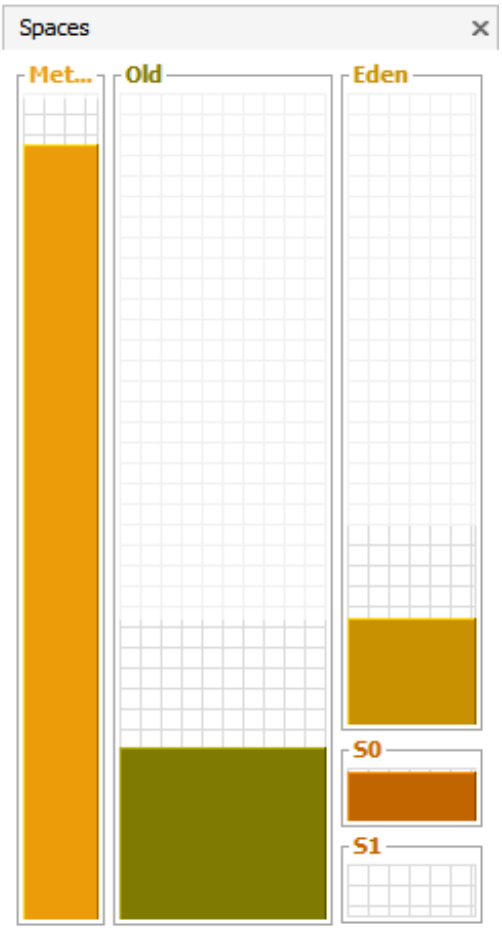
現在，是時候監視垃圾回收進程了，開啟你的Java程序，它將自動被檢測到並顯示到Java VisualVM界面，左側「Application」（應用程序）窗口下，「Local」（本地節點）下，所有本地運行的Java程序都會被列出。



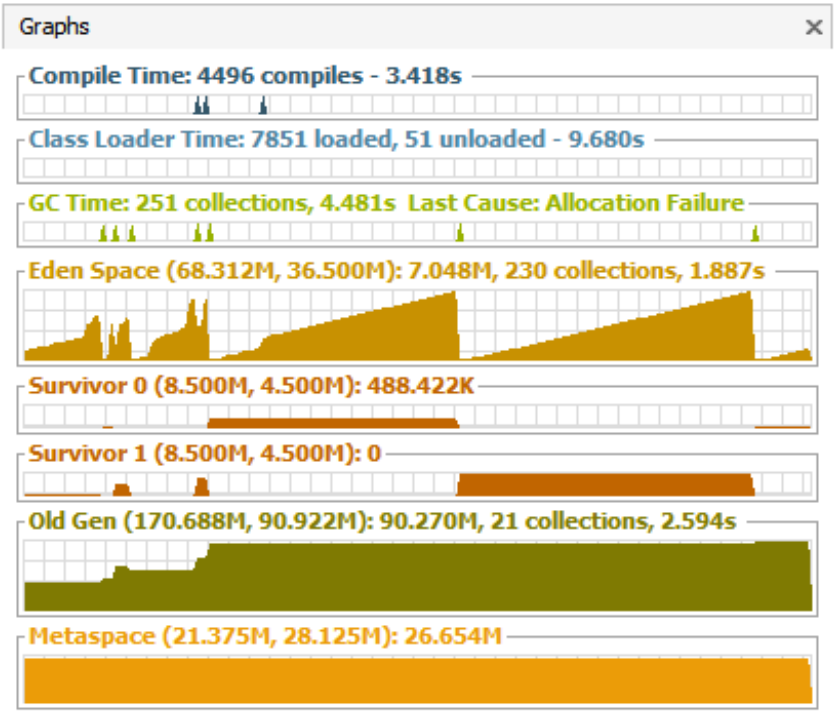
Java VisualVM是一個Java應用程序，因此它也會被列在其中，教程的意圖在於使用VisualVM來監視它自己的GC進程。

雙擊「Local」（本地）下的VisualVM圖標。

現在，程序監控窗口在右側打開，這有許多不同關於應用程序性能的相關監視指數的tab頁，目前為止，我們最感興趣的是「Visual GC」，點擊它。



上面圖片顯示在Old、Eden、S0和S1上空間利用情況，下圖顯示了每部分空間的分配和釋放情況。它按照指定的刷新率保持持續刷新。



上面圖片所展示的是正常運行程序的情況，當出現內存洩露或者反常的行為時，它會在圖表中明確的顯示出來。最少我們能理解他是與對象內存分配和垃圾回收相關的事情。隨後，通過其他tab頁（像「Threads」）和Thread Dump的幫助，我們能夠減少這個問題。

在「Monitor」tab頁中，我們能夠監控並定時展示所有堆內存使用情況圖。通過「Perform GC」按鈕可以啟動垃圾回收進程。

在「Sampler」tab頁中，我們能夠啟動內存和CPU性能分析，它將顯示詳細每個實例使用的實時報告，它將幫助我們明確性能問題。

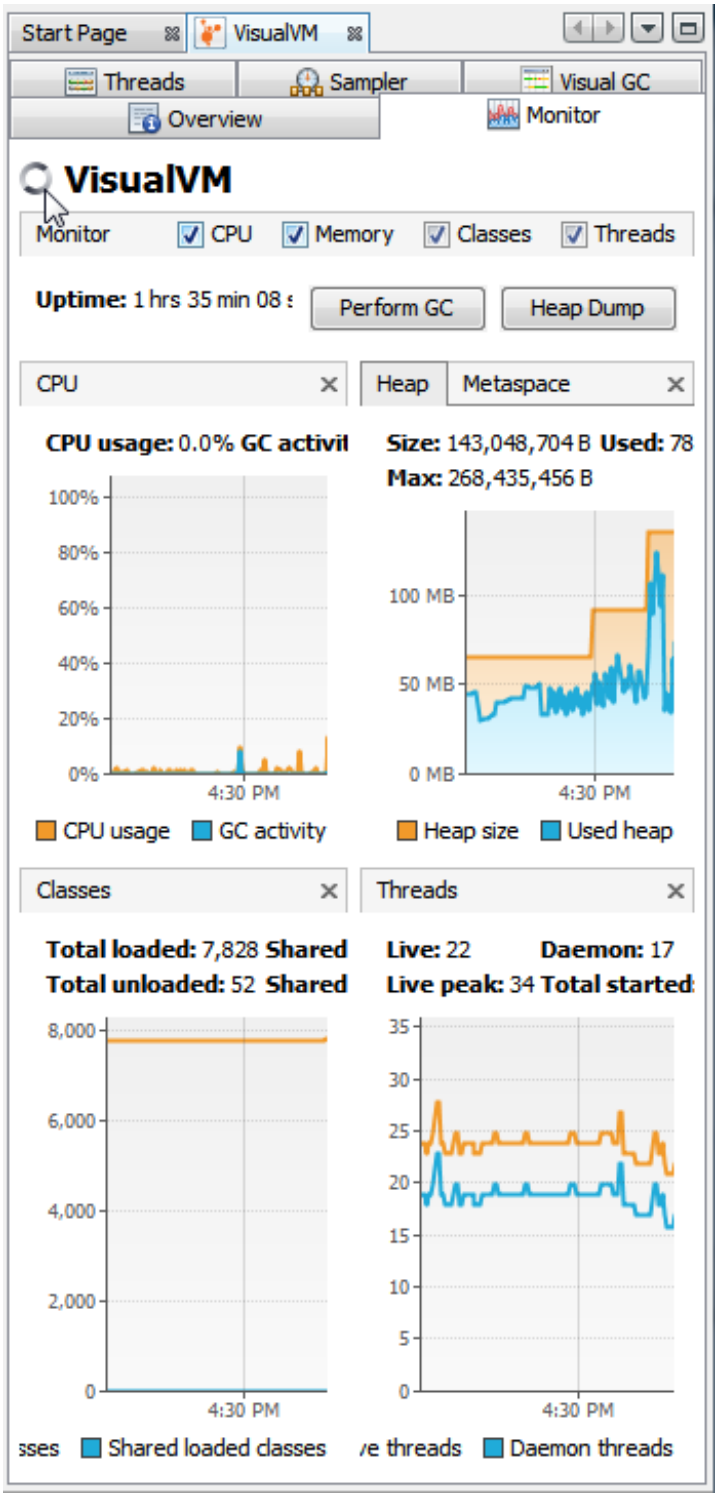
這篇教程是我們四篇Java垃圾回收系列教程的最後一篇。

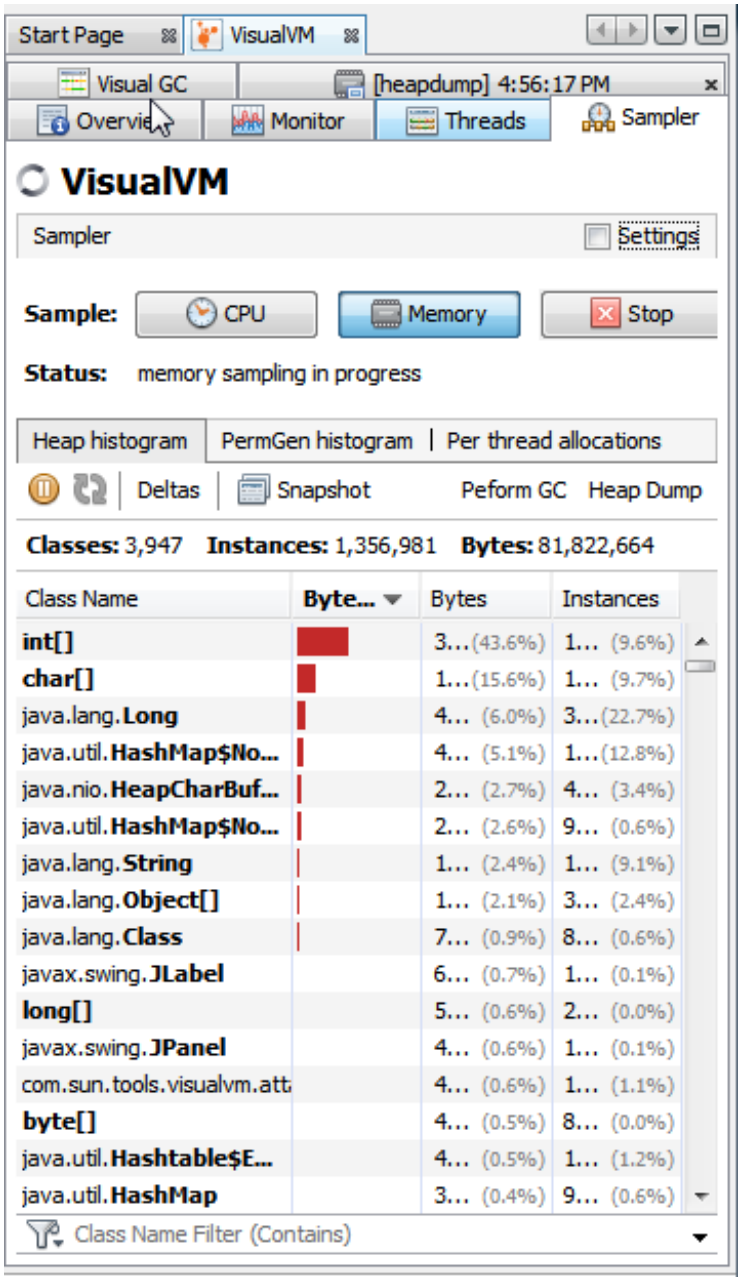
原文鏈接： javapapers 翻譯： ImportNew.com - lomoxy
譯文鏈接： http://www.importnew.com/13838.html
[轉載請保留原文出處、譯者和譯文鏈接。]

關於作者： lomoxy

（新浪微博：**@肖元fff**）

查看lomoxy的更多文章 >>




Start Page
VisualVM
Visual GC
[heapdump] 4:56:17 PM
Overvie
Monitor
Threads
Sampler
VisualVM
Sampler
Settings
Sample:
CPU
Memory
Stop
Status:
memory sampling in progress
Heap histogram
PermGen histogram
Per thread allocations
Deltas
Snapshot
Peform GC
Heap Dump
Classes: 3,947
Instances: 1,356,981
Bytes: 81,822,664
Class Name
Byte...
Bytes
Instances
int[] 3...(43.6%) 1... (9.6%)
char[] 1...(15.6%) 1... (9.7%)
java.lang.Long 4... (6.0%) 3...(22.7%)
java.util.HashMap$No... 4... (5.1%) 1...(12.8%)
java.nio.HeapCharBuf... 2... (2.7%) 4... (3.4%)
java.util.HashMap$No... 2... (2.6%) 9... (0.6%)
java.lang.String 1... (2.4%) 1... (9.1%)
java.lang.Object[] 1... (2.1%) 3... (2.4%)
java.lang.Class 7... (0.9%) 8... (0.6%)
javax.swing.JLabel 6... (0.7%) 1... (0.1%)
long[] 5... (0.6%) 2... (0.0%)
javax.swing.JPanel 4... (0.6%) 1... (0.1%)
com.sun.tools.visualvm.att 4... (0.6%) 1... (1.1%)
byte[] 4... (0.5%) 8... (0.0%)
java.util.Hashtable$E... 4... (0.5%) 1... (1.2%)
java.util.HashMap 3... (0.4%) 9... (0.6%)
Class Name Filter (Contains)