Eclipse Memory Analyzer 堆转储文件分析

目录

[Eclipse Memory Analyzer 堆转储文件分析 1](#_Toc323135522)

[1 背景知识 1](#_Toc323135523)

[1.1. Java如何管理内存 1](#_Toc323135524)

[1.2. Java Heap Dump 是什么？ 2](#_Toc323135525)

[2 mat介绍 2](#_Toc323135526)

[2.1. MAT是什么？ 2](#_Toc323135527)

[2.2 为什么使用MAT？ 3](#_Toc323135528)

[3 概述 3](#_Toc323135529)

[4 准备环境和测试数据 3](#_Toc323135530)

[5 安装 MAT 3](#_Toc323135531)

[6 配置环境参数 5](#_Toc323135532)

[7 测试类准备 6](#_Toc323135533)

[8 设置JVM 参数 6](#_Toc323135534)

[9 Windows平台使用Mat 7](#_Toc323135535)

[9.1 JAVA应用配置JVM参数，及生成hprof文件 7](#_Toc323135536)

[9.2 Tomcat应用配置JVM参数，及生成hprof文件 8](#_Toc323135537)

[9.3 Jboss应用配置JVM参数，及生成hprof文件 10](#_Toc323135538)

[10 LINUX使用MAT 12](#_Toc323135539)

[10.1 JAVA应用配置JVM参数，及生成hprof文件 12](#_Toc323135540)

[10.2 Tomcat应用配置JVM参数，及生成hprof文件 13](#_Toc323135541)

[10.3 Jboss应用配置JVM参数，及生成hprof文件 14](#_Toc323135542)

[11 使用mat进行分析hprof文件 15](#_Toc323135543)

[12 Mat工具其他功能说明 22](#_Toc323135544)

[12.1 Histogram 22](#_Toc323135545)

[12.2 Dominator Tree 23](#_Toc323135546)

[12.3 Path to GC Roots 25](#_Toc323135547)

[12.4 Dominator Tree Grouped by Class Loader 25](#_Toc323135548)

[13 其他资料 26](#_Toc323135549)

# 背景知识

## Java如何管理内存

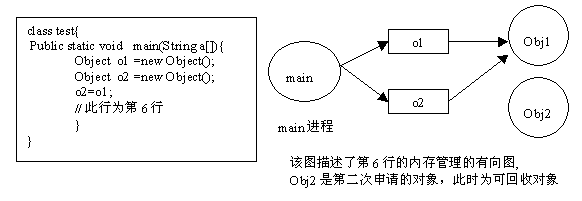
Java的内存管理就是对象的分配和释放问题。在Java中，程序员需要通过关键字new为每个对象申请内存空间 (基本类型除外)，所有的对象都在堆 (Heap)中分配空间。另外，对象的释放是由GC决定和执行的。在Java中，内存的分配是由程序完成的，而内存的释放是有GC完成的，这种收支两条线 的方法确实简化了程序员的工作。但同时，它也加重了JVM的工作。这也是Java程序运行速度较慢的原因之一。因为，GC为了能够正确释放对象，GC必须监控每一个对象的运行状态，包括对象的申请、引用、被引用、赋值等，GC都需要进行监控。

监视对象状态是为了更加准确地、及时地释放对象，而释放对象的根本原则就是该对象不再被引用。

为了更好理解GC的工作原理，我们可以将对象考虑为有向图的顶点，将引用关系考虑为图的有向边，有向边从引用者指向被引对象。另外，每个线程对象可以作为一个图的起始顶点，例如大多程序从main进程开始执行，那么该图就是以main进程顶点开始的一棵根树。在这个有向 图中，根顶点可达的对象都是有效对象，GC将不回收这些对象。如果某个对象 (连通子图)与这个根顶点不可达(注意，该图为有向图)，那么我们认为这个(这些)对象不再被引用，可以被GC回收。

以下，我们举一个例子说明内存管理。对于程序的每一个时刻，我们都有一个有向图表示JVM的内存分配情况

以下右图，就是左边程序运行到第6行的示意图。



Java 使用有向图的方式进行内存管理，可以消除引用循环的问题，例如有三个对象，相互引用，只要它们和根进程不可达的，那么GC也是可以回收它们的。这种方式的 优点是管理内存的精度很高，但是效率较低。另外一种常用的内存管理技术是使用计数器，例如COM模型采用计数器方式管理构件，它与有向图相比，精度行低 (很难处理循环引用的问题)，但执行效率很高。

## Java Heap Dump 是什么？

我们知道Java Heap 是所有类实例和数组对象分配的一个运行时数据区，其间所有Java VM线程在执行期间共享Heap 中的数据。那么一个Java heap dump相当于在一个特殊的时间点上生成的一个快照，它就像给一个繁忙的数据仓库在给定的时间上来了一个照片，我们通过这张快照可以识别哪些组件在那快照的那时间点上是可用的。

由于Java 说明文档并没有提及到Java heap dump，在各个不同的JVM厂商，存在各个对Java heap dump的介绍。 如IBM JVM的Java heap dump 提供的信息大至和Java Heap差不多。

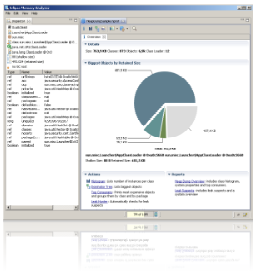
Sun公司提供的信息基本上是JVM Stack，运行时常量池和Java Heap等.

# mat介绍

## 2.1. MAT是什么？

MAT(Memory Analyzer Tool)，首页：<http://www.eclipse.org/mat/>。一个基于Eclipse的内存分析工具，是一个快速、功能丰富的JAVA heap分析工具，它可以帮助我们查找内存泄漏和减少内存消耗。使用内存分析工具从众多的对象中进行分析，快速的计算出在内存中对象的占用大小，看看是谁阻止了垃圾收集器的回收工作，并可以通过报表直观的查看到可能造成这种结果的对象。

 Eclipse Memory Analyzer（MAT）是著名的跨平台集成开发环境 Eclipse Galileo 版本的 33 个组成项目中之一，它是一个功能丰富的 JAVA 堆转储文件分析工具，可以帮助你发现内存漏洞和减少内存消耗。本文主要介绍如何安装配置 Memory Analyzer，并结合一个实例，介绍如何利用 MAT 来进行堆转储文件分析，找到内存泄露的根源。



## 2.2 为什么使用MAT？

当服务器应用占用了过多内存的时候，会遇到OutOfMemoryError。如何快速定位问题呢？Eclipse MAT的出现使这个问题变得非常简单。它能够离线分析dump的文件数据。  
    Eclipse MAT是SAP公司贡献的一个工具，可以在Eclipse网站下载到它，完全免费的。它可比Sun提供的内存镜像分析工具jhat要强太多了。

# 概述

对于大型 JAVA 应用程序来说，再精细的测试也难以堵住所有的漏洞，即便我们在测试阶段进行了大量卓有成效的工作，很多问题还是会在生产环境下暴露出来，并且很难在测试环境中进行重现。JVM 能够记录下问题发生时系统的部分运行状态，并将其存储在堆转储 (Heap Dump) 文件中，从而为我们分析和诊断问题提供了重要的依据。

通常内存泄露分析被认为是一件很有难度的工作，一般由团队中的资深人士进行。不过，今天我们要介绍的 MAT（Eclipse Memory Analyzer）被认为是一个“傻瓜式“的堆转储文件分析工具，你只需要轻轻点击一下鼠标就可以生成一个专业的分析报告。和其他内存泄露分析工具相 比，MAT 的使用非常容易，基本可以实现一键到位，即使是新手也能够很快上手使用。

MAT 的使用是如此容易，你是不是也很有兴趣来亲自感受下呢，那么第一步我们先来安装 MAT。

# 准备环境和测试数据

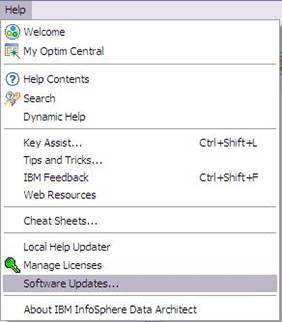
我们使用的是 Eclipse Memory Analyzer V1.1，Sun JDK 6

# 安装 MAT

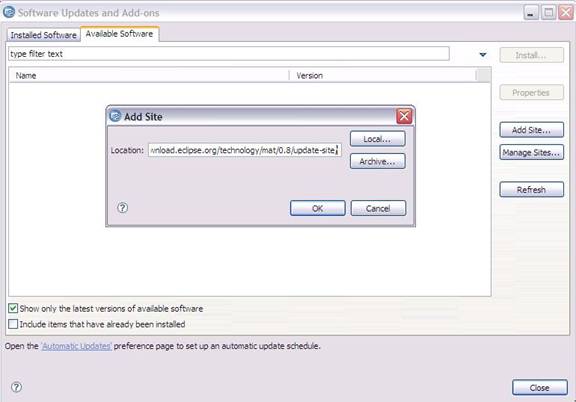
和其他插件的安装非常类似，MAT 支持两种安装方式，一种是“单机版“的，也就是说用户不必安装 Eclipse IDE 环境，MAT 作为一个独立的 Eclipse RCP 应用运行；另一种是”集成版“的，也就是说 MAT 也可以作为 Eclipse IDE 的一部分，和现有的开发平台集成。

集成版的安装需要借助 Update Manager。

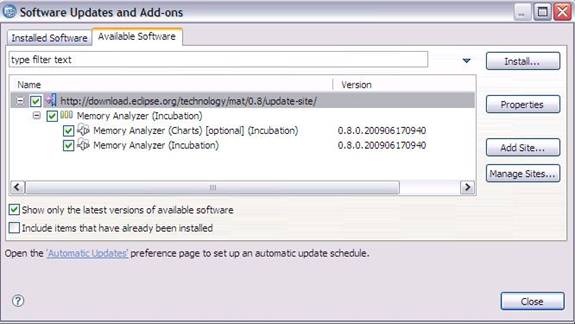
如图 1 所示，首先通过 Help -> Software Updates... 启动软件更新管理向导。

**图 1. 安装插件第一步**  


选择“Available Software“然后按如图 2 所示的方式添加 MAT 的更新地址 http://download.eclipse.org/mat/1.1/update-site/。

**图 2. 安装插件第二步**  


如图 3 所示，接下来选择你想要安装的 MAT 的功能点，需要注意的是 Memory Analyzer (Chart) 这个功能是一个可选的安装项目，它主要用来生成相关的报表，不过如果需要用到这个功能，你还需要额外的安装 BIRT Chart Engine。

**图 3. 安装插件第三步**  


插件安装完毕，你还需要重新启动 Eclipse 的工作平台。

比较而言，单机版的安装方式非常简单，用户只需要下载相应的安装包，然后解压缩即可运行，这也是被普遍采用的一种安装方式。在下面的例子里，我们使用的也是单机版的 MAT。具体的下载要求和地址可参见其产品下载页面：<http://www.eclipse.org/mat/downloads.php>。

另外，如果你需要用 MAT 来分析 IBM JVM 生成的 dump 文件的话，还需要额外安装 IBM Diagnostic Tool Framework ，具体的下载和安装配置步骤请参见：http://www.ibm.com/developerworks/java/jdk/tools /dtfj.html

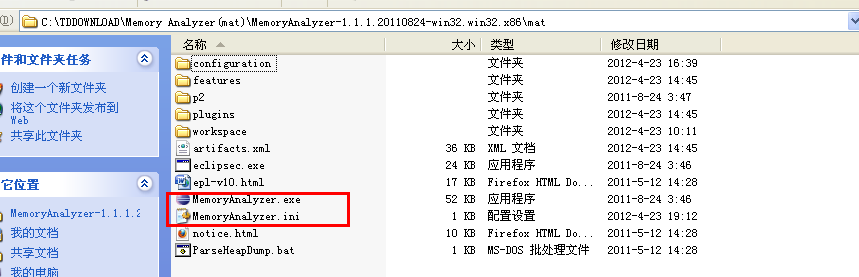
# 配置环境参数

安装完成之后，为了更有效率的使用 MAT，我们还需要做一些配置工作。因为通常而言，分析一个堆转储文件需要消耗很多的堆空间，为了保证分析的效率和性能，在有条件的情况下，我们会建议分 配给 MAT 尽可能多的内存资源。你可以采用如下两种方式来分配内存更多的内存资源给 MAT。

一种是修改启动参数 MemoryAnalyzer.exe -vmargs -Xmx4g

另一种是编辑文件 MemoryAnalyzer.ini，在里面添加类似信息 -vmargs – Xmx4g。

至此，MAT 就已经成功地安装配置好了，开始进入实战吧。



# 测试类准备





# 设置JVM 参数

首先需要设置了如下所示的 JVM 参数：

-XX:+HeapDumpOnOutOfMemoryError

JVM 就会在发生内存泄露时抓拍下当时的内存状态，也就是我们想要的堆转储文件。

如果你不想等到发生崩溃性的错误时才获得堆转储文件，也可以通过设置如下 JVM 参数来按需获取堆转储文件。

-XX:+HeapDumpOnCtrlBreak

除此之外，还有很多的工具，例如 JMap，JConsole 都可以帮助我们得到一个堆转储文件。本文实例就是使用 JMap 直接获取了 Eclipse Galileo 进程的堆转储文件。您可以使用如下命令：

JMap -dump:format=b,file=<dumpfile> <pid>

jmap -dump:format=b,file=HeapDump.hprof /path/to/bin/java core\_dump\_file

Via Tools:

\* Sun (Linux, Solaris; not on Windows) JMap Java 5: jmap -heap:format=b <pid>

\* Sun (Linux, Solaris; Windows see link) JMap Java 6: jmap.exe -dump:format=b,file=HeapDump.hprof <pid>

\* Sun (Linus, Solaris) JMap with Core Dump File: jmap -dump:format=b,file=HeapDump.hprof /path/to/bin/java core\_dump\_file

\* Sun JConsole: Launch jconsole.exe and invoke operation dumpHeap() on HotSpotDiagnostic MBean

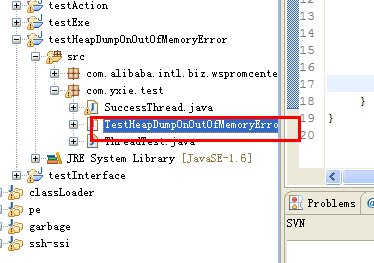
\* SAP JVMMon: Launch jvmmon.exe and call menu for dumping the heap

不过，您需要了解到，不同厂家的 JVM 所生成的堆转储文件在数据存储格式以及数据存储内容上有很多区别， MAT 不是一个万能工具，它并不能处理所有类型的堆存储文件。但是比较主流的厂家和格式，例如 Sun, HP, SAP 所采用的 HPROF 二进制堆存储文件，以及 IBM 的 PHD 堆存储文件等都能被很好的解析（您需要安装额外的插件，请参考 相关说明，本文不作详细解释）。

万事俱备，接下来，我们就可以开始体验一键式的堆存储分析功能了。

# Windows平台使用Mat

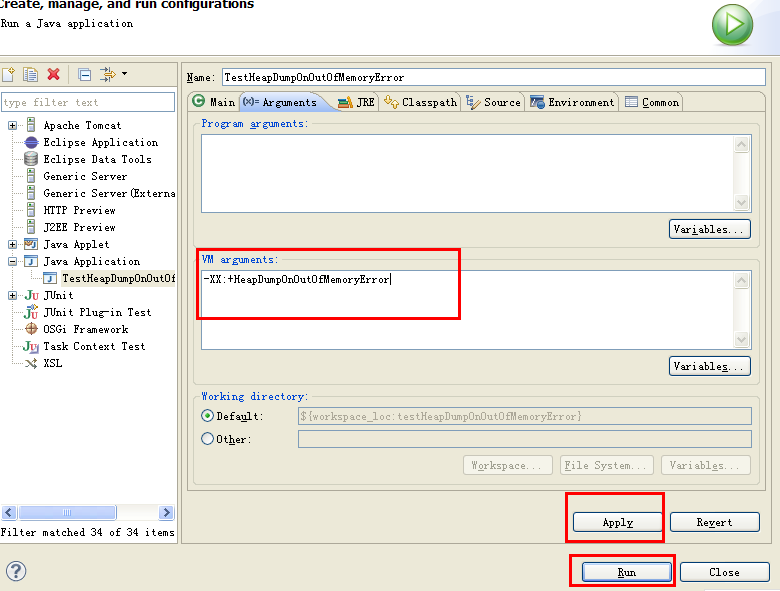
## JAVA应用配置JVM参数，及生成hprof文件

右键，Run as 🡺 Run Conigurations..中的Arguments设置VM argument为：

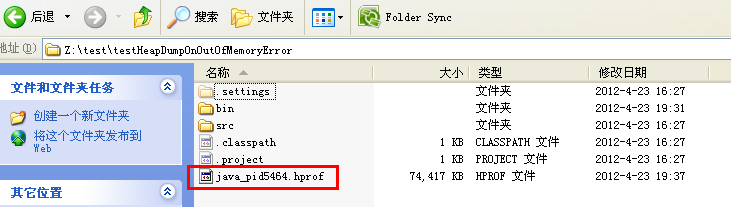
-XX:+HeapDumpOnOutOfMemoryError

另外一种指定生成文件存储位置:

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/xieyun/test/TestHeapDumpOnOutOfMemoryError.hprof



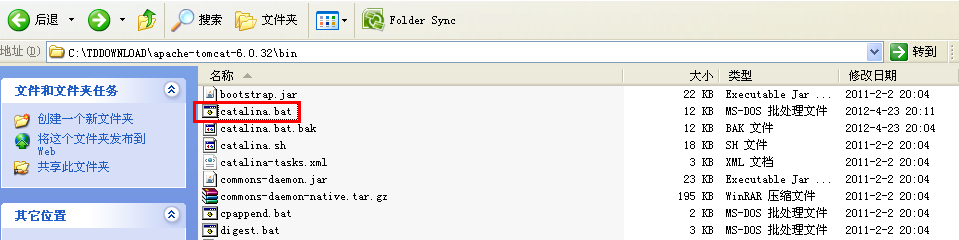
生成的hprof文件，就在工程目录下。



具体分析请参见后面统一说明。

## Tomcat应用配置JVM参数，及生成hprof文件

首先需要添加JVM参数。修改C:\TDDOWNLOAD\apache-tomcat-6.0.32\bin\catalina.bat



打开Tomcat根目录下的bin文件夹，编辑catalina.bat，在set JAVA\_OPTS=%JAVA\_OPTS%....这句之后加上

if not "%LOGGING\_CONFIG%" == "" goto noJuliConfig

set LOGGING\_CONFIG=-Dnop

if not exist "%CATALINA\_BASE%\conf\logging.properties" goto noJuliConfig

set LOGGING\_CONFIG=-Djava.util.logging.config.file="%CATALINA\_BASE%\conf\logging.properties"

:noJuliConfig

set JAVA\_OPTS=%JAVA\_OPTS% %LOGGING\_CONFIG% -XX:+HeapDumpOnOutOfMemoryError

if not "%LOGGING\_MANAGER%" == "" goto noJuliManager

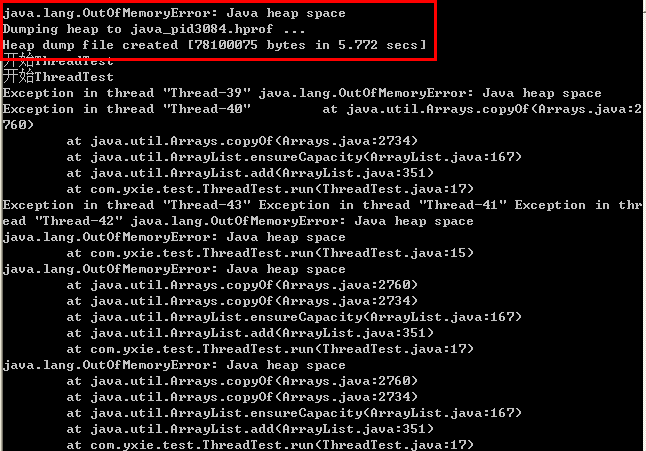
set LOGGING\_MANAGER=-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager

:noJuliManager

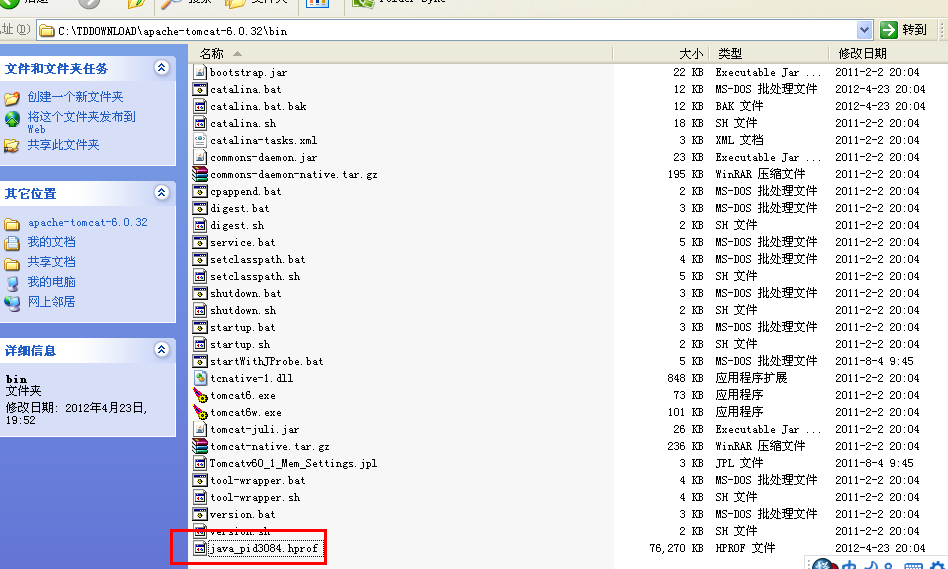
set JAVA\_OPTS=%JAVA\_OPTS% %LOGGING\_MANAGER% -XX:+HeapDumpOnOutOfMemoryError

启动tomcat:

<http://localhost:8080/testHeapDumpOnOutOfMemoryErrorWeb/HeapDumpOnOutOfMemoryErrorServlet>



生成的hprof文件在:



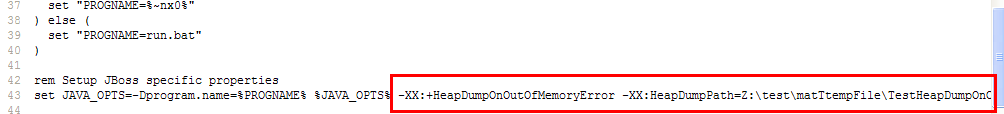
具体分析请参见后面统一说明。

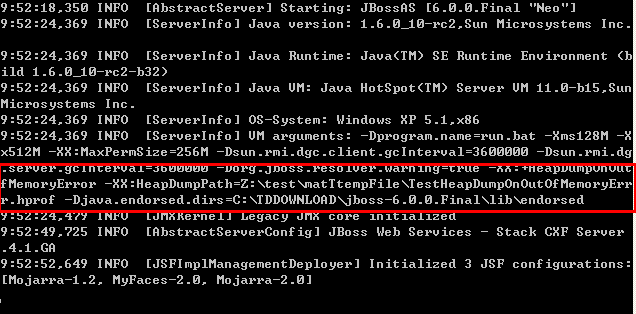
## Jboss应用配置JVM参数，及生成hprof文件

Jboss需要jdk1.6以上才能支持HeapDumpOnOutOfMemoryError配置。

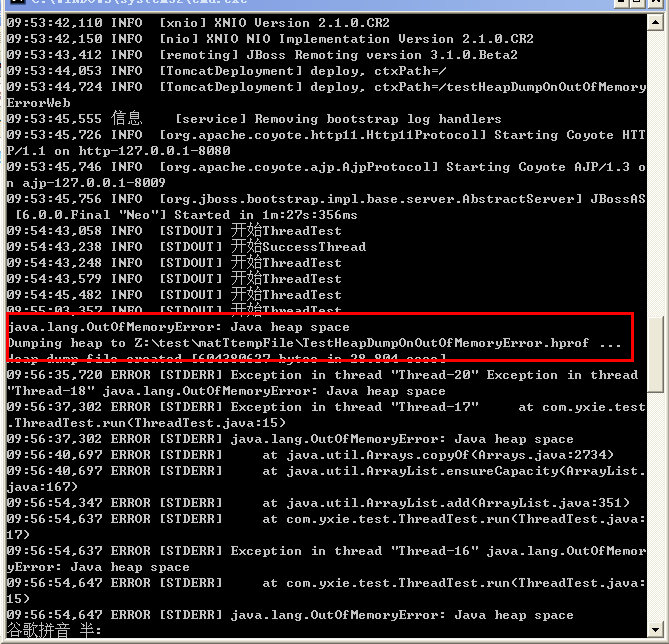
需要修改C:\TDDOWNLOAD\jboss-6.0.0.Final\bin\run.bat

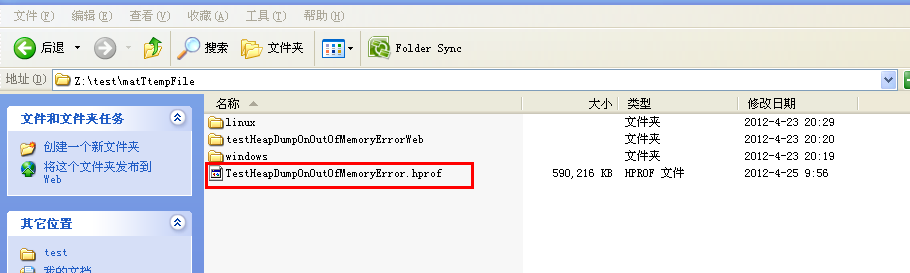
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=Z:\test\matTtempFile\TestHeapDumpOnOutOfMemoryError.hprof





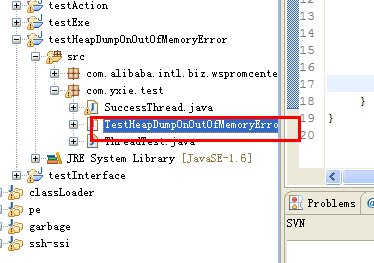
发生outOfMemery内存溢出，可以看到我们生成了对应hprof文件。





# LINUX使用MAT

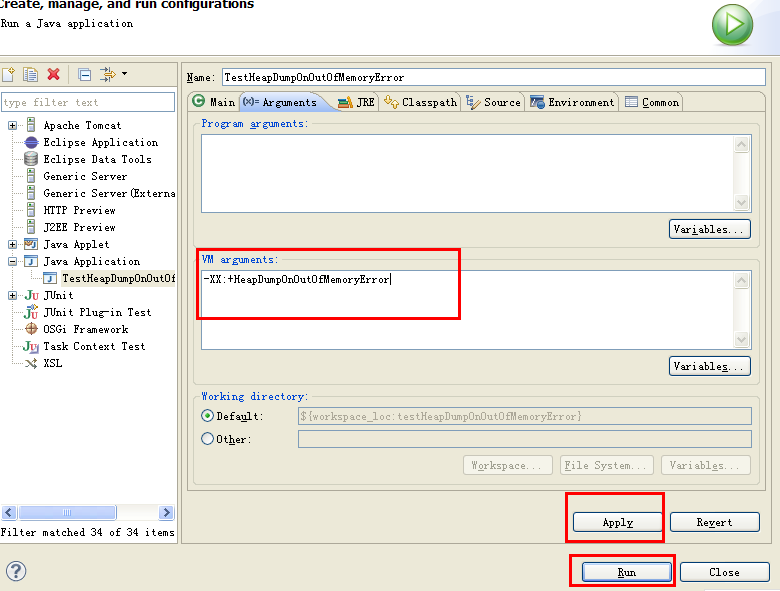
## JAVA应用配置JVM参数，及生成hprof文件

右键，Run as 🡺 Run Conigurations..中的Arguments设置VM argument为：

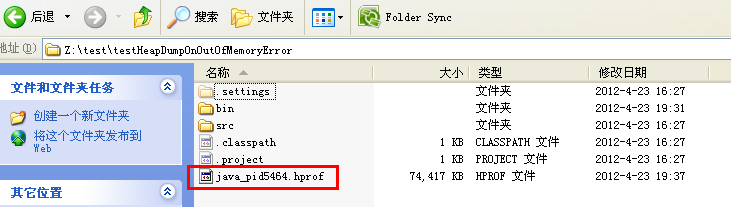
-XX:+HeapDumpOnOutOfMemoryError

另外一种指定生成文件存储位置:

-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/xieyun/test/TestHeapDumpOnOutOfMemoryError.hprof



生成的hprof文件，就在工程目录下。



具体分析请参见后面统一说明。

## Tomcat应用配置JVM参数，及生成hprof文件

首先需要添加JVM参数。修改/home/xieyun/software/apache-tomcat-6.0.32/bin/catalina.sh

if [ -z "$LOGGING\_MANAGER" ]; then

JAVA\_OPTS="$JAVA\_OPTS -XX:+HeapDumpOnOutOfMemoryError -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"

else

JAVA\_OPTS="$JAVA\_OPTS -XX:+HeapDumpOnOutOfMemoryError $LOGGING\_MANAGER"

Fi

-XX:+HeapDumpOnOutOfMemoryError一定要放在最后，不然不会生效。

启动tomcat:

xieyun@xieyun-OptiPlex-760:~/software/apache-tomcat-6.0.32/bin$ ./startup.sh

Using CATALINA\_BASE: /home/xieyun/software/apache-tomcat-6.0.32

Using CATALINA\_HOME: /home/xieyun/software/apache-tomcat-6.0.32

Using CATALINA\_TMPDIR: /home/xieyun/software/apache-tomcat-6.0.32/temp

Using JRE\_HOME: /usr/ali/java

Using CLASSPATH: /home/xieyun/software/apache-tomcat-6.0.32/bin/bootstrap.jar

启动tomcat:

<http://localhost:8080/testHeapDumpOnOutOfMemoryErrorWeb/HeapDumpOnOutOfMemoryErrorServlet>

catalina.out输出:

2012-4-23 20:41:16 org.apache.catalina.startup.Catalina start

信息: Server startup in 833 ms

开始ThreadTest

开始ThreadTest

开始ThreadTest

开始ThreadTest

开始ThreadTest

开始SuccessThread

java.lang.OutOfMemoryError: Java heap space

Dumping heap to java\_pid31185.hprof ...

Heap dump file created [718127281 bytes in 27.183 secs]

Exception in thread "Thread-13" java.lang.OutOfMemoryError: Java heap space

at com.yxie.test.ThreadTest.run(ThreadTest.java:15)

Exception in thread "Thread-10" java.lang.OutOfMemoryError: Java heap space

at com.yxie.test.ThreadTest.run(ThreadTest.java:15)

Exception in thread "Thread-9" java.lang.OutOfMemoryError: Java heap space

at java.util.Arrays.copyOf(Arrays.java:2734)

at java.util.ArrayList.ensureCapacity(ArrayList.java:167)

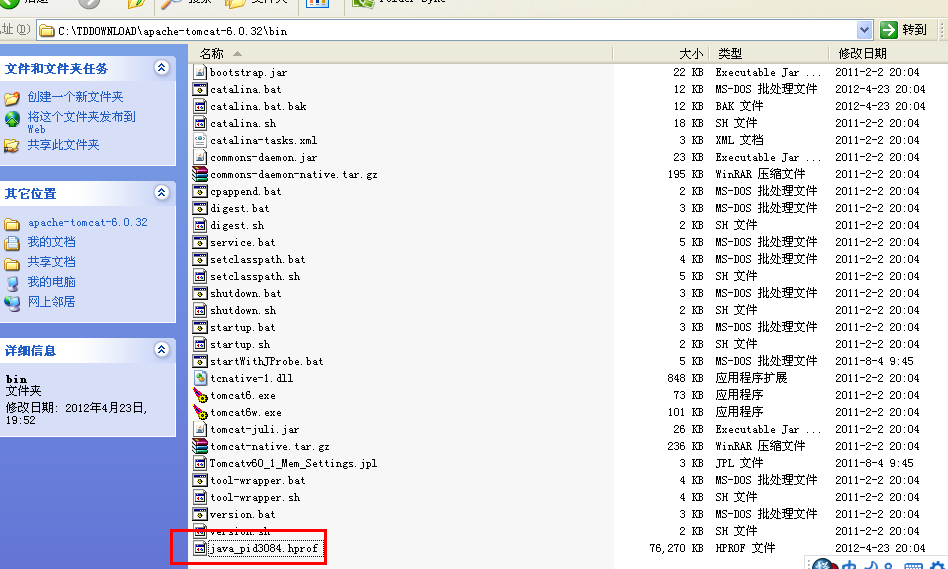
at java.util.ArrayList.add(ArrayList.java:351)

at com.yxie.test.ThreadTest.run(ThreadTest.java:17)

Exception in thread "Thread-12" java.lang.OutOfMemoryError: Java heap space

Exception in thread "Thread-11" java.lang.OutOfMemoryError: Java heap space

生成的hprof文件在:



具体分析请参见后面统一说明。

## Jboss应用配置JVM参数，及生成hprof文件

简单的linux jboss配置，和windows jboss配置是一样的。需要修改run.sh文件。对应windows jboss配置修改就好啦。请参见: windows 平台使用Mat Jboss应用配置JVM参数，及生成hprof文件

Alibaba的项目，用的是jboss，如果我们需要生成heap dump可以使用jmp或者添加对应的参数。

1. 配置jvm参数

我是通过定时器来测试的。需要修改: /home/xieyun/work/standalone/superdeal/deploy/autoconf/promotionProductSoldOutNapoliTimer.sh.vm

JAVA\_OPTS="-Xdebug -Xrunjdwp:transport=dt\_socket,address=8787,server=y,suspend=n"

$JAVA\_HOME/bin/java -Xmx512m -Dwget=true $JAVA\_OPTS com.alibaba.intl.standalone.wholesale.superdeal.main.PromotionProductSoldOutNapoliTimer -XX:+HeapDumpOnOutOfMemoryError

1. 启动jboss,然后通过jmap命令生成对应hprof文件

jmap -dump:format=b,file=/home/xieyun/test/matTtempFile/heapDump.hprof 30146

3、通过mat工具进行分析。

Linux web应用intl-wholesale

需要修改/alibaba.intl.wholesale.deploy-1.0-SNAPSHOT/autoconf/env.vm文件。修改的代码对应

export PRODUCTION\_MODE=dev

$PRODUCTION\_MODE = "dev"模式的代码。如:

elif [ $PRODUCTION\_MODE = "dev" ]; then

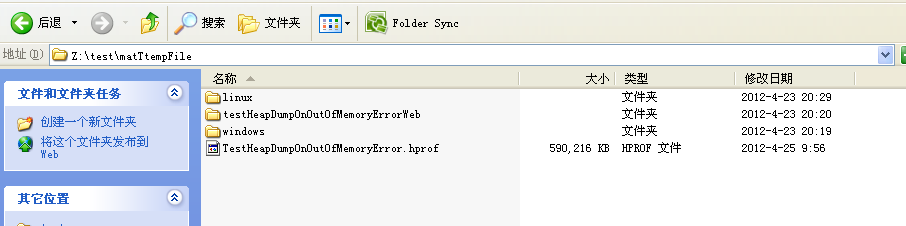
#we should reduce resource usage on developing mode

JAVA\_MEM\_OPTS=" -Xms64m -Xmx1024m -XX:MaxPermSize=128m -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/xieyun/test/matTtempFile/TestHeapDumpOnOutOfMemoryError.hprof"

JAVA\_OPTS=" $JAVA\_MEM\_OPTS $JAVA\_DEBUG\_OPT "

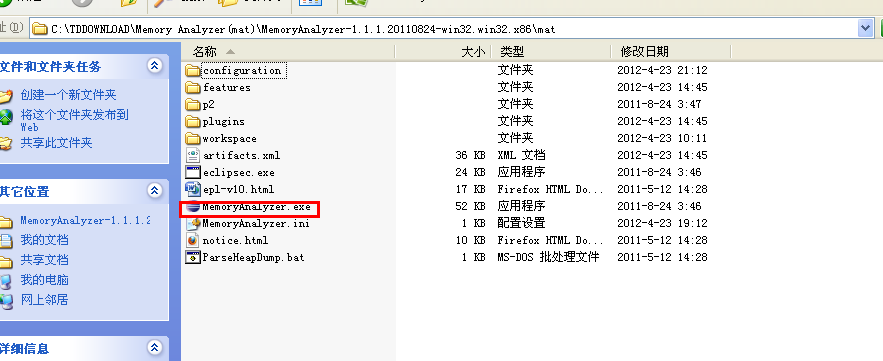
启动服务。然后访问<http://www.aliexpress.com:1080/proengine/release/displayPromotionMultipriceSearchBackDoor.htm>

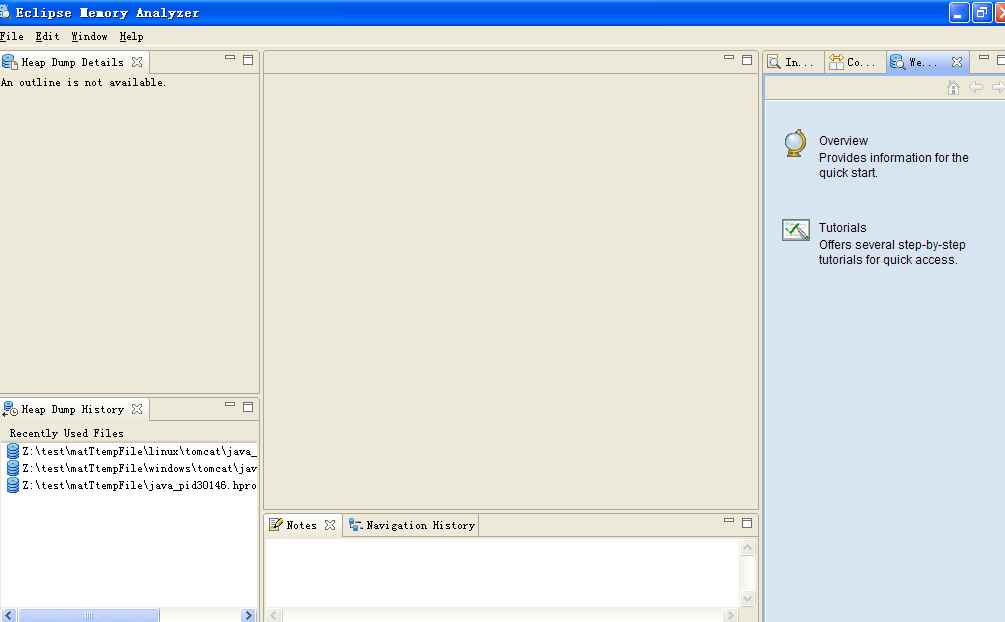
该页面会调用多线程导致outOfMemery错误。会生成hprof文件:



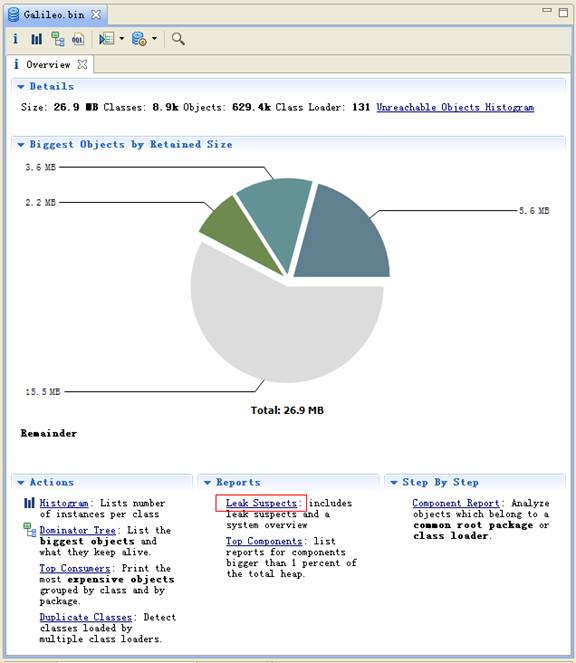
# 使用mat进行分析hprof文件

首先，启动前面安装配置好的 Memory Analyzer tool , 然后选择菜单项 File- Open Heap Dump 来加载需要分析的堆转储文件。文件加载完成后，你可以看到如图 4 所示的界面：





点击 File 🡺 Open Heap Dump。导入堆快照。

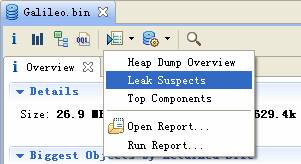


通过上面的概览，我们对内存占用情况有了一个总体的了解。先检查一下 MAT 生成的一系列文件。

**图 5. 文件列表**  


可以看到 MAT 工具提供了一个很贴心的功能，将报告的内容压缩打包到一个 zip 文件，并把它存放到原始堆转储文件的存放目录下，这样如果您需要和同事一起分析这个内存问题的话，只需要把这个小小的 zip 包发给他就可以了，不需要把整个堆文件发给他。并且整个报告是一个 HTML 格式的文件，用浏览器就可以轻松打开。

接下来我们就可以来看看生成的报告都包括什么内容，能不能帮我们找到问题所在吧。您可以点击工具栏上的 Leak Suspects 菜单项来生成内存泄露分析报告，也可以直接点击饼图下方的 Reports->Leak Suspects 链接来生成报告。

**图 6. 工具栏菜单**  


分析三步曲

通常我们都会采用下面的“三步曲”来分析内存泄露问题：

首先，对问题发生时刻的系统内存状态获取一个整体印象。

第二步，找到最有可能导致内存泄露的元凶，通常也就是消耗内存最多的对象

接下来，进一步去查看这个内存消耗大户的具体情况，看看是否有什么异常的行为。

下面将用一个基本的例子来展示如何采用“三步曲”来查看生产的分析报告。

查看报告之一：内存消耗的整体状况

**图 7. 内存泄露分析报告**  


如图 7 所示，在报告上最醒目的就是一张简洁明了的饼图，从图上我们可以清晰地看到一个可疑对象消耗了系统 99% 的内存。

在图的下方还有对这个可疑对象的进一步描述。我们可以看到内存是由 java.util.Vector 的实例消耗的，com.ibm.oti.vm.BootstrapClassLoader 负责这个对象的加载。这段描述非常短，但我相信您已经可以从中找到很多线索了，比如是哪个类占用了绝大多数的内存，它属于哪个组件等等。

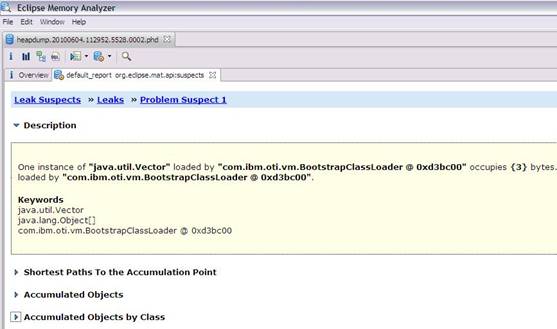
接下来，我们应该进一步去分析问题，为什么一个 Vector 会占据了系统 99% 的内存，谁阻止了垃圾回收机制对它的回收。

查看报告之二：分析问题的所在

首先我们简单回顾下 JAVA 的内存回收机制，内存空间中垃圾回收的工作由垃圾回收器 (Garbage Collector,GC) 完成的，它的核心思想是：对虚拟机可用内存空间，即堆空间中的对象进行识别，如果对象正在被引用，那么称其为存活对象，反之，如果对象不再被引用，则为垃 圾对象，可以回收其占据的空间，用于再分配。

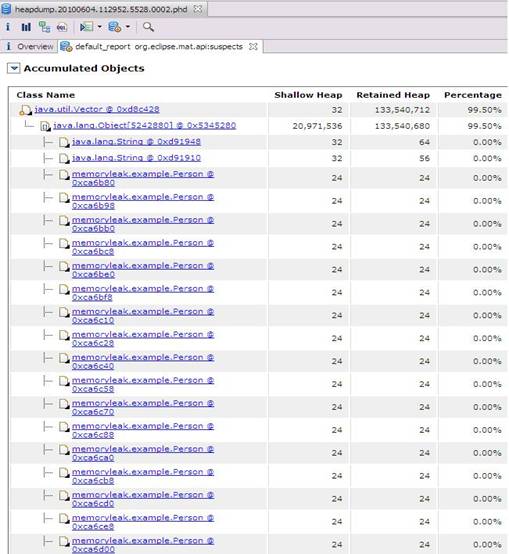
在垃圾回收机制中有一组元素被称为根元素集合，它们是一组被虚拟机直接引用的对象，比如，正在运行的线程对象，系统调用栈里面的对象以及被 system class loader 所加载的那些对象。堆空间中的每个对象都是由一个根元素为起点被层层调用的。因此，一个对象还被某一个存活的根元素所引用，就会被认为是存活对象，不能被回收，进行内存释放。因此，我们可以通过分析一个对象到根元素的引用路径来分析为什么该对象不能被顺利回收。如果说一个对象已经不被任何程序逻辑所需要但是还存在被根元素引用的情况，我们可以说这里存在内存泄露。

现在，让我们开始真正的寻找内存泄露之旅，点击“Details ”链接，可以看到如图 8 所示对可疑对象 1 的详细分析报告。

**图 8. 可疑对象 1 的详细分析报告**  


1. 我们查看下从 GC 根元素到内存消耗聚集点的最短路径：  
   **图 9. 从根元素到内存消耗聚集点的最短路径**  
   

我们可以很清楚的看到整个引用链，内存聚集点是一个拥有大量对象的集合，如果你对代码比较熟悉的话，相信这些信息应该能给你提供一些找到内存泄露的思路了。

接下来，我们再继续看看，这个对象集合里到底存放了什么，为什么会消耗掉如此多的内存。  
**图 10. 内存消耗聚集对象信息**  


在这张图上，我们可以清楚的看到，这个对象集合中保存了大量 Person 对象的引用，就是它导致的内存泄露。

至此，我们已经拥有了足够的信息去寻找泄露点，回到代码，我们发现，是下面的代码导致了内存泄露 :

**清单 1. 内存泄漏的代码段**

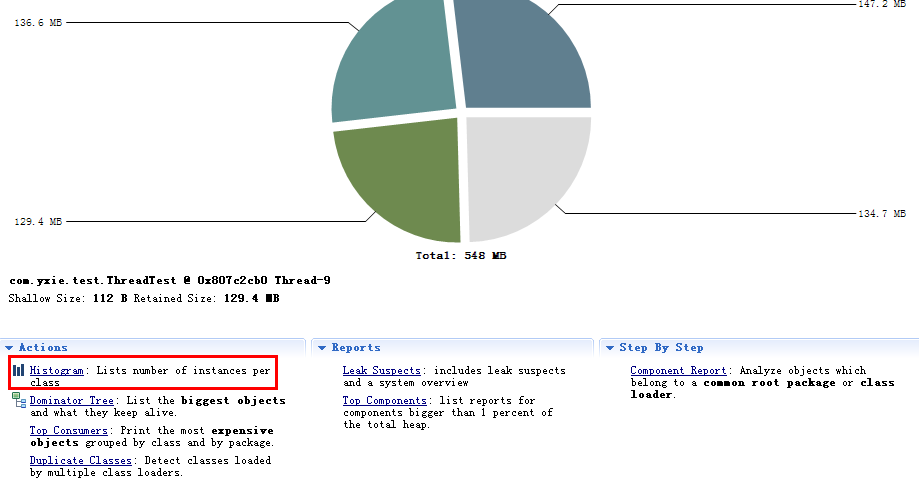
|  |
| --- |
| while (1<2)  {  Person person = new Person("name","address",i);  v.add(person);  person = null;  } |

总结

从上面的例子我们可以看到用 MAT 来进行堆转储文件分析，寻找内存泄露非常简单，尤其是对于新手而言，这是一个很好的辅助分析工具。但是，MAT 绝对不仅仅是一个“傻瓜式”内存分析工具，它还提供很多高级功能，比如 MAT 支持用 OQL（Object Query Language）对 heap dump 中的对象进行查询，支持对线程的分析等，有关这些功能的使用可以参考 MAT 的帮助文档。

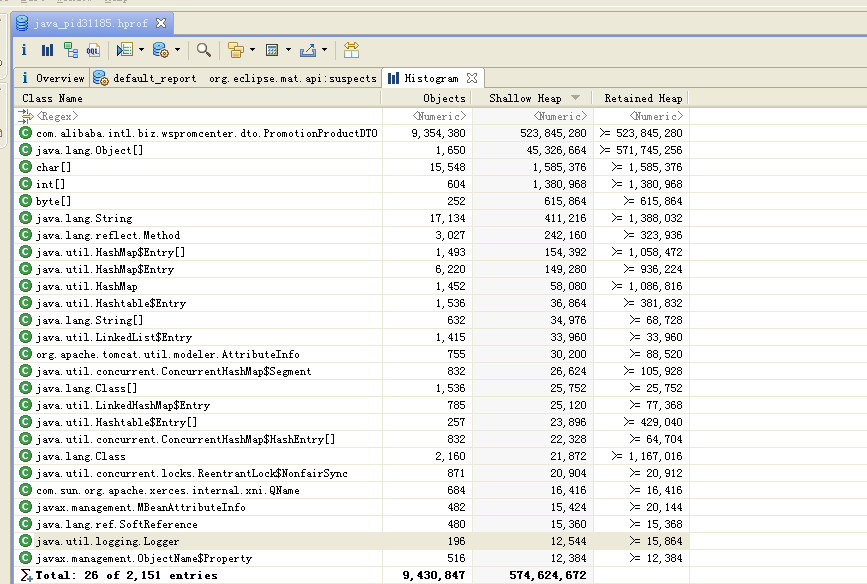
# Mat工具其他功能说明

## Histogram

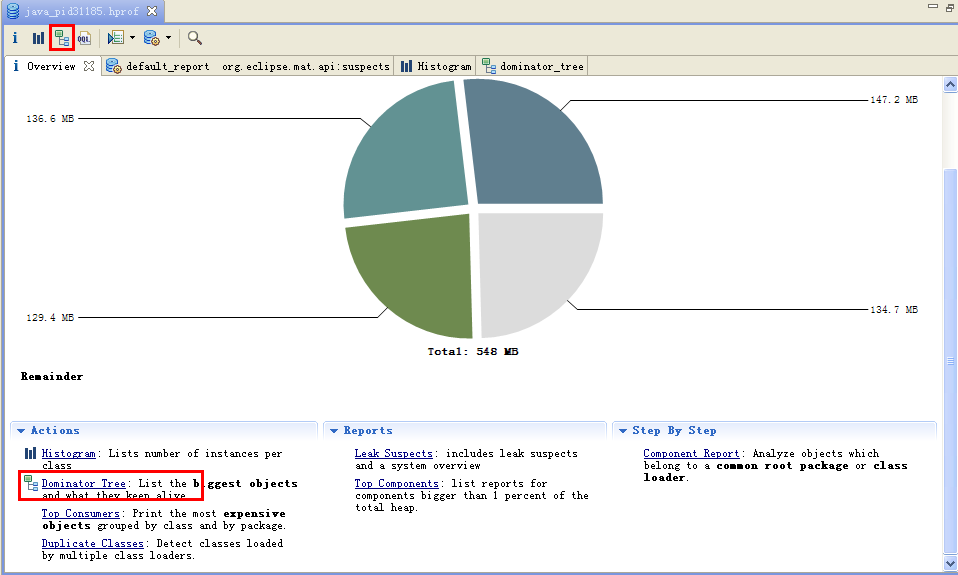


The histogram lists the objects grouped by their class. The Memory Analyzer can approximate the retained size very quickly. This is a good indicator where to continue with the analysis.

直方图列出了他们的分组class。内存分析器可以近似保留的大小，速度非常快。这是一个很好的继续分析的指标。

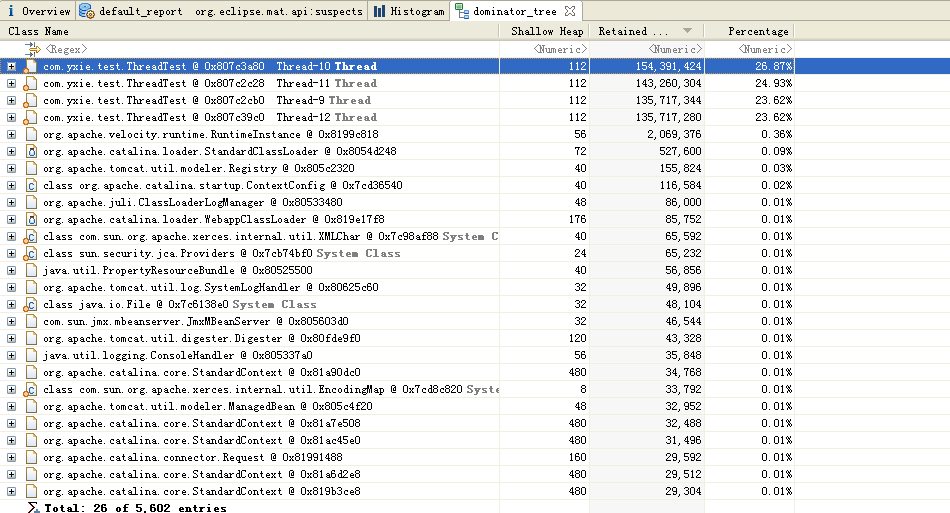


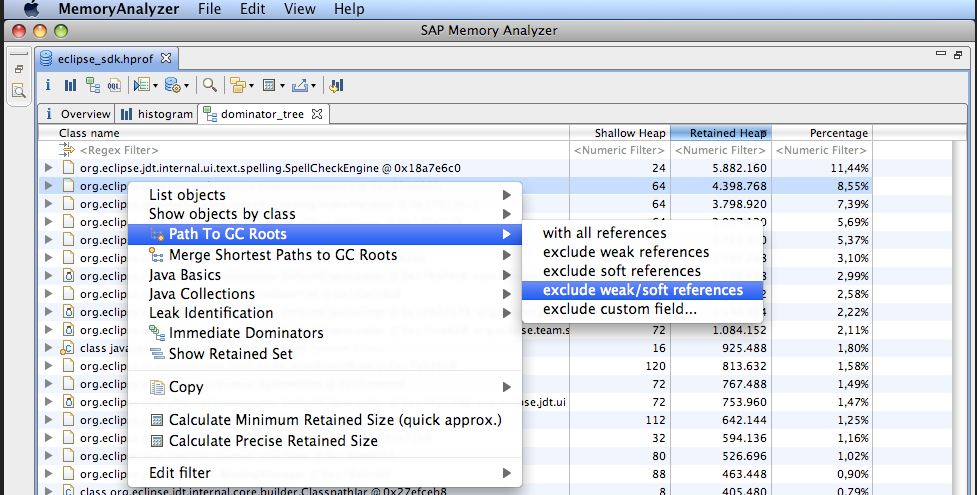
## Dominator Tree



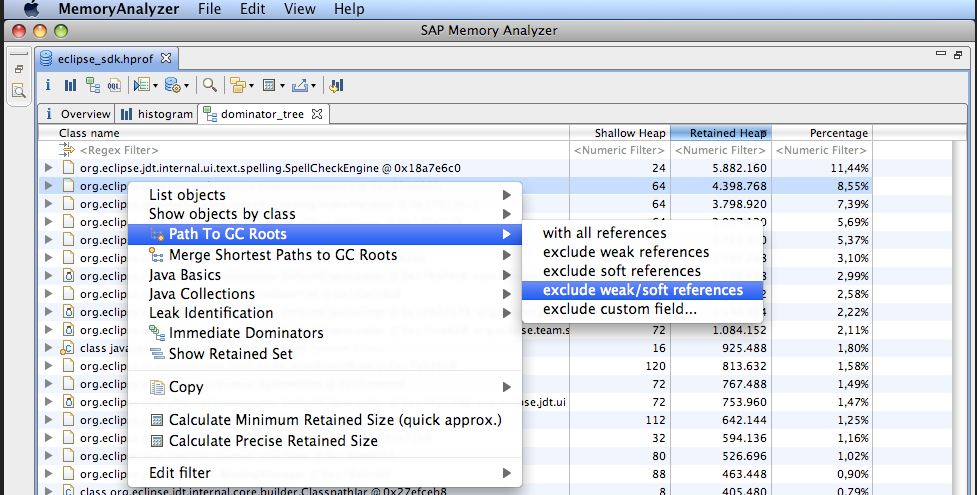
The Dominator Tree lists the biggest objects. One could call it a "Keep-Alive Tree" because the next level shows those objects which are immediately prevented from being garbage collected. Right-click to drill down: view the outgoing and incoming reference or view the path to the GC roots to see the reference chain which keeps the objects alive.

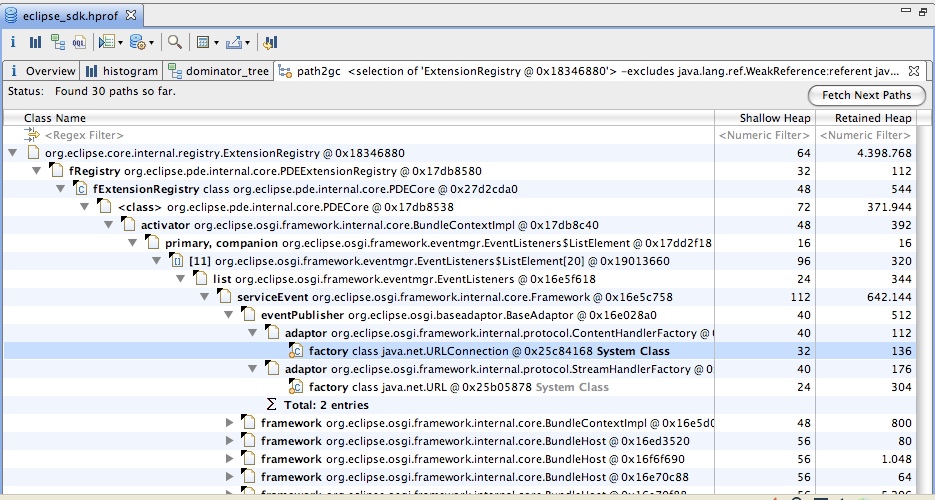
支配树列出了最大的对象。人们可以把它称为“保持活动树”，因为一个新的水平显示那些被立即防止被垃圾收集的对象。右键单击向下钻取：查看传出和传入的参考或查看GC根的路径，看到这使活着的对象的引用链。





## Path to GC Roots



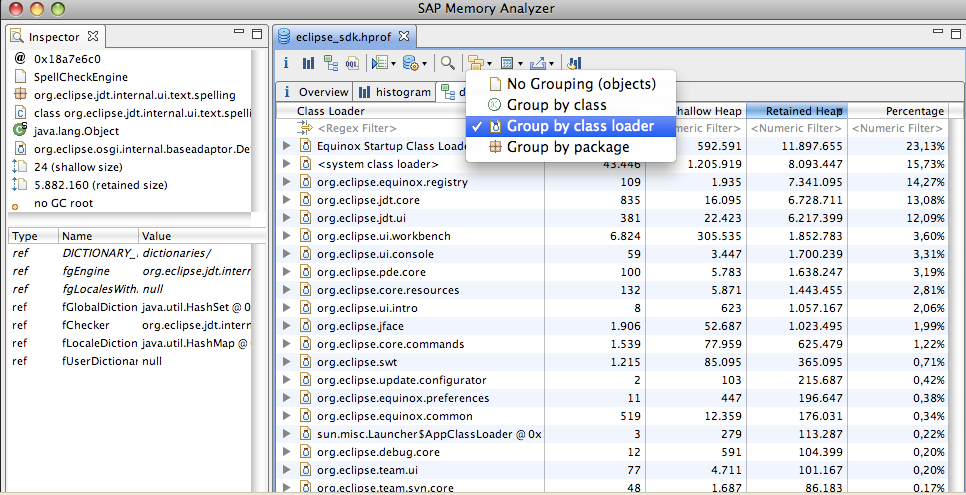


The path to the GC Roots shows the reference chain which prevents that the object is garbage collected. Objects decorated with a yellow dot are Garbage Collection (GC) Roots, i.e. objects which are assumed to be alive. Usually GC Roots are objects that are currently on the call stack of a thread or system classes.

GC根的路径，显示防止垃圾收集的对象是参考链。饰有黄点的对象，垃圾收集（GC）的根源，即对象，这被认为是活着的。通常，GC根是当前一个线程或系统类的调用堆栈的对象。

## Dominator Tree Grouped by Class Loader

按类装载器的支配树



Any decent architectures loads components by different class loaders. Many views of the Memory Analyzer allow you to group the objects by class loader and, hence, easily analyze memory by component. To map the class loader to a meaningful component name (for example the plug-in id), one can plug-in name resolver.

任何像样的架构，由不同的类加载器加载的组件。内存分析器的许多意见，让你由类加载器的对象进行分组，因此，很容易分析由组件的内存。要映射的类加载器的一个有意义的组件的名称（例如插件ID），可以插入名称解析。

http://www.eclipse.org/mat/about/screenshots.php

# 其他资料

<http://www.ibm.com/developerworks/cn/opensource/os-cn-ecl-ma/index.html>

<http://wiki.eclipse.org/MemoryAnalyzer/Extending_Memory_Analyzer>

<http://blog.csdn.net/fenglibing/article/details/6298326>

<http://blog.sina.com.cn/s/blog_53db93d80100kjc6.html>

<http://blog.sina.com.cn/s/blog_53db93d80100kjc6.html>

<http://www.blogjava.net/rosen/archive/2010/05/21/321575.html>

<http://www.blogjava.net/rosen/archive/2010/06/13/323522.html>

<http://blog.csdn.net/zouxueping/article/details/6751884>

http://memoryanalyzer.blogspot.com/

安装: http://sapnwnewbie.blogspot.com/2010/10/installing-memory-analyser-tool-for.html