

ImportNew

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java/小组](#)
- [工具资源](#)

- 导航条 - ▾

JVM (7) : JVM调优-工具篇

2017/03/07 | 分类: [基础技术](#) | [2 条评论](#) | 标签: [JVM](#)

分享到:

原文出处: [纯洁的微笑](#)



16年的时候花了一些时间整理了一些关于jvm的介绍文章,到现在回顾起来还是一些还没有补充全面,其中就包括如何利用工具来监控调优前后的性能变化。工具做为图形化界面来展示更能直观地发现问题,另一方面一些耗费性能的分析(dump文件分析)一般也不会在生产直接分析,往往dump下来的文件达1G左右,人工分析效率较低,因此利用工具来分析jvm相关问题,长长可以到达事半功倍的效果来。

jvm监控分析工具一般分为两类,一种是jdk自带的工具,一种是第三方的分析工具。jdk自带工具一般在jdk bin目录下面,以exe的形式直接点击就可以使用,其中包含分析工具已经很强大,几乎涉及了方方面面,但是我们最常使用的只有两款:jconsole.exe和jvisualvm.exe;第三方的分析工具有很多,各自的侧重点不同,比较有代表性的:MAT(Memory Analyzer Tool)、GChisto等。

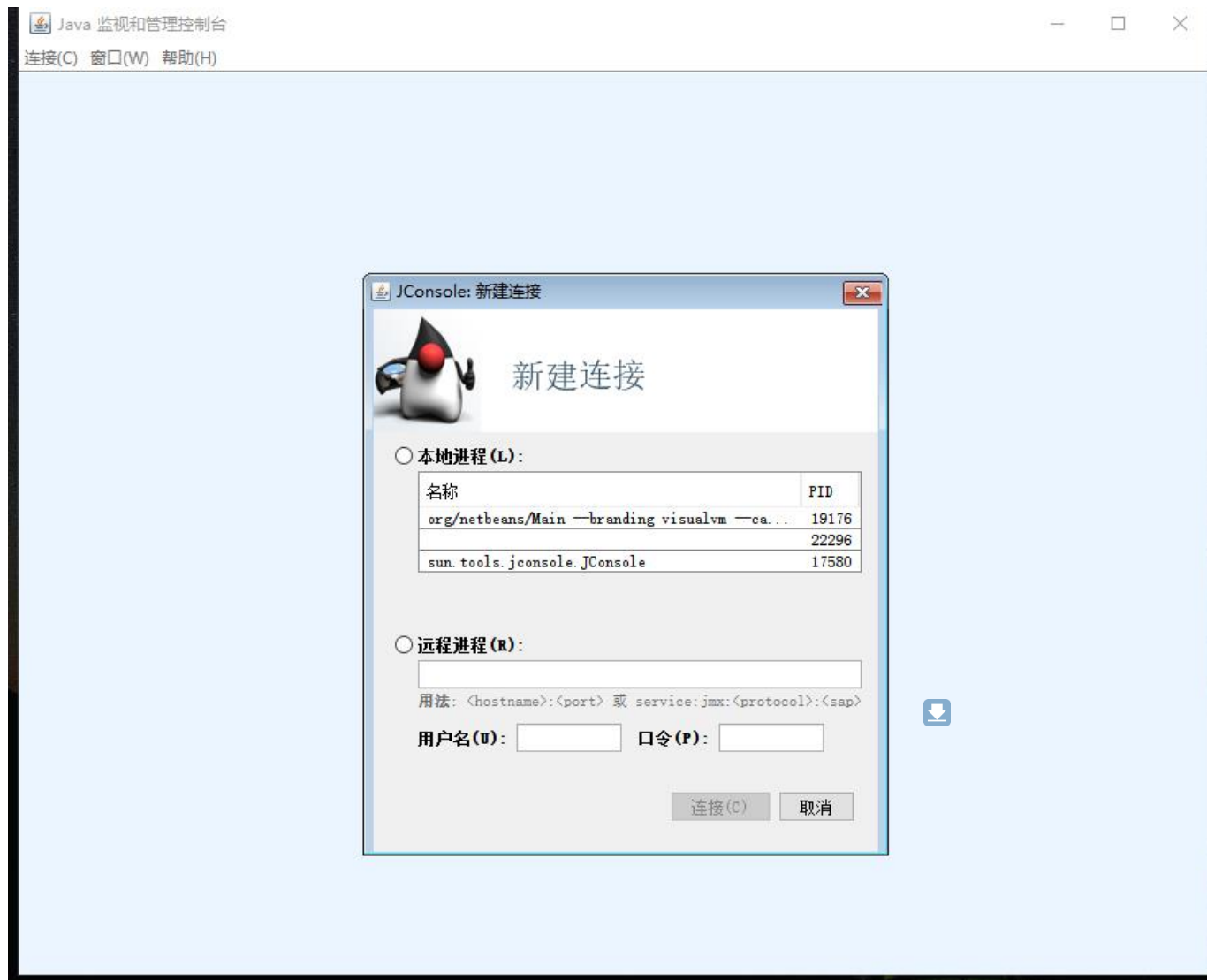
对于大型 JAVA 应用程序来说,再精细的测试也难以堵住所有的漏洞,即便我们在测试阶段进行了大量卓有成效的工作,很多问题还是会在生产环境下暴露出来,并且很难在测试环境中进行重现。JVM 能够记录下问题发生时系统的部分运行状态,并将其存储在堆转储 (Heap Dump) 文件中,从而为我们分析和诊断问题提供了重要的依据。其中VisualVM和MAT是dump文件的分析利器。

jdk自带的工具

jconsole

Jconsole (Java Monitoring and Management Console) 是从java5开始,在JDK中自带的java监控和管理控制台,用于对JVM中内存,线程和类等的监控,是一个基于JMX (java management extensions) 的GUI性能监测工具。jconsole使用jvm的扩展机制获取并展示虚拟机中运行的应用程序的性能和资源消耗等信息。

直接在jdk/bin目录下点击jconsole.exe即可启动，界面如下：

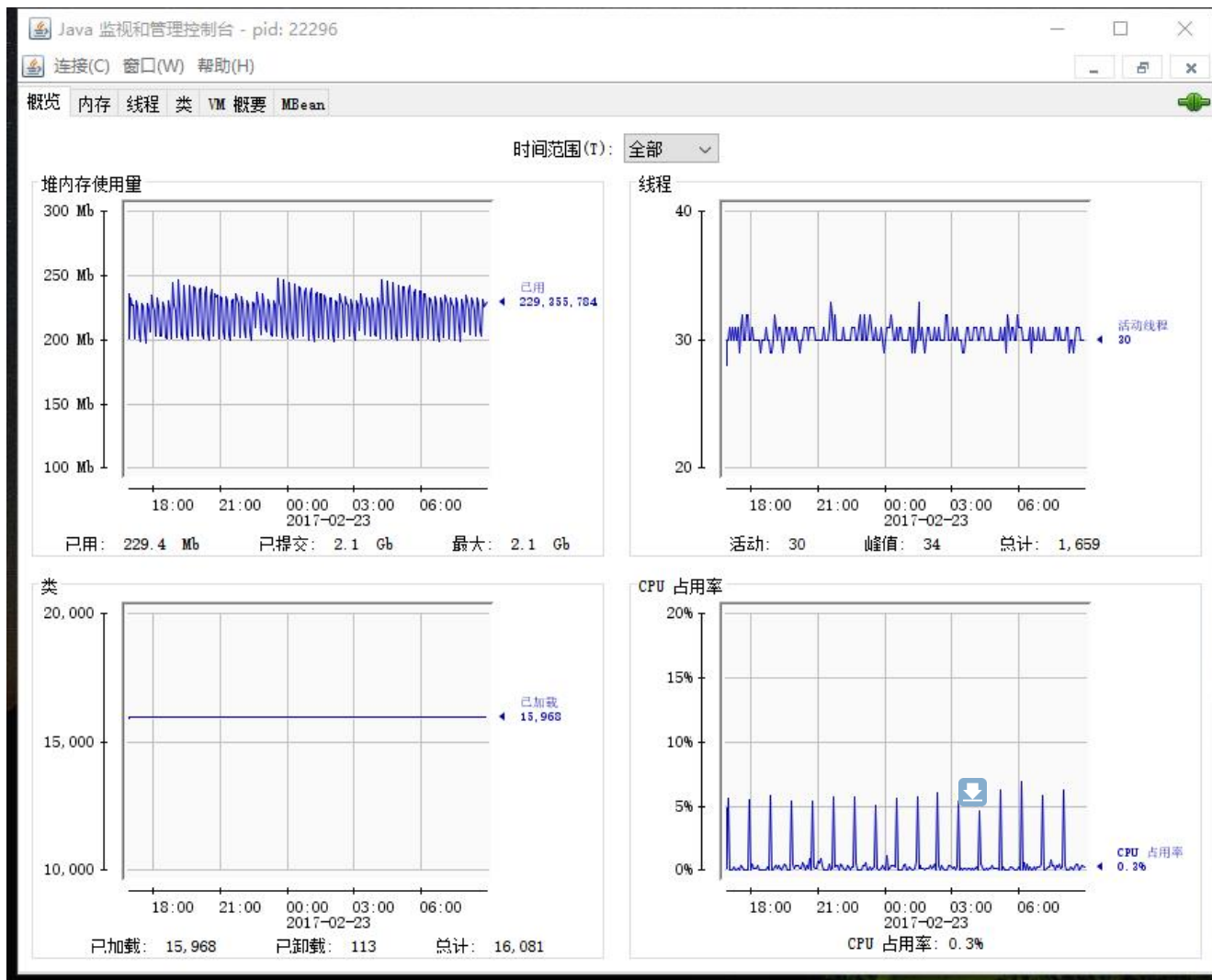


在弹出的框中可以选择本机的监控本机的java应用，也可以选择远程的java服务来监控，如果监控远程服务需要在tomcat启动脚本中添加如下代码：

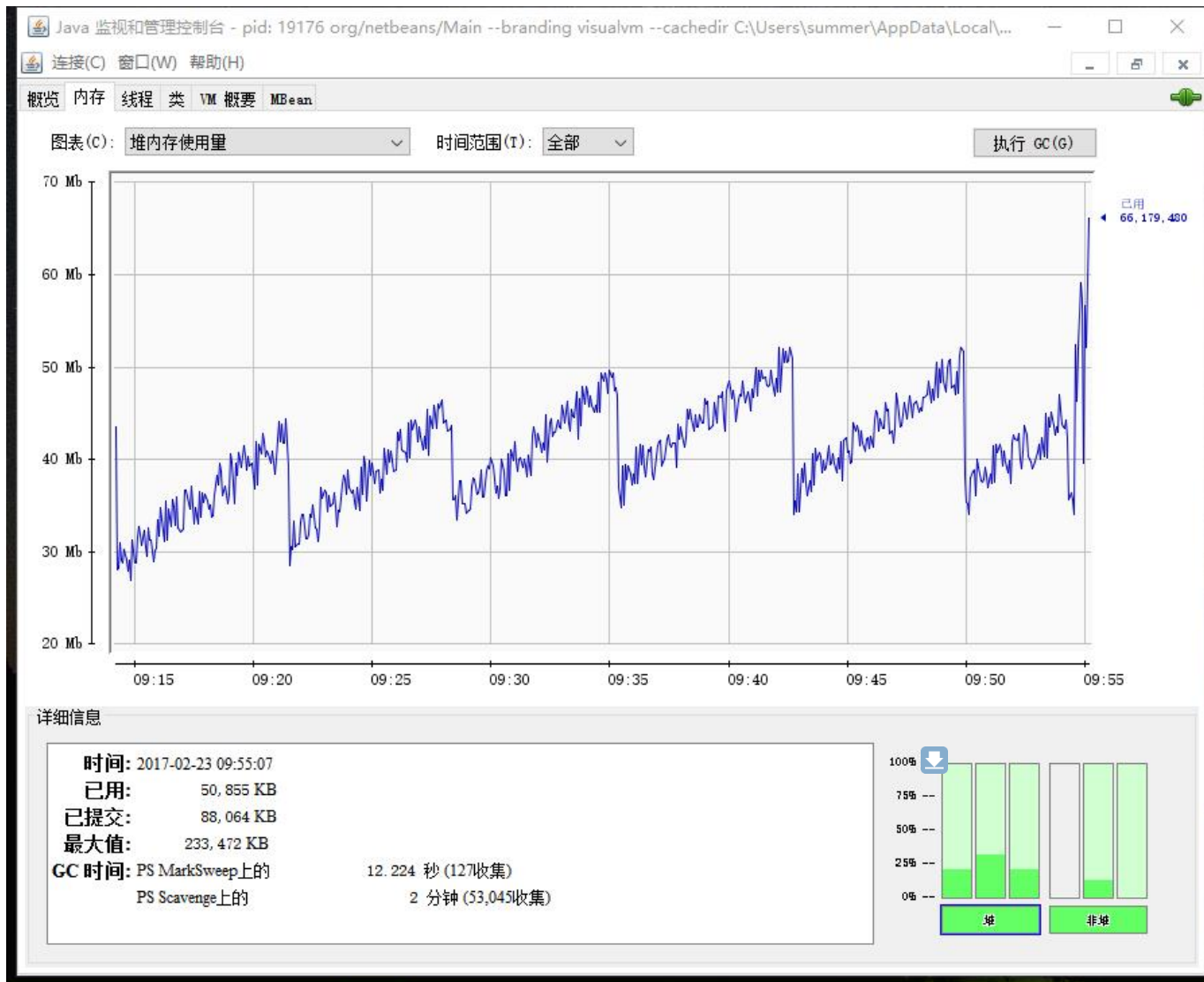
```
1 -Dcom.sun.management.jmxremote.port=6969
2 -Dcom.sun.management.jmxremote.ssl=false
3 -Dcom.sun.management.jmxremote.authenticate=false
```

连接进去之后，就可以看到jconsole概览图和主要的功能：概述、内存、线程、类、VM、MBeans

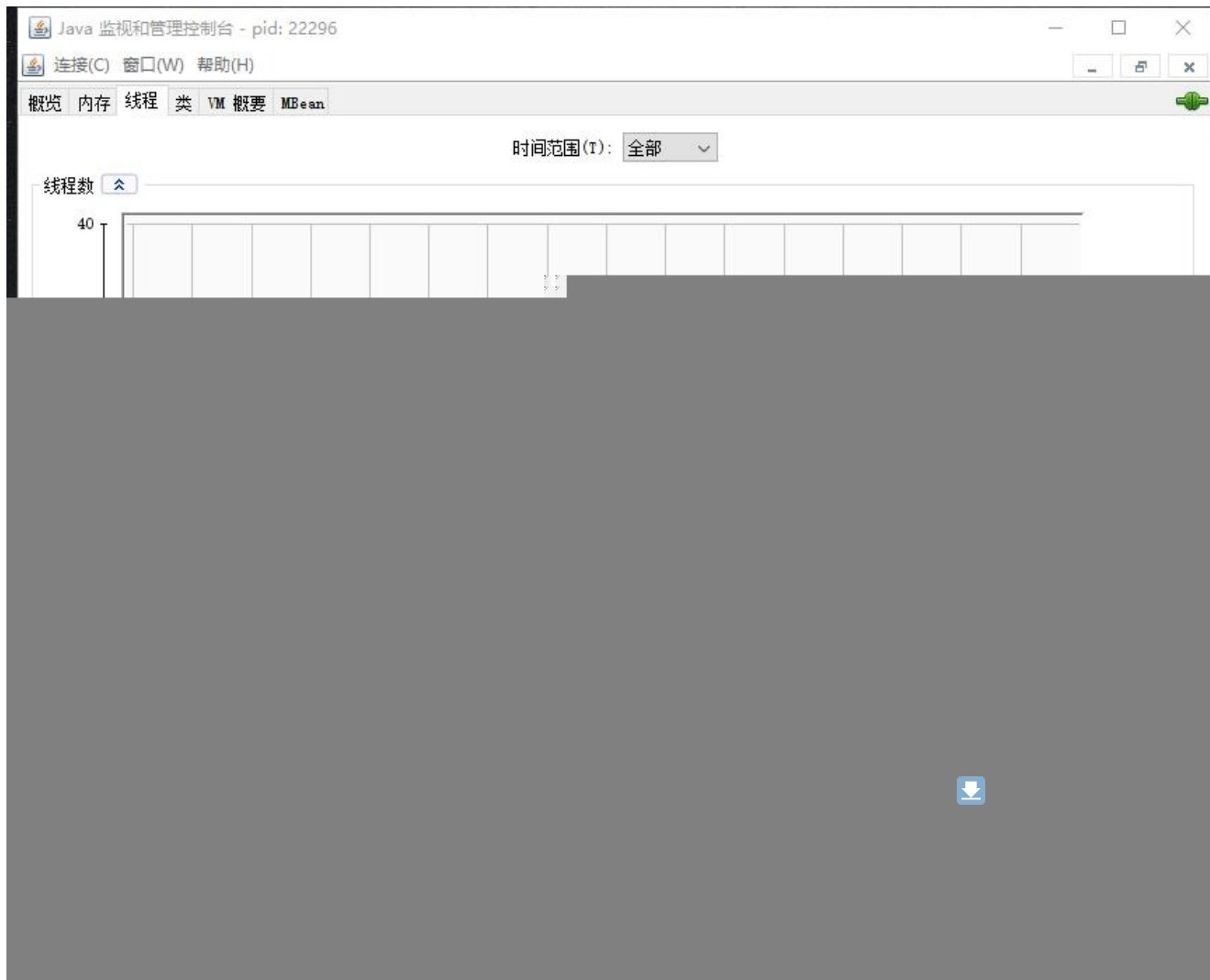
- 概述，以图表的方式显示出堆内存使用量，活动线程数，已加载的类，CUP占用率的折线图，可以非常清晰的观察在程序执行过程中的变动情况。



- 内存，主要展示了内存的使用情况，同时可以查看堆和非堆内存的变化值对比，也可以点击执行GC来处罚GC的执行



- 线程，主界面展示线程数的活动数和峰值，同时点击左下方线程可以查看线程的详细信息，比如线程的状态是什么，堆栈内容等，同时也可以点击“检测死锁”来检查线程之间是否有死锁的情况。



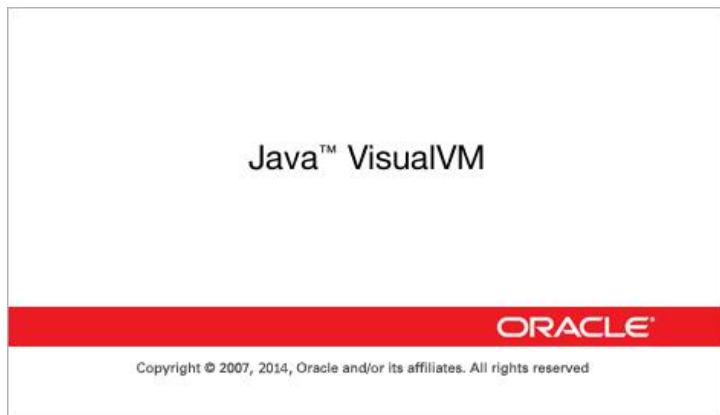
- 类, 主要展示已加载类的相关信息。
- VM 概要, 展示JVM所有信息总览, 包括基本信息、线程相关、堆相关、操作系统、VM参数等。
- Mbean, 查看Mbean的属性, 方法等。

VisualVM

简介

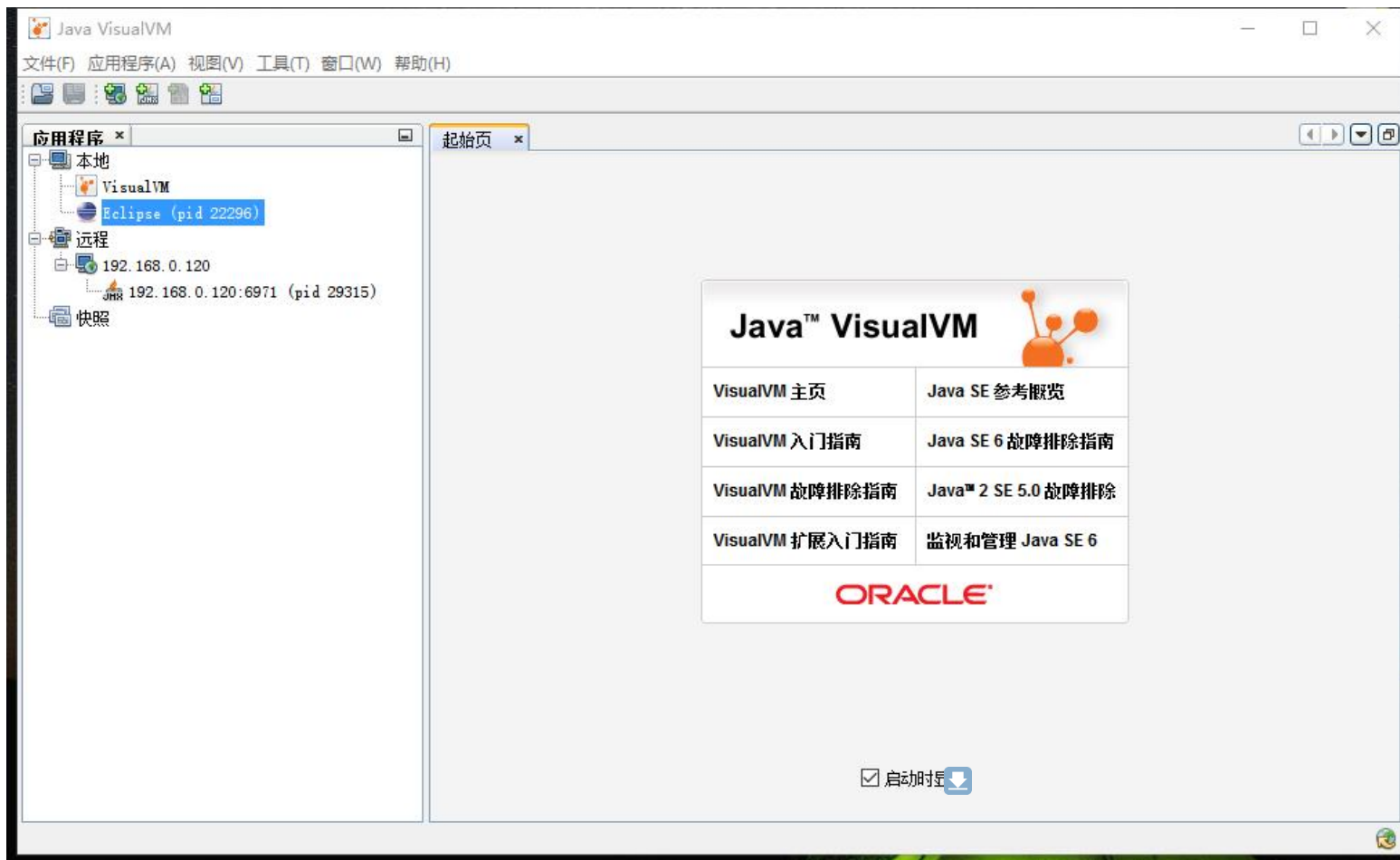
VisualVM 是一个工具, 它提供了一个可视界面, 用于查看 Java 虚拟机 (Java Virtual Machine, JVM) 上运行的基于 Java 技术的应用程序 (Java 应用程序) 的详细信息。VisualVM 对 Java Development Kit (JDK) 工具所检索的 JVM 软件相关数据进行组织, 并通过一种使您可以快速查看有关多个 Java 应用程序的数据的方式提供该信息。您可以查看本地应用程序以及远程主机上

运行的应用程序的相关数据。此外，还可以捕获有关 JVM 软件实例的数据，并将该数据保存到本地系统，以供后期查看或与其他用户共享。

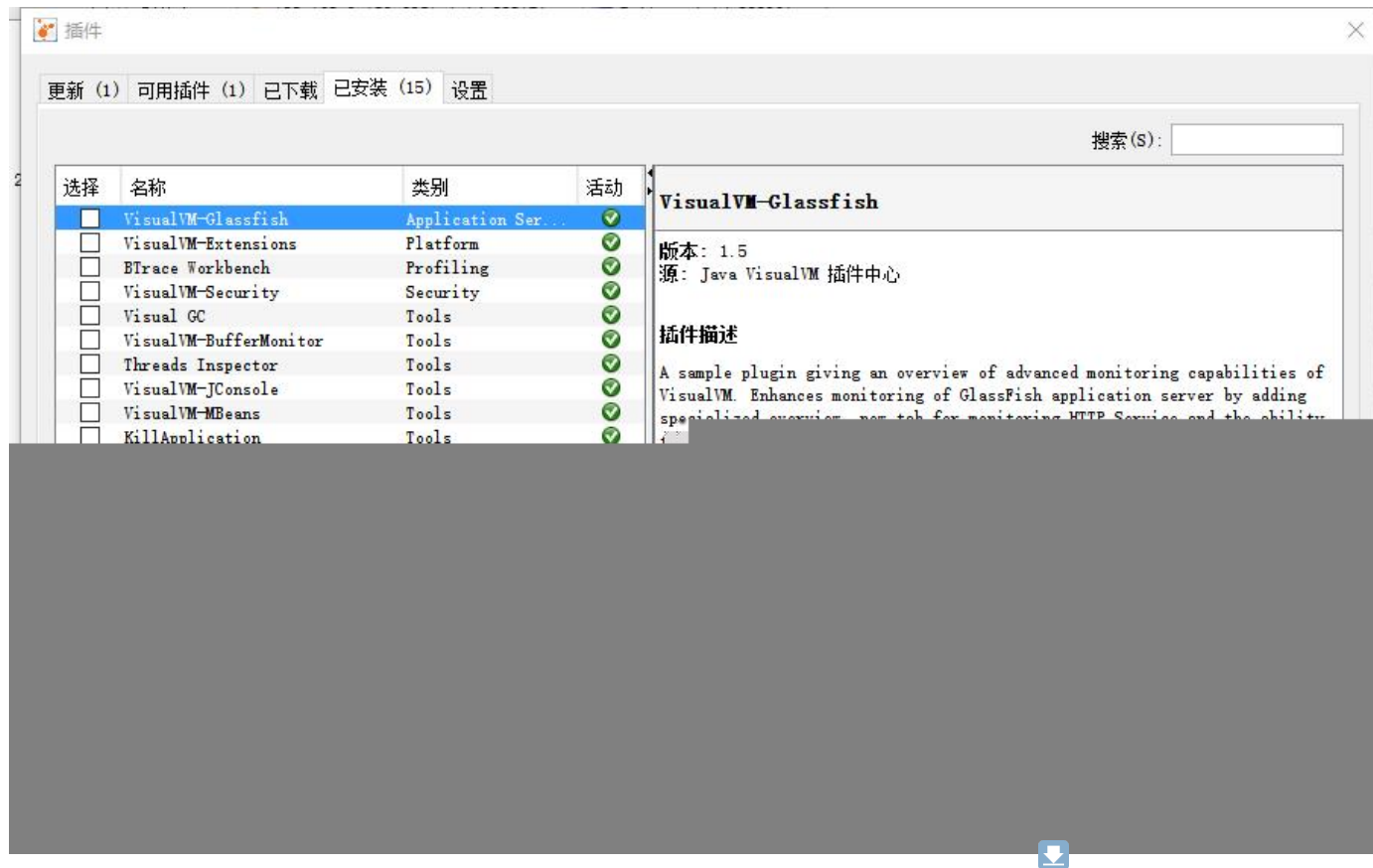


VisualVM 是javajdk自带的最牛逼的调优工具了吧，也是我平时使用最多调优工具，几乎涉及了jvm调优的方方面面。同样是在jdk/bin目录下面双击jvisualvm.exe既可使用，启动起来后和jconsole 一样同样可以选择本地和远程，如果需要监控远程同样需要配置相关参数，主界面如下；





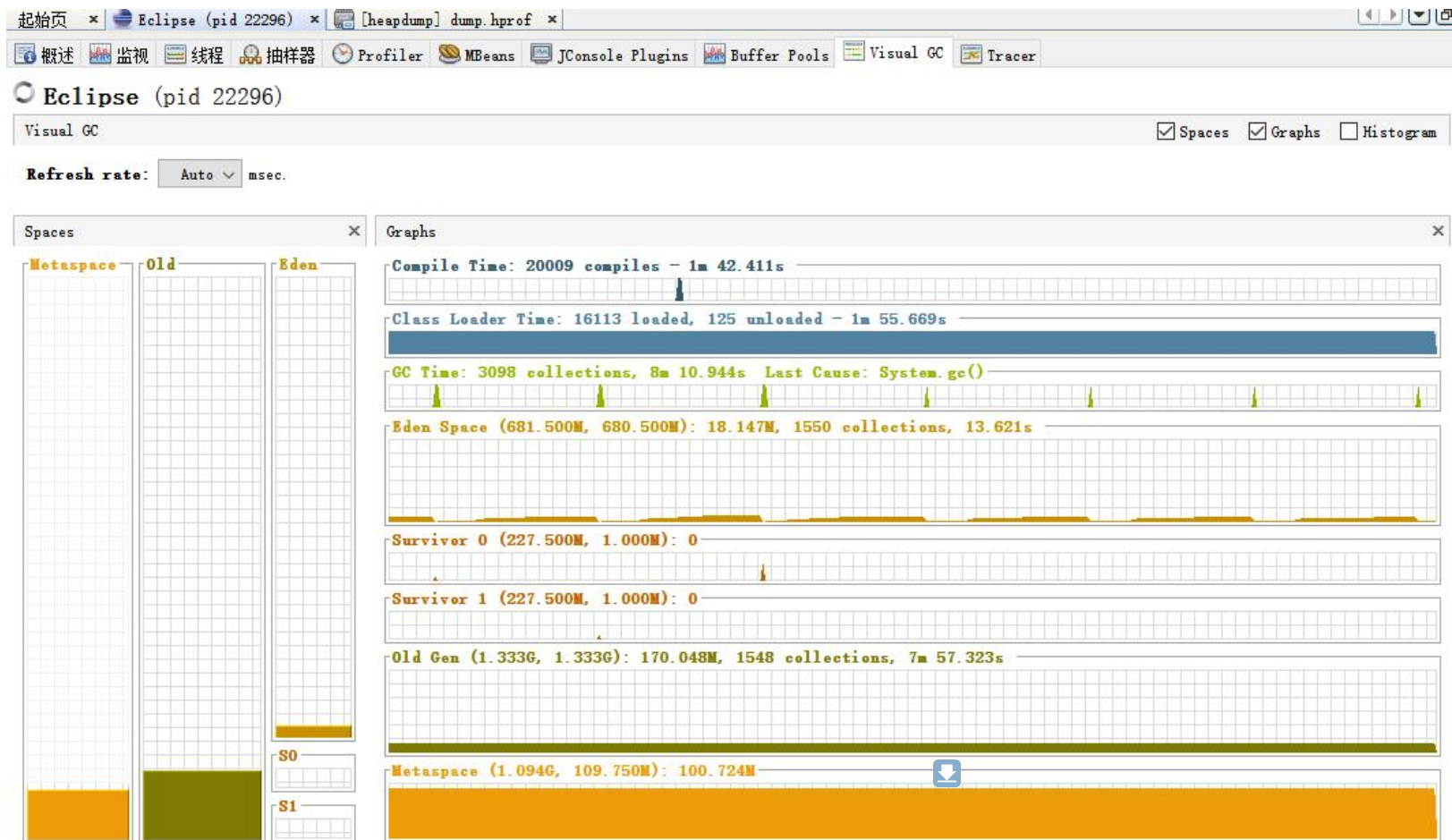
VisualVM可以根据需要安装不同的插件，每个插件的关注点都不同，有的主要监控GC，有的主要监控内存，有的监控线程等。



如何安装:

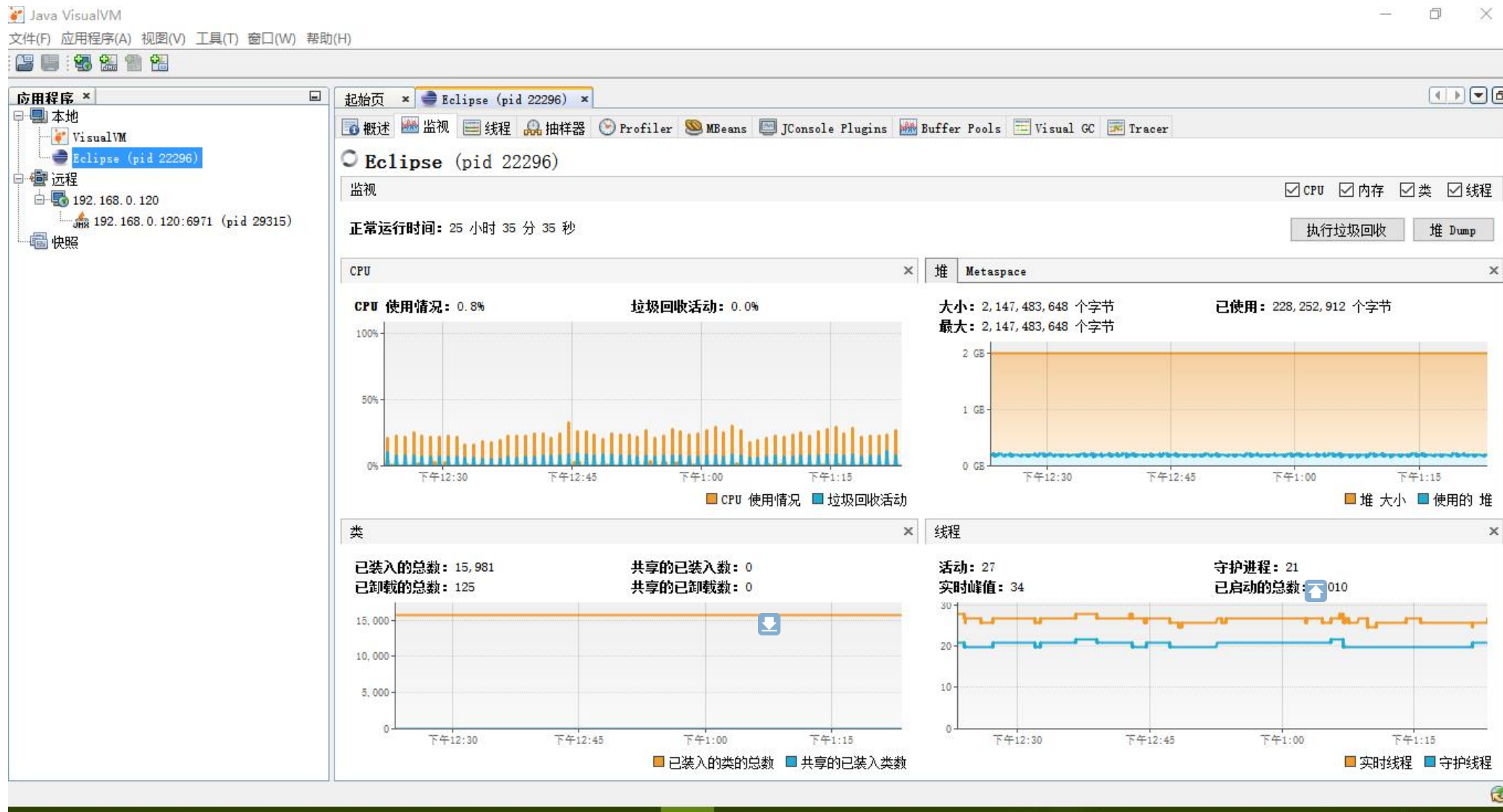
- 1、从主菜单中选择“工具” > “插件”。
- 2、在“可用插件”标签中，选中该插件的“安装”复选框。单击“安装”。
- 3、逐步完成插件安装程序。

我这里以 Eclipse(pid 22296)为例，双击后直接展开，主界面展示了系统和jvm两大块内容，点击右下方jvm参数和系统属性可以参考详细的参数信息。



因为VisualVM的插件太多，我这里主要介绍三个我主要使用几个：监控、线程、Visual GC

监控的主页其实也就是，cpu、内存、类、线程的图表



线程和jconsole功能没有太大的区别

Java VisualVM

文件(F) 应用程序(A) 视图(V) 工具(T) 窗口(W) 帮助(H)

应用程序 x

- 本地
 - VisualVM
 - Eclipse (pid 22296)
- 远程
 - 192.168.0.120
 - 192.168.0.120:6971 (pid 29315)
- 快照

起始页 x Eclipse (pid 22296) x

概述 监视 线程 抽样器 Profiler MBeans JConsole Plugins Buffer Pools Visual GC Tracer

Eclipse (pid 22296)

线程 ☒ 线程可视化 ☒ Threads inspector

实时线程: 27
守护线程: 21

线程 Dump

时间线

视图: 所有线程

名称	13:25:55	13:26:00	13:26:05	13:26:10	运行	总计
Worker-210					0 ms (0%)	1,984,690 ms
Worker-209					0 ms (0%)	1,226,081 ms
RMI TCP Connection(389)-19%					388,345 ms (84.7%)	458,383 ms
JMX server connection time					0 ms (0%)	10,779,430 ms
RMI TCP Connection(381)-19%					625,159 ms (46.5%)	1,344,165 ms

运行 休眠 等待 驻留 监视

Threads inspector

☒ Active Thread: Equinox Container: C
☐ Attach Listener
☐ Bundle File Closer
☐ EMF Reference Cleaner
☐ EventAdmin Async Event Dispatcher I
☐ Finalizer

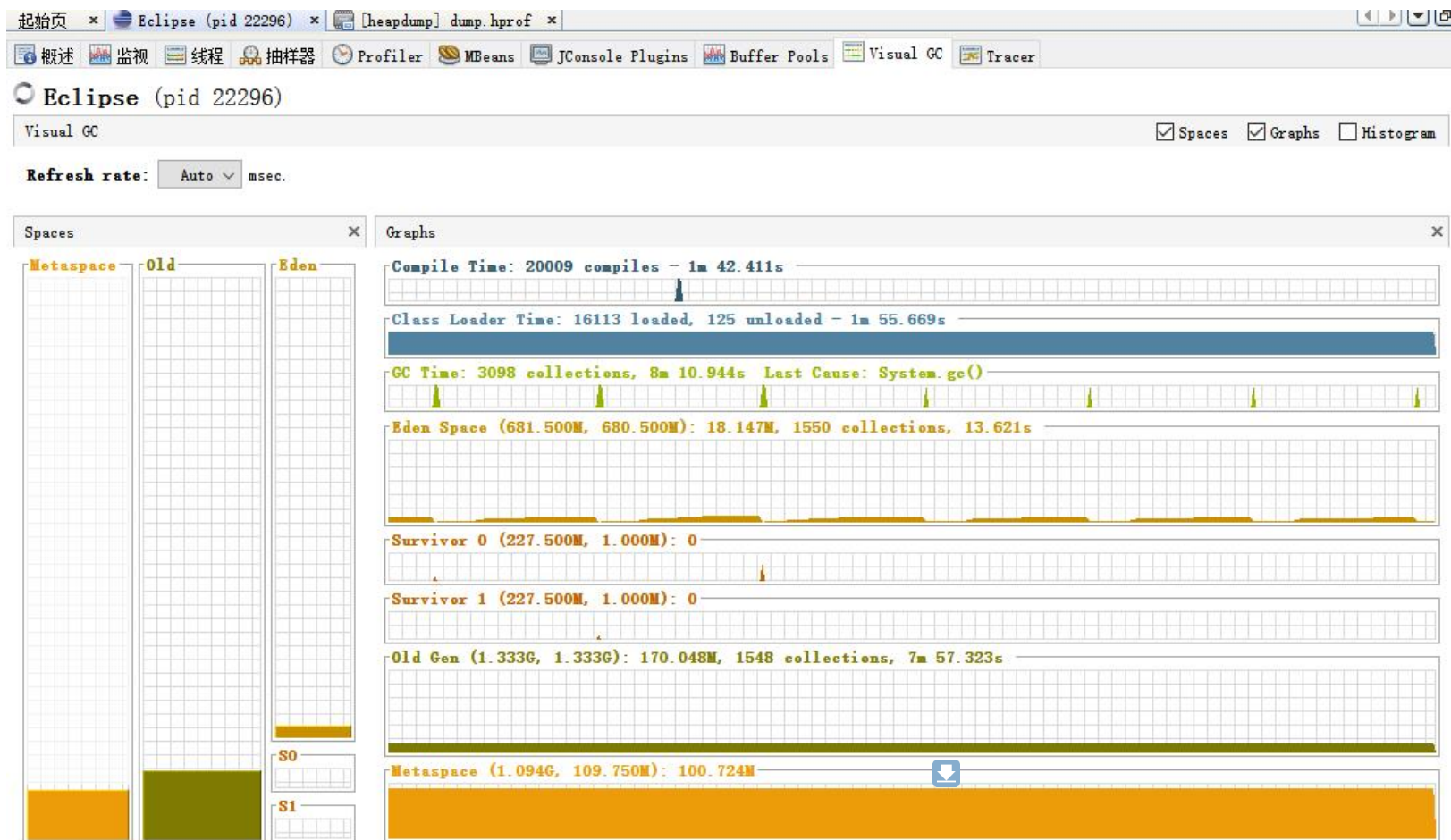
Refresh

2017-02-23 13:26:02

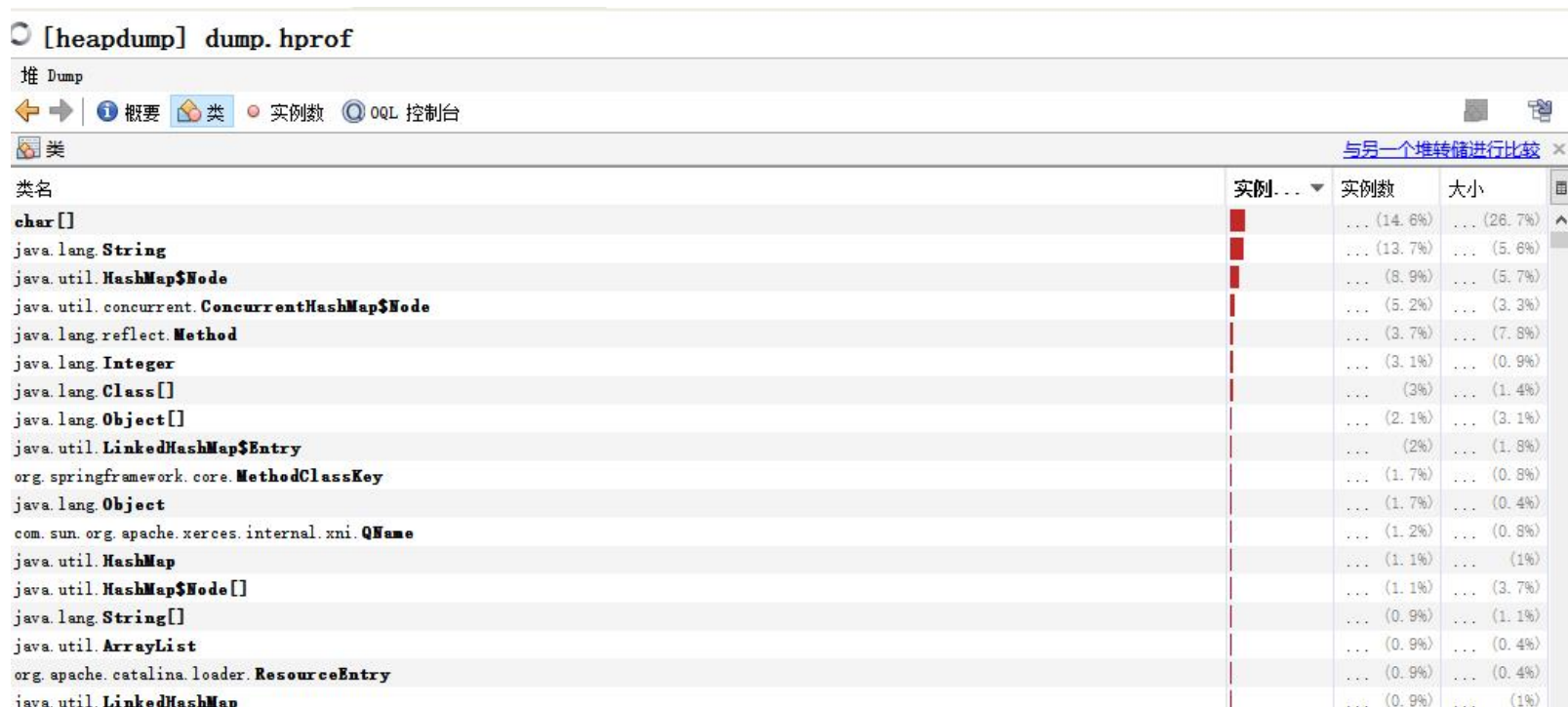
Active Thread: Equinox Container: 00dbbaae-b1f8-0016-1dc3-965c20261b88" - Thread t@11

```
java.lang.Thread.State: TIMED_WAITING
    at sun.misc.Unsafe.park(Native Method)
    - parking to wait for <294897d2> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
    at java.util.concurrent.locks.LockSupport.parkNanos(Unknown Source)
    at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(Unknown Source)
    at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(Unknown Source)
    at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(Unknown Source)
```

Visual GC 是常常使用的一个功能，可以明显的看到年轻代、老年代的内存变化，以及gc频率、gc的时间等。



以上的功能其实jconsole几乎也有，VisualVM更全面更直观一些，另外VisualVM非常多的其它功能，可以分析dump的内存快照，dump出来的线程快照并且进行分析等，还有其它很多的插件大家可以去探索



第三方调优工具

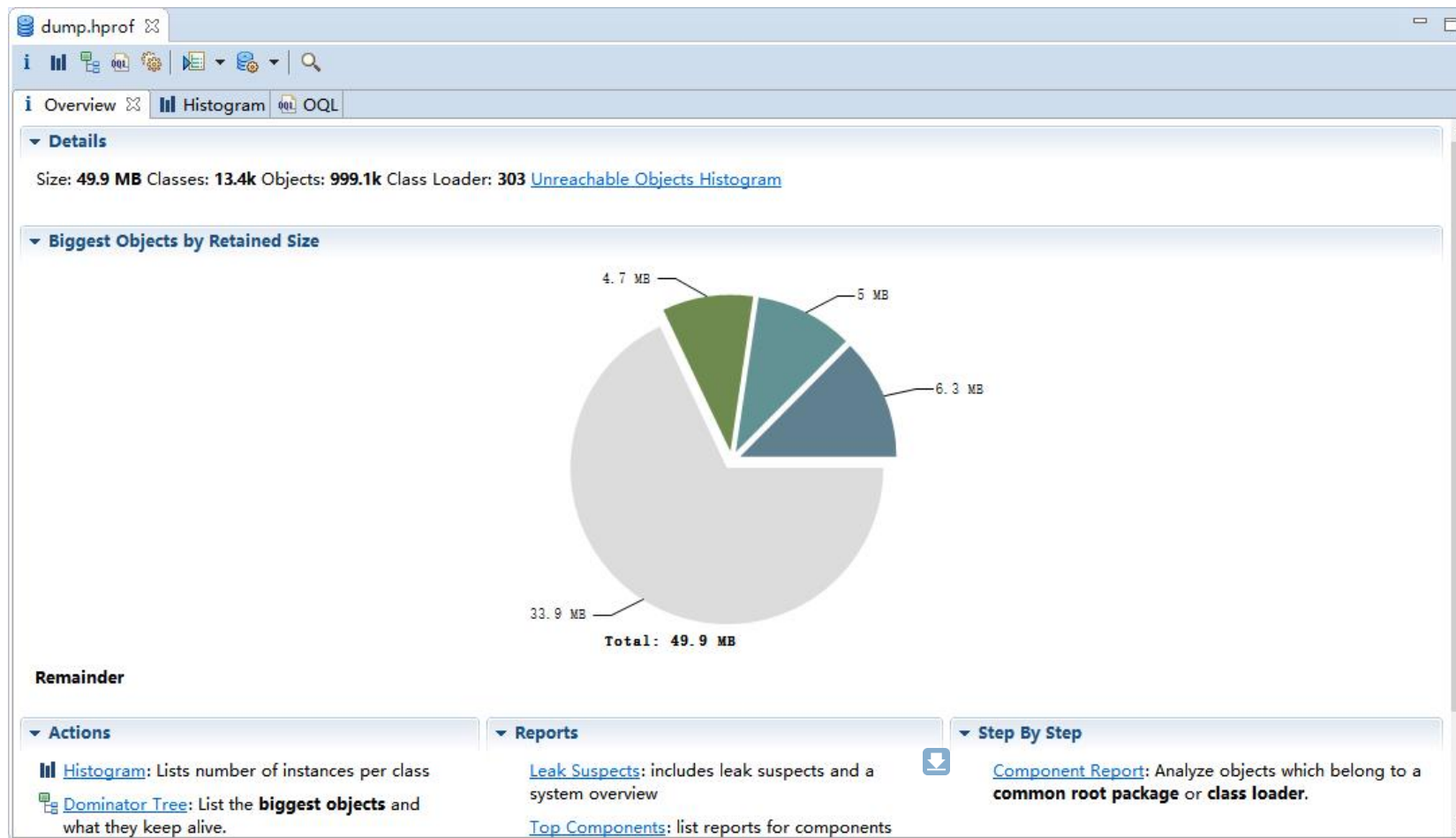
MAT

MAT是什么?

MAT(Memory Analyzer Tool), 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。

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以eclipse 插件的形式来安装, 具体的安装过程就不在描述了, 可以利用visualvm或者是 jmap命令生产堆文件, 导入eclipse mat中生成分析报告:



生产这汇报表的同时也会在dump文件的同级目录下生成三份（dump_Top_Consumers.zip、dump_Leak_Suspects.zip、dump_Top_Components.zip）分析结果的html文件，方便发送给相关同事来查看。

需要关注的是下面的Actions、Reports、Step by Step区域：

- Histogram：列出内存中的对象，对象的个数以及大小，支持正则表达式查找，也可以计算出该类所有对象的retained size

Class Name	Objects	Shallow Heap	Retained H...
<Regex>	<Numeric>	<Numeric>	<Numeric>
char[]	141,322	17,874,056	
byte[]	5,469	3,968,496	
java.lang.String	131,811	3,163,464	
java.util.HashMap\$Node	97,300	3,113,600	
java.lang.reflect.Method	30,327	2,668,776	
java.util.concurrent.ConcurrentH...	57,195	1,830,240	
java.util.HashMap\$Node[]	10,292	1,361,536	
java.lang.Object[]	16,371	876,208	
java.util.LinkedHashMap\$Entry	19,534	781,360	
java.util.HashMap	11,586	556,128	
java.lang.Integer	33,760	540,160	
java.util.concurrent.ConcurrentH...	531	502,896	
java.lang.String[]	10,256	494,128	
org.springframework.core.Met...	19,184	460,416	
int[]	4,881	455,544	
java.util.LinkedHashMap	8,124	454,944	
org.codehaus.groovy.runtime....	7,679	430,024	
java.lang.Class[]	19,300	419,912	
com.sun.org.apache.xerces.inte...	12,846	411,072	
java.lang.Object	19,100	305,600	
java.util.Hashtable\$Entry	9,267	296,544	
org.hibernate.hql.internal.ast.tr...	5,968	286,464	
java.util.LinkedList	8,237	263,584	

- Dominator Tree: 列出最大的对象以及其依赖存活的Object (大小是以Retained Heap为标准排序的)

Inspector Window (Left):

- Address: @ 0x3d3473de0
- Class: ParallelWebappClassLoader
- Package: org.apache.catalina.loader
- Class Name: class org.apache.catalina.loader.ParallelWebappClassLoader @...
- Superclass: org.apache.catalina.loader.WebappClassLoaderBase
- Address: java.net.URLClassLoader @ 0x3d3474000
- Size: 136 (shallow size)
- Size: 6,568,144 (retained size)
- GC Root: no GC root

dominator_tree Table (Right):

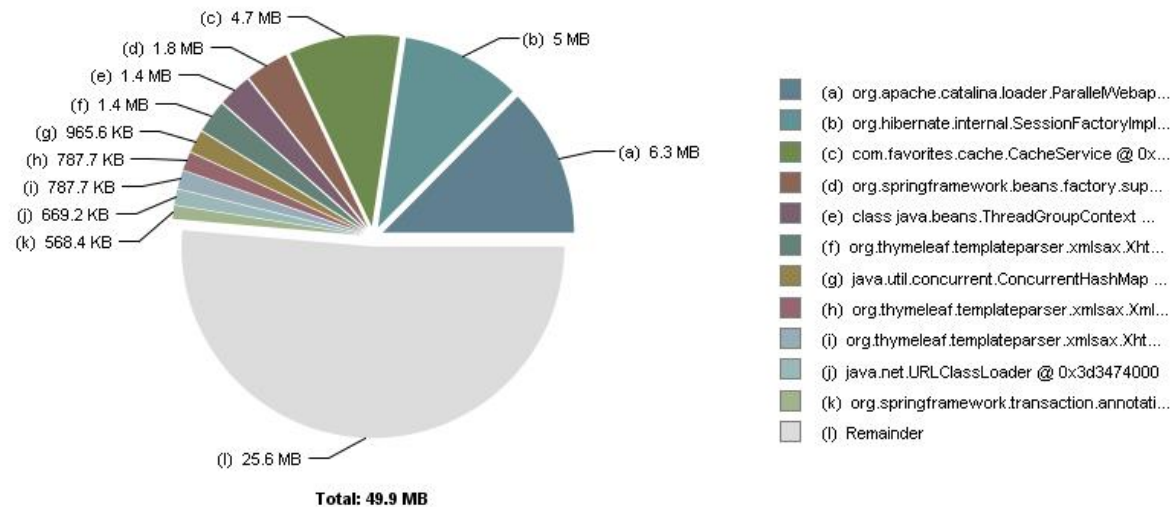
Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
org.apache.catalina.loader.ParallelWebappClassLoader @ 0x3d3473de0	136	6,568,144	12.55%
java.util.concurrent.ConcurrentHashMap @ 0x3d36f3708	64	2,191,400	4.19%
java.util.concurrent.ConcurrentHashMap @ 0x3d38646d0	64	2,011,272	3.84%
class org.springframework.beans.CachedIntrospectionResults @ 0x3d35a6f0	32	350,120	0.67%
class org.springframework.core.annotation.AnnotationUtils @ 0x3d38ede1c	40	198,552	0.38%
class org.unbescape.html.HtmlEscapeSymbols @ 0x3d6140b48	16	169,704	0.32%
java.util.HashMap @ 0x3d37fc9b8	48	169,328	0.32%
class org.jsoup.nodes.Entities @ 0x3d5effa80	24	130,768	0.25%
java.util.concurrent.ConcurrentHashMap @ 0x3d385e2a8	64	123,896	0.24%
class org.springframework.cglib.core.AbstractClassGenerator @ 0x3d41437	8	116,216	0.22%
class org.springframework.core.SerializableTypeWrapper @ 0x3d3729d88	8	87,984	0.17%
java.util.HashMap @ 0x3d5effb80	48	54,528	0.10%
class com.mysql.jdbc.CharsetMapping @ 0x3d4f171e0	48	51,216	0.10%
class org.springframework.util.ReflectionUtils @ 0x3d38dce68	32	50,192	0.10%
class org.aspectj.bridge.IMessageHandler @ 0x3d41ebb30	16	49,856	0.10%
java.util.Vector @ 0x3d3800660	32	41,008	0.08%
java.util.ArrayList @ 0x3d374aa08	24	35,664	0.07%
class ognl.OgnlRuntime @ 0x3d6297be0	200	34,696	0.07%
class org.aspectj.apache.bcel.Constants @ 0x3d4920e18	824	27,584	0.05%
class com.mysql.jdbc.Constants @ 0x3d4658c10	56	24,600	0.05%
class org.aspectj.lang.reflect.AjTypeSystem @ 0x3d41eda40	8	22,256	0.04%
class org.springframework.core.convert.support.DefaultConversionService @ 0x3d431ab00	8	22,032	0.04%
java.util.HashMap @ 0x3d37af078	48	17,648	0.03%
class org.springframework.core.GenericTypeResolver @ 0x3d35a63a0	8	15,776	0.03%

- Top Consumers : 通过图形列出最大的object

Top Consumers

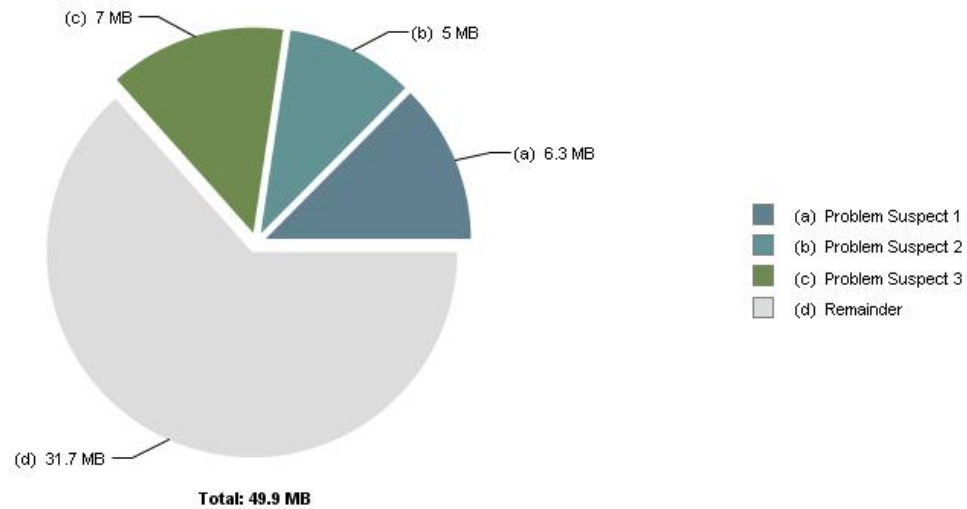
Top Consumers

▼ Biggest Objects (Overview)



Biggest Objects

- duplicate classes : 检测由多个类装载器加载的类
- Leak Suspects : 内存泄漏分析



▼ Problem Suspect 1

The classloader/component **"org.apache.catalina.loader.ParallelWebappClassLoader @ 0x3d3473de0"** occupies **6,568,144 (12.55%)** bytes. The memory is accumulated in classloader/component **"org.apache.catalina.loader.ParallelWebappClassLoader @ 0x3d3473de0"**.

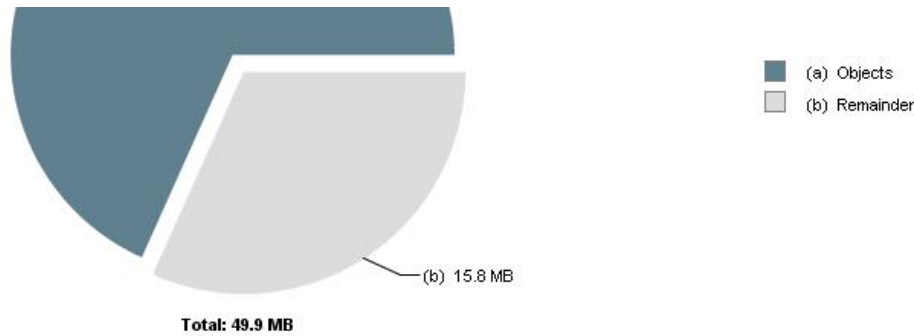
Keywords

org.apache.catalina.loader.ParallelWebappClassLoader @ 0x3d3473de0

[Details »](#)



- Top Components: 列出大于总堆数的百分之1的报表。



Top Consumers

Retained Set

▼ Possible Memory Waste

▼ Duplicate Strings

Found 539 occurrences of char[] with at least 10 instances having identical content. Total size is 1,636,840 bytes.

Top elements include:

- [illegible]

- Component Report:分析对象属于同一个包或者被同一个类加载器加载

以上只是一个初级的介绍，mat还有更强大的使用，比如对比堆内存，在生产环境中往往为了定位问题，每隔几分钟dump出一下内存快照，随后在对比不同时间的堆内存的变化来发现问题。

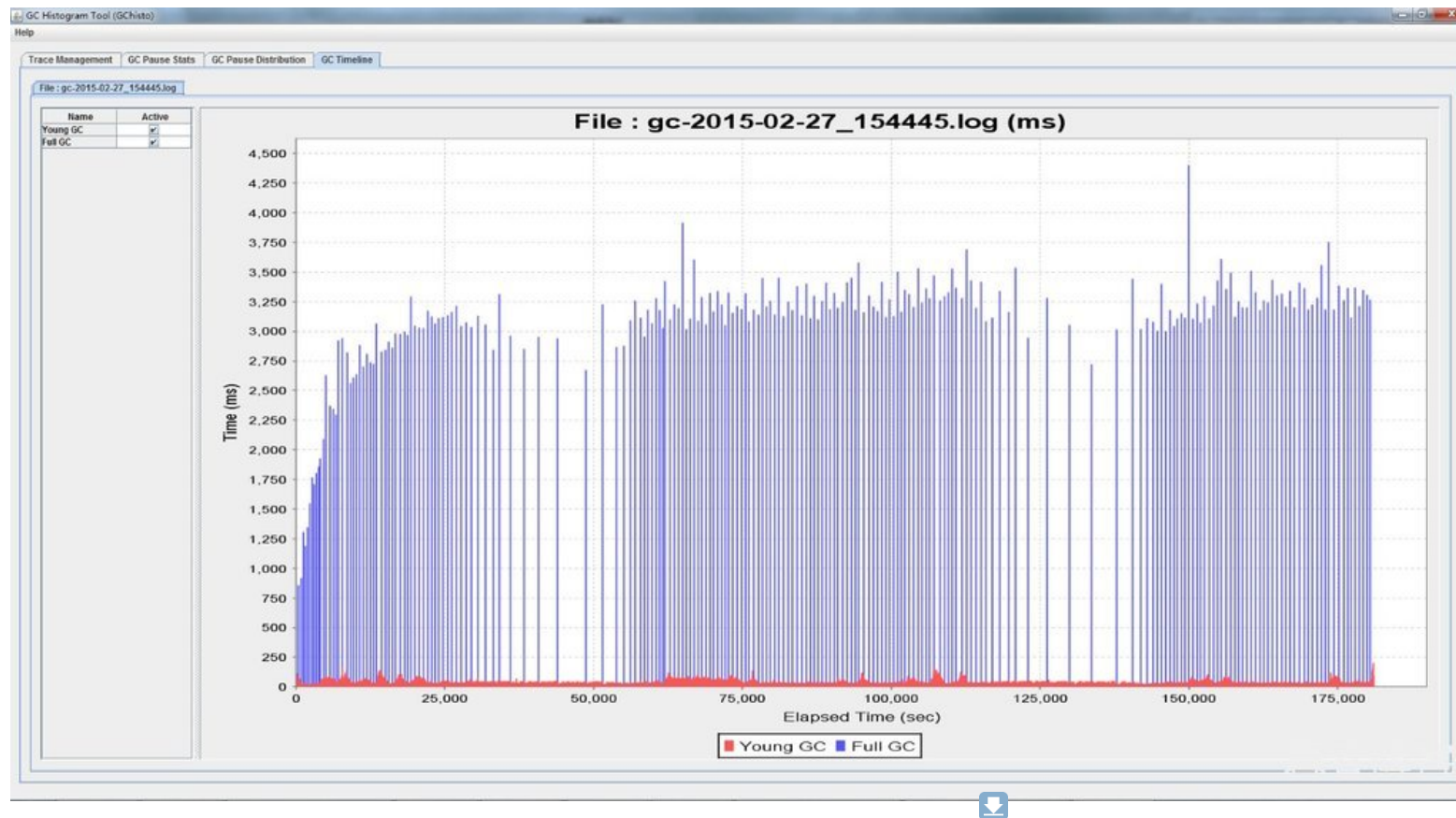
GChisto

GCChisto是一款专业分析gc日志的工具，可以通过gc日志来分析：Minor GC、full gc的时间、频率等等，通过列表、报表、图表等不同的形式来反应gc的情况。虽然界面略显粗糙，但是功能还是不错的。

配置好本地的jdk环境之后，双击GChisto.jar,在弹出的输入框中点击 add 选择gc.log日志

- GC Pause Stats:可以查看GC 的次数、GC的时间、GC的开销、最大GC时间和最小GC时间等，以及相应的柱状图

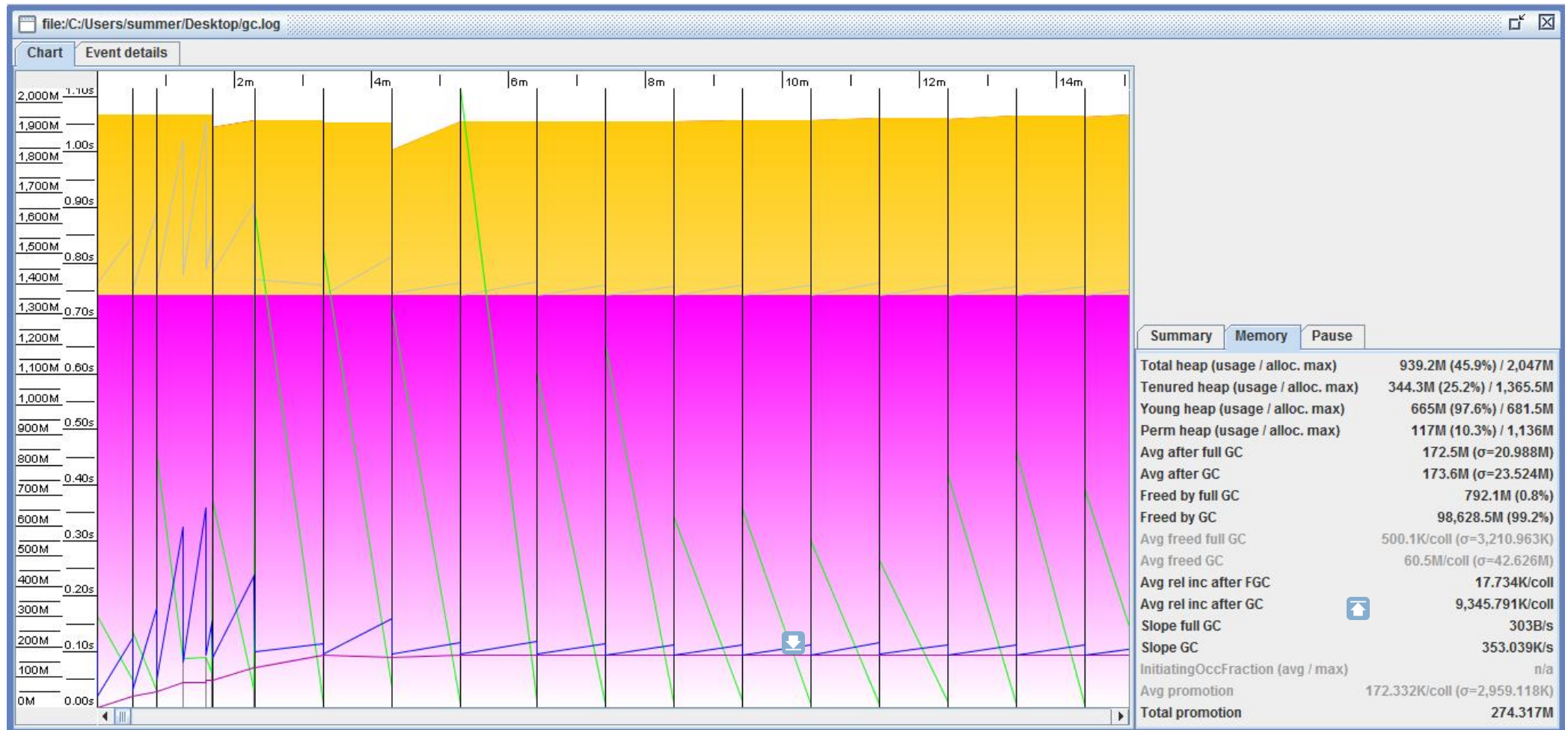
- GC Pause Distribution:查看GC停顿的详细分布，x轴表示垃圾收集停顿时间，y轴表示是停顿次数。
- GC Timeline：显示整个时间线上的垃圾收集



不过这款工具已经不再维护，不能识别最新jdk的日志文件。

gcviewer

GCViewer也是一款分析小工具，用于可视化查看由Sun / Oracle, IBM, HP 和 BEA Java 虚拟机产生的垃圾收集器的日志，gcviewer个人感觉显示的界面比较乱没有GChisto更专业一些。



以上的两款gc分析日志，一个不太维护了，一个不太专业，求推荐更好的gc分析工具。

本系列：

- [JVM \(1\) : Java 类的加载机制](#)
- [JVM \(2\) : JVM内存结构](#)
- [JVM \(3\) : Java GC算法 垃圾收集器](#)
- [JVM \(4\) : Jvm调优-命令篇](#)
- [JVM \(5\) : tomcat性能调优和性能监控 \(visualvm\)](#)
- [JVM \(6\) : JVM调优-从eclipse开始](#)
- [JVM \(7\) : JVM调优-工具篇](#)



相关文章

- [Java虚拟机 \(JVM\) 概述](#)
- [从JVM heap dump里查找没有关闭文件的引用](#)
- [使用 JITWatch 查看 JVM 的 JIT 编译代码](#)
- [JVM堆内存使用率持续上升的一种排查思路](#)
- [直播一次问题排查过程](#)
- [Java 虚拟机16: Metaspace](#)
- [Java 虚拟机 13: 互斥同步、锁优化及synchronized和volatile](#)
- [Java 虚拟机 12 : Java 内存模型](#)
- [Java 虚拟机 11 : 运行期优化](#)
- [Java 虚拟机10: 类加载器](#)

发表评论

Comment form

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*



请填写评论内容

(*) 表示必填项

[提交评论](#)

2 条评论

1. 張san 说道:

[2017/08/23 下午 5:40](#)

你好, 请教一下, jvm这些命令或者工具, 你们平时生产环境都用来做些什么呢? 能否举一两个具体场景, 多谢!
例如线上发生了什么现象, 通过jvm工具发现了什么问题, 最后解决了什么潜在bug~
因为我们所处环境根本不会用到这些东西, 所以感觉学习起来还是有些抽象~

 0  0

[回复](#)

◦ 唐小娟 说道:

[2017/08/27 下午 1:11](#)

举个例子 譬如发现程序在一段时间内停止运行 过了一段时间又恢复了 可能就是系统在gc 至于如何优化JVM的参数 减少系统这种停止运行的时间 这些工具可以帮助你

 1  0

[回复](#)

[« JVM \(6\) : JVM调优-从eclipse开始](#)

[JVM \(8\) : JVM知识点总览-高级Java工程师面试必备 »](#)

Search for:

Search

SpringCloud

Git

Nginx

Redis

ActiveMQ

Hadoop

FastDFS

MyCat

Velocity

SpringBoot



更多前沿技术

- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

- 0 [内存屏障和 volatile 语义](#)
- 1 [SpringBoot | 第十七章：web ...](#)
- 2 [SpringBoot | 第十八章：web 应用开...](#)
- 3 [Java 线程池详解](#)
- 4 [JDK 源码阅读：DirectByteBu...](#)
- 5 [Map 大家族的那点事儿（5）：We...](#)
- 6 [Map 大家族的那点事儿（6）：Lin...](#)
- 7 [Map 大家族的那点事儿（7）：Concu...](#)
- 8 [如果非得了解下 git 系统....](#)
- 9 [SpringBoot | 第十九章：web 应用开发...](#)



最新评论

-  Re: [内存屏障和 volatile 语义](#)
会思考的作者 小宇宙
- 

Re: [SpringBoot | 第十五章：基于Pos...](#)
一直用postman www.wuliaokankan.cn

Re: [探究 Java 虚拟机栈](#)
不错 aa

Re: [Java并发编程：CountDownLatch、CyclicB...](#)
> \"release()用来释放许可。注意，在释放许可之前，必须先获获得许可。\"Semapho... 苍穆

Re: [HashMap的工作原理](#)
那为什么不使用HashMap也要说清楚呀，要不然稀里糊涂的 渔夫

Re: [并发编程 – Concurr...](#)
总结的很细致，感谢作者！ 落雨无声

Re: [做一次面向对象的体操：将JSO...](#)
大侠， TransferUtil 和 Order 类没有，能否贴出来，学习学习。谢谢。 sailor

Re: [Map大家族的那点事儿\(1\)：M...](#)
可以的 李红波



关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻：)

ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。import 可认为是学习和吸收，new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email: ImportNew.com@gmail.com

新浪微博: [@ImportNew](#)

推荐微信号



反馈建议: ImportNew.com@gmail.com

广告与商务合作QQ: 2302462408

推荐关注

[小组](#) – 好的话题、有启发的回复、值得信赖的圈子

[头条](#) – 写了文章? 看干货? 去头条!

[相亲](#) – 为IT单身男女服务的征婚传播平台

[资源](#) – 优秀的工具资源导航

[翻译](#) – 活跃 & 专业的翻译小组

[博客](#) – 国内外的精选博客文章

[设计](#) – UI,网页, 交互和用户体验

[前端](#) – JavaScript, HTML5, CSS

[安卓](#) – 专注Android技术分享

[iOS](#) – 专注iOS技术分享

[Java](#) – 专注Java技术分享

[Python](#) – 专注Python技术分享

© 2018 ImportNew

