Tomcat性能监控

##### 工具：java visualVM

##### 环境部署：Apache Tomcat 6.0、jdk1.6.0\_43、智慧校园三大平台

##### 1、简介

VisualVM 是一款免费的性能分析工具。它通过 jvmstat、JMX、SA（Serviceability Agent）以及 Attach API 等多种方式从程序运行时获得实时数据，从而进行动态的性能分析。同时，它能自动选择更快更轻量级的技术尽量减少性能分析对应用程序造成的影响，提高性能分析的精度。不足之处是VisualVM只支持jdk6.0以上的版本

##### 2、性能分析的主要方式

性能分析：性能分析是通过收集程序运行时的执行数据来帮助开发人员定位程序需要被优化的部分，从而提高程序的运行速度或是内存使用效率，主要有以下三个方面：

CPU 性能分析：CPU 性能分析的主要目的是统计函数的调用情况及执行时间，或者更简单的情况就是统计应用程序的 CPU 使用情况。通常有 CPU 监视和 CPU 快照两种方式来显示 CPU 性能分析结果。

内存性能分析：内存性能分析的主要目的是通过统计内存使用情况检测可能存在的内存泄露问题及确定优化内存使用的方向。通常有内存监视和内存快照两种方式来显示内存性能分析结果。

线程性能分析：线程性能分析主要用于在多线程应用程序中确定内存的问题所在。一般包括线程的状态变化情况，死锁情况和某个线程在线程生命期内状态的分布情况等

##### 3、安装插件

[VisualVM 插件中心](http://visualvm.java.net/pluginscenters.html)提供很多插件以供安装。

从 VisualVM 插件中心安装插件安装步骤 :

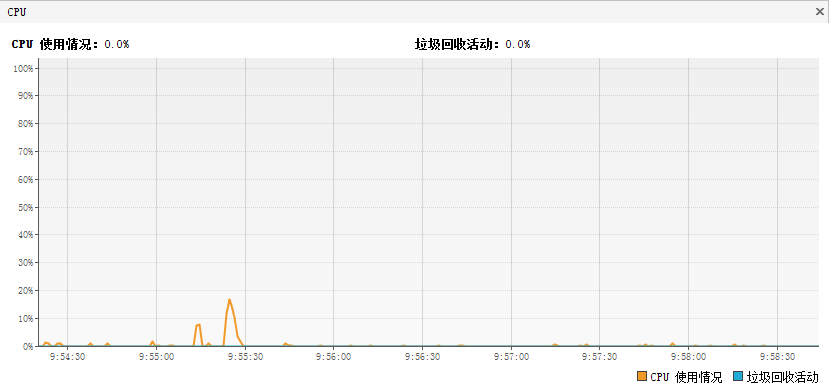
* 从主菜单中选择“工具”>“插件”。
* 在“可用插件”标签中，选中该插件的“安装”复选框。单击“安装”。
* 逐步完成插件安装程序。

##### 4、功能介绍

CPU 分析

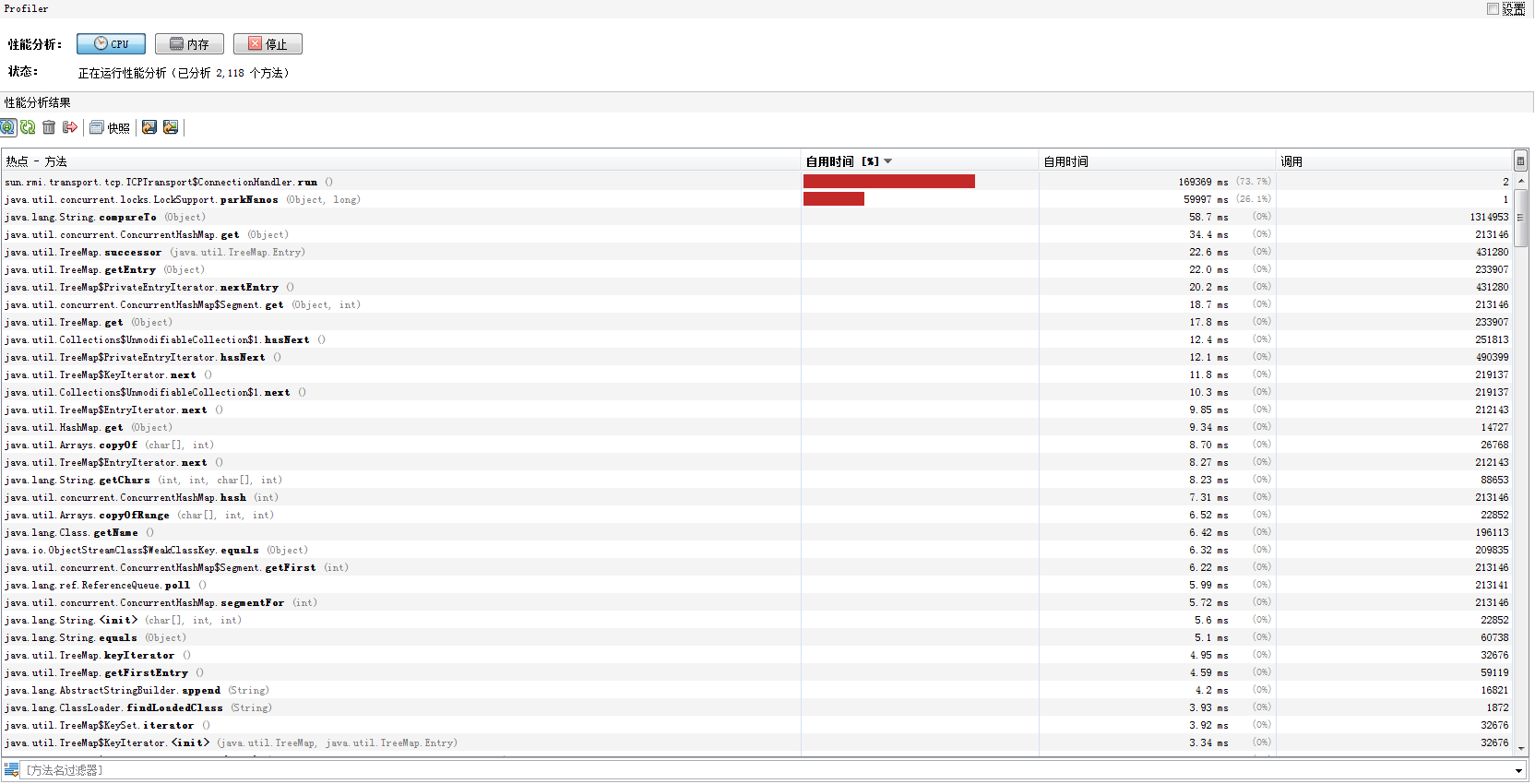
VisualVM 能够监控应用程序在一段时间的 CPU 的使用情况，显示 CPU 的使用率、方法的执行效率和频率等相关数据帮助我们发现应用程序的性能瓶颈。我们可以通过 VisualVM 的监视标签和 Profiler 标签对应用程序进行 CPU 性能分析。

在监视标签内，我们可以查看 CPU 的使用率以及垃圾回收活动对性能的影响。过高的 CPU 使用率可能是由于我们的项目中存在低效的代码，可以通过 Profiler 标签的 CPU 性能分析功能进行详细的分析。如果垃圾回收活动过于频繁，占用了较高的 CPU 资源，可能是由内存不足或者是新生代和旧生代分配不合理导致的等。



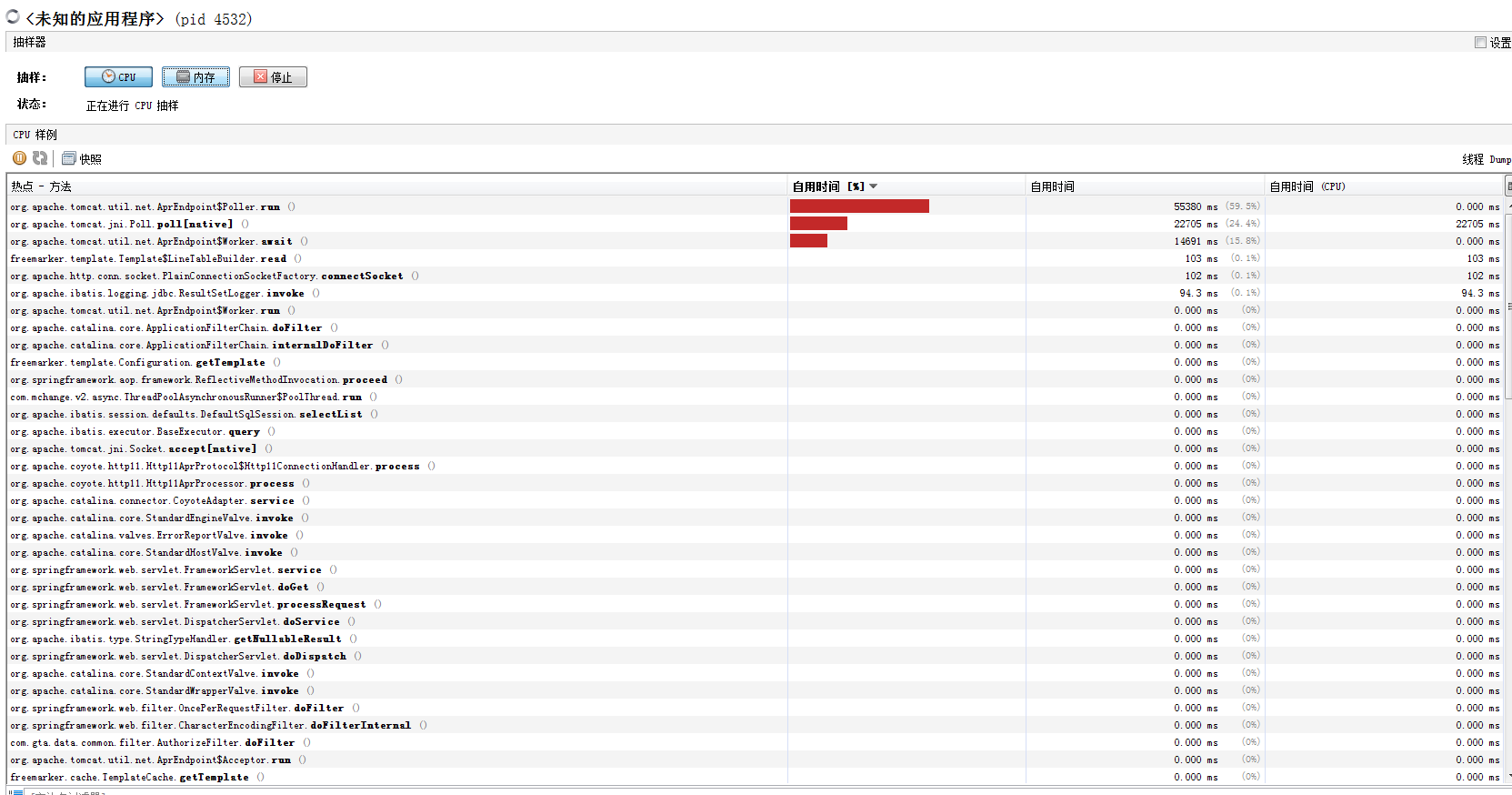
在 Profiler 标签，点击“CPU”按钮启动一个 CPU 性能分析会话 ,VisualVM 会检测应用程序所有的被调用的方法。当进入一个方法时，线程会发出一个“method entry”的事件，当退出方法时同样会发出一个“method exit”的事件，这些事件都包含了时间戳。然后 VisualVM 会把每个被调用方法的总的执行时间和调用的次数按照运行时长展示出来。

此外，我们也可以通过性能分析结果下方的方法名过滤器对分析结果进行过滤。



点击取样器Sampler， 点击“CPU”按钮， 启动CPU性能分析会话，VisualVM 会检测应用程序所有的被调用的方法，

在CPU samples tab 下可以看到我们的方法AprEndpoint() 的自用时间最长， 如下图：



内存分析

VisualVM 通过检测 JVM 中加载的类和对象信息等帮助我们分析内存使用情况，我们可以通过 VisualVM 的监视标签和 Profiler 标签对应用程序进行内存分析。

在监视标签内，我们可以看到实时的应用程序内存堆以及永久保留区域的使用情况。

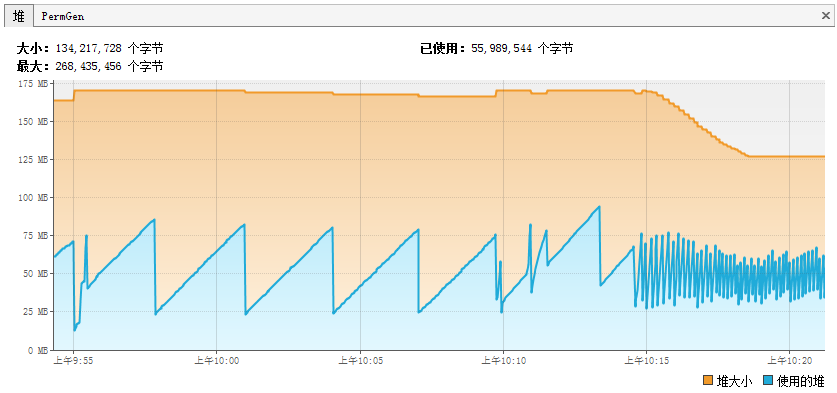


图1 内存堆使用情况

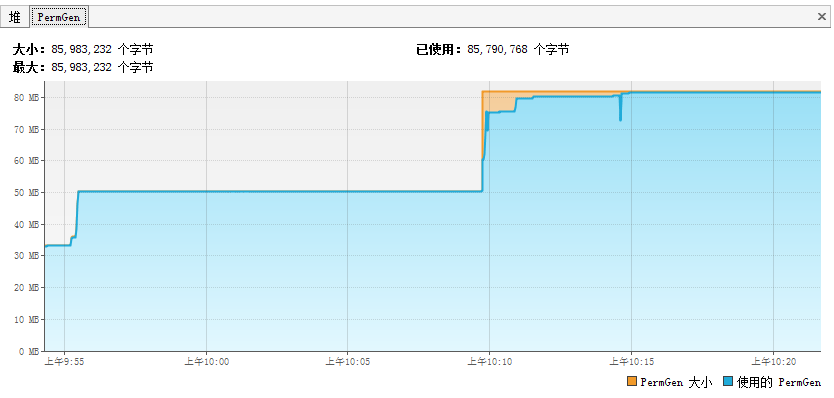
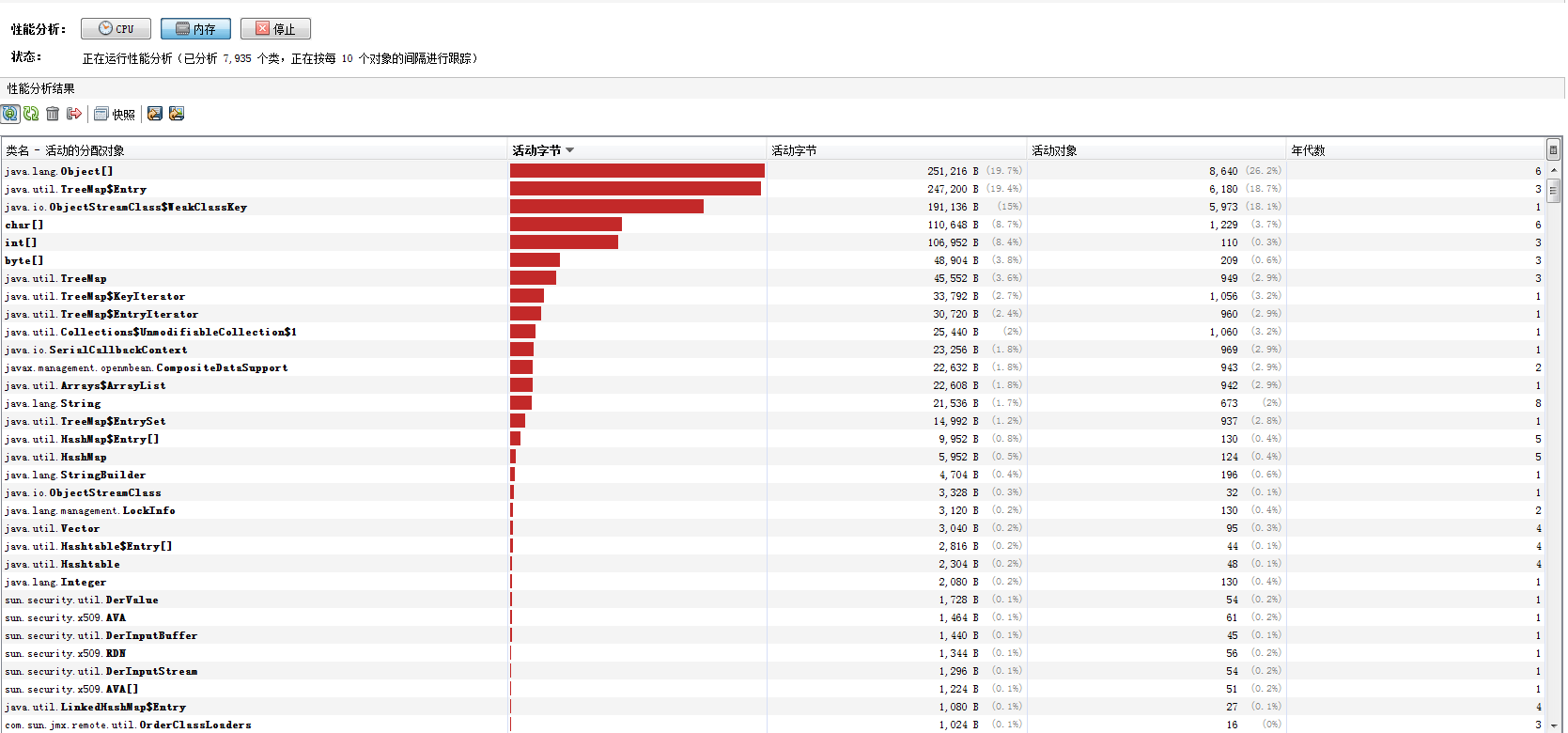


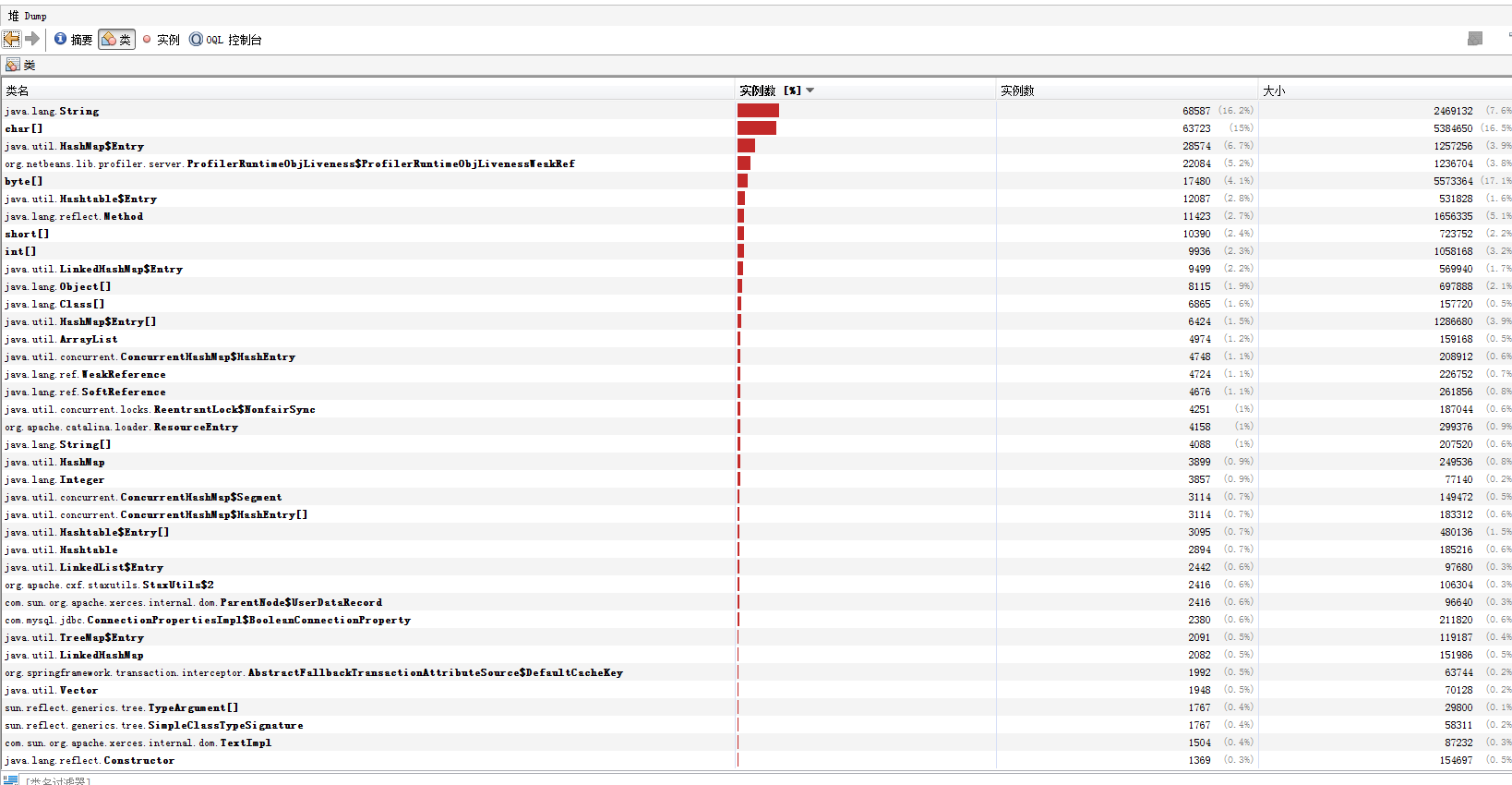
图2 永久保留区域使用情况

在 Profiler 标签，点击“内存”按钮将启动一个内存分析会话，等 VisualVM 收集和统计完相关性能数据信息，将会显示在性能分析结果。通过内存性能分析结果，我们可以查看哪些对象占用了较多的内存，存活的时间比较长等，以便做进一步的优化。

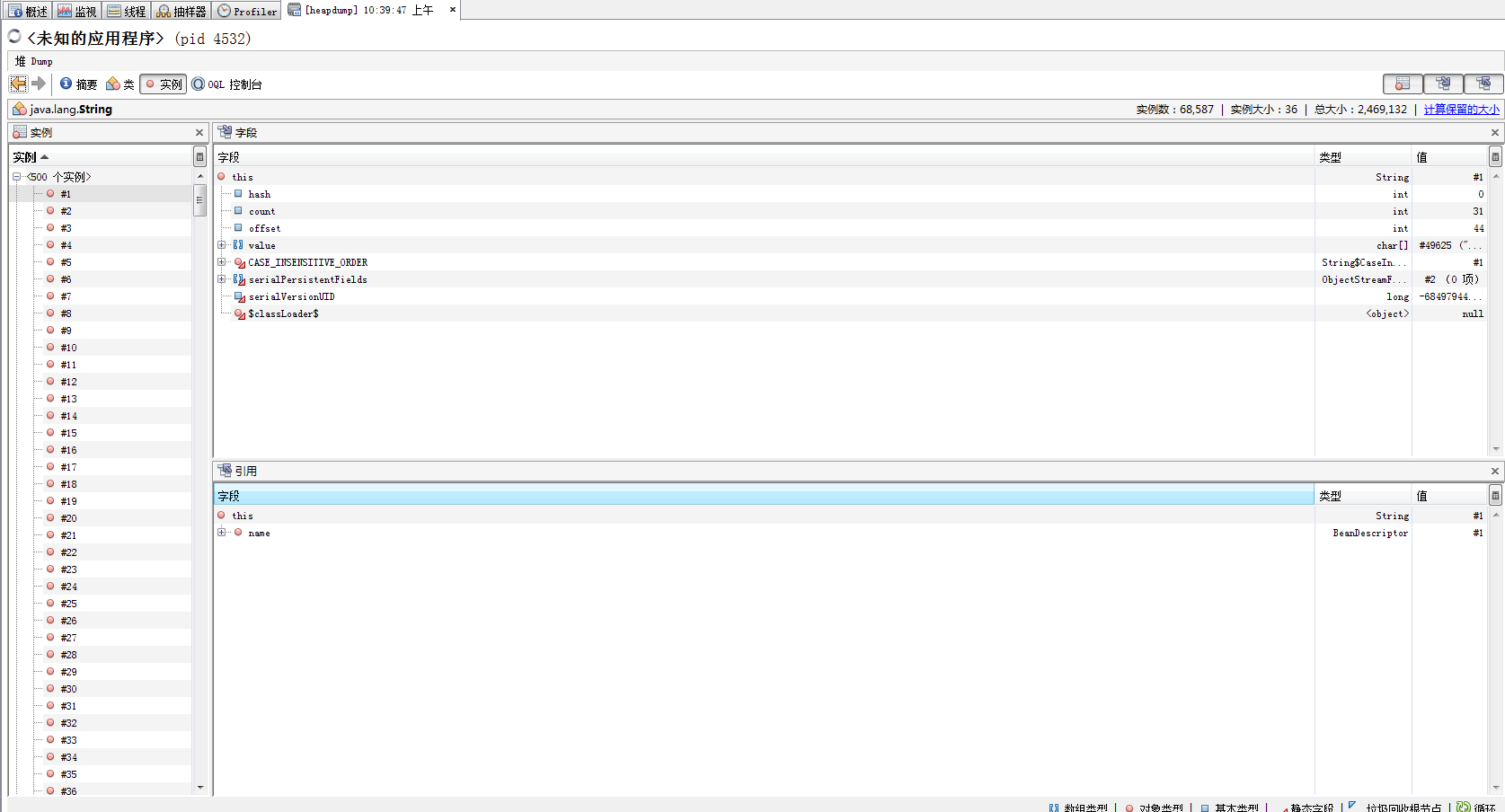
此外，我们可以通过性能分析结果下方的类名过滤器对分析结果进行过滤。



点击堆dump，发现java.lang.string所占用的内存是最大的



双击它，得到如下实例结果：



实例是按Size由大到小排列的，第一个就是最大的， 展开Field区域的 values

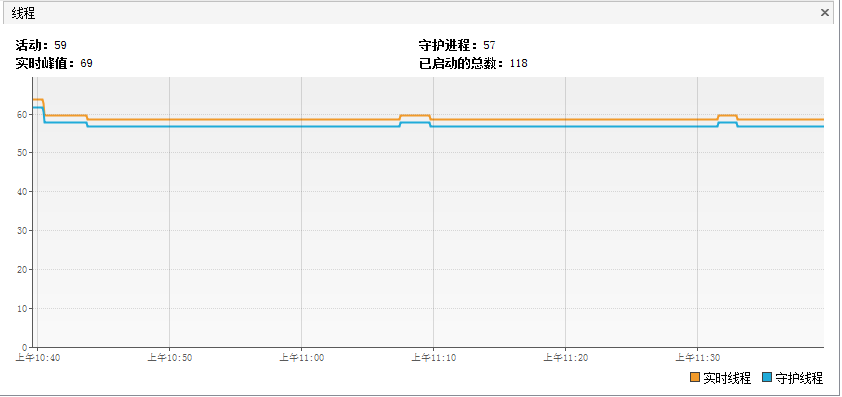
得到结果：char[]类型的 全局变量占用内存特别大， 注意局部变量是无法通过 堆dump来得到分析结果的。

另外，对于“堆 dump”来说，在远程监控jvm的时候，VisualVM是没有这个功能的，只有本地监控的时候才有。

线程分析

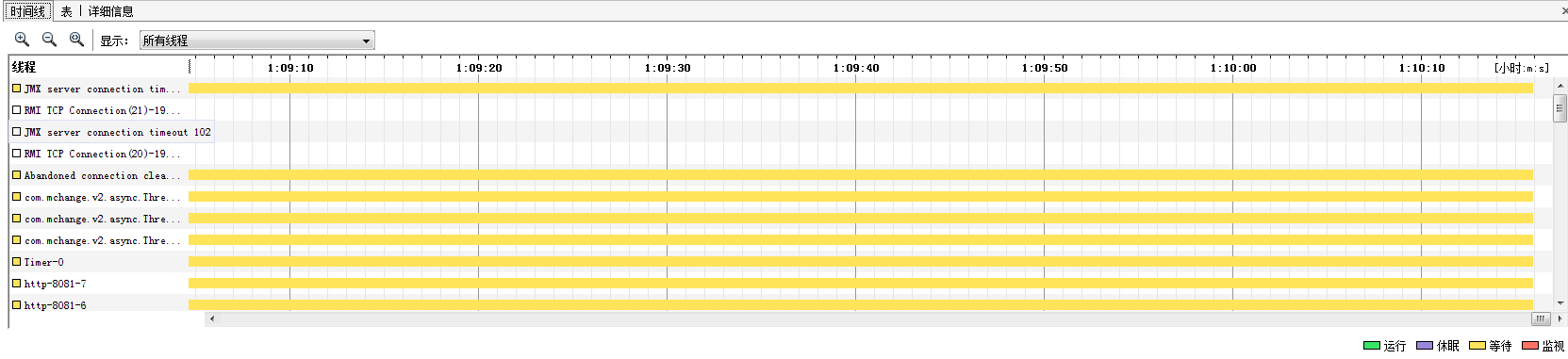
Java 语言能够很好的实现多线程应用程序。当我们对一个多线程应用程序进行调试或者开发后期做性能调优的时候，往往需要了解当前程序中所有线程的运行状态，是否有死锁、热锁等情况的发生，从而分析系统可能存在的问题。

在 VisualVM 的监视标签内，我们可以查看当前应用程序中所有活动线程和守护线程的数量等实时信息。

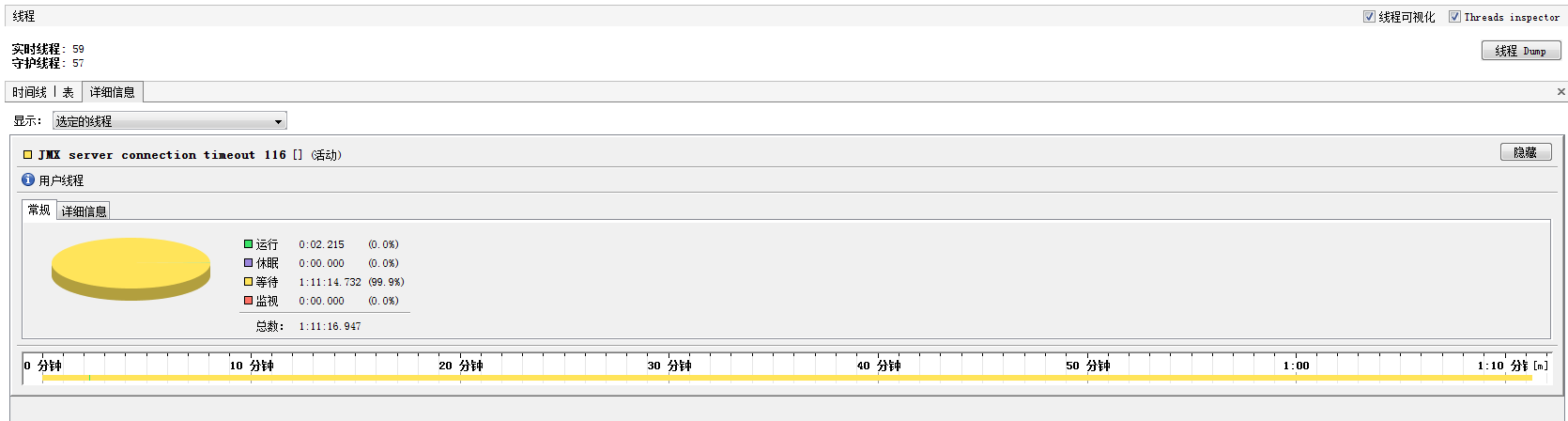


VisualVM 的线程标签提供了三种视图，默认会以时间线的方式展现。另外两种视图分别是表视图和详细信息视图。

时间线视图上方的工具栏提供了缩小，放大和自适应三个按钮，以及一个下拉框，我们可以选择将所有线程、活动线程或者完成的线程显示在视图中。



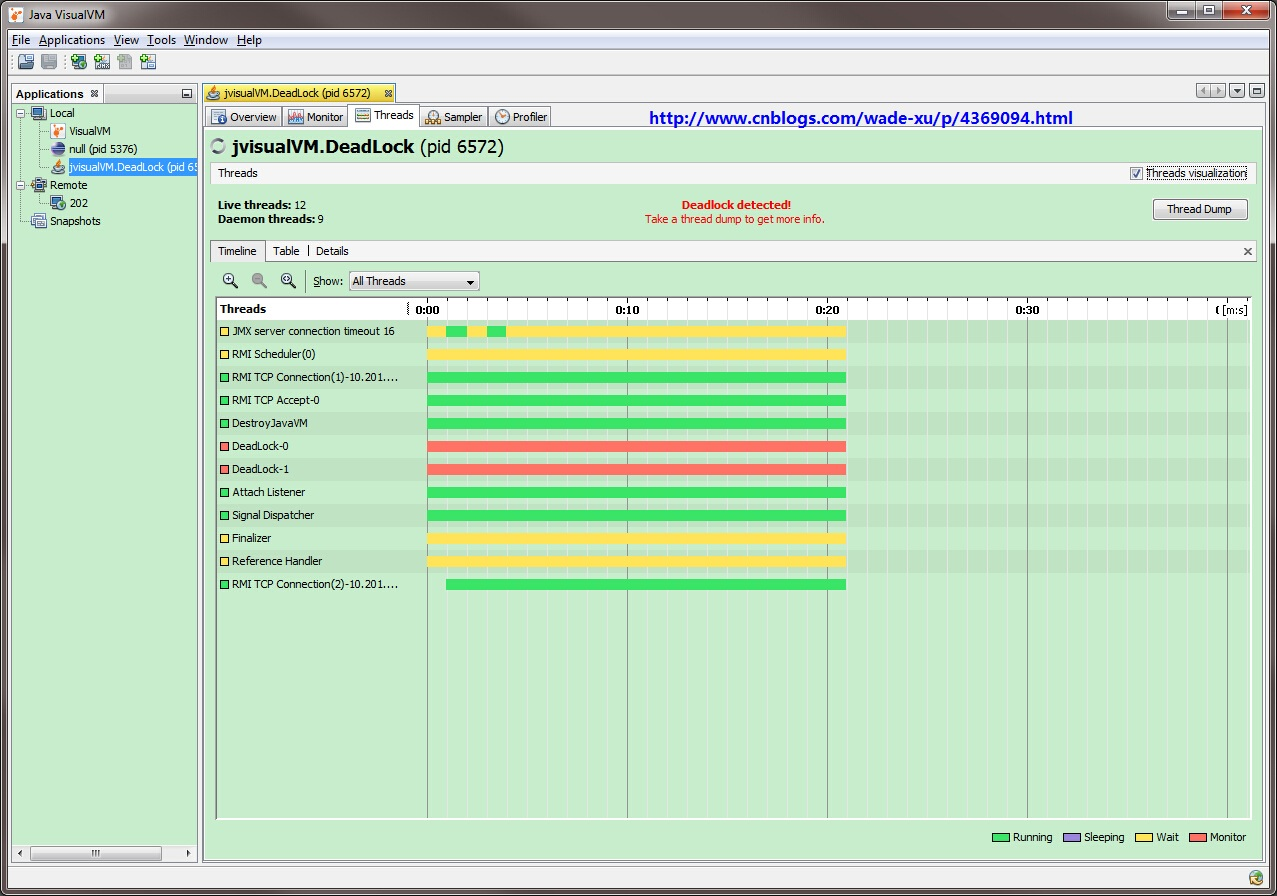




打开VisualVM检测到的JVM进程，我们可以看到这个tab在闪，VisualVM已经检测到我这个package下面的DeadLock类出错了

切换到Thread tab， 可以看到死锁了， Deadlock detected!

另外可以点击Thread Dump 线程转储，进一步分析，在这里就不赘述了，有兴趣的读者可以自行实验。



##### 5、Tomcat性能调优

tomcat内存优化

根据服务器物理内容情况配置相关参数优化tomcat性能。当应用程序需要的内存超出堆的最大值时虚拟机就会提示内存溢出，并且导致应用服务崩溃。因此一般建议堆的最大值设置为可用内存的最大值的80%。 Tomcat默认可以使用的内存为128MB，在较大型的应用项目中，这点内存是不够的，需要调大.

内存溢出最常见的情况：

　　1.OutOfMemoryError: Java heap space

　　2.OutOfMemoryError: PermGen space

Tomcat 默认可以使用的内存为128MB,Windows下,在文件/bin/catalina.bat，Unix下，在文件/bin/catalina.sh的前面，增加如下设置： JAVA\_OPTS='-Xms【初始化内存大小】 -Xmx【可以使用的最大内存】 -XX:PermSize=64M -XX:MaxPermSize=128m' 需要把几个参数值调大。例如： JAVA\_OPTS='-Xms256m -Xmx512m' 表示初始化内存为256MB，可以使用的最大内存为512MB。

参数详解

-server 启用jdk 的 server 版；

-Xms java虚拟机初始化时的最小内存；

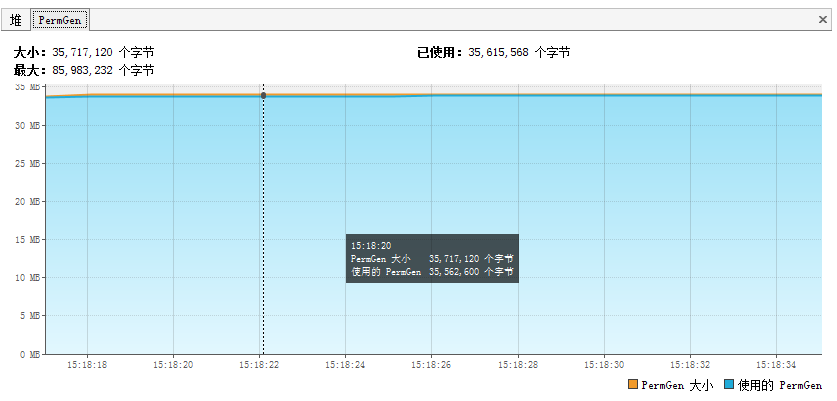
-Xmx java虚拟机可使用的最大内存；

-XX:PermSize 内存永久保留区域

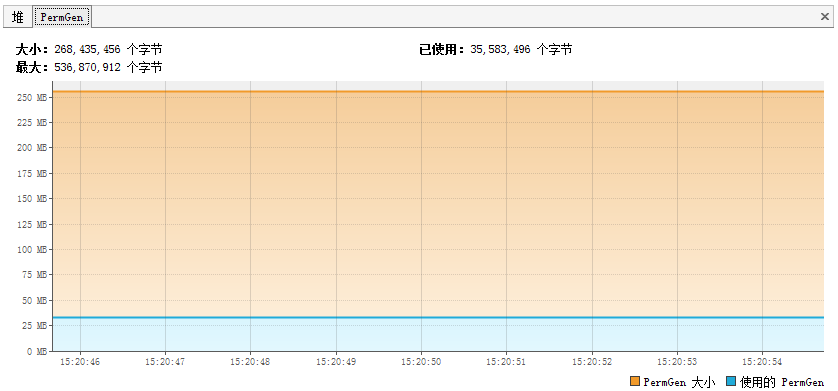
-XX:MaxPermSize 内存最大永久保留区域

-Xmn jvm最小内存

未设置jdk内存前：



进入tomcat,打开catalina.bat，加入一行：SET JAVA\_OPTS=-Xms1024m -Xmx1024m -XX:PermSize=256m -XX:MaxPermSize=512m，重启tomcat及visualVM,查看结果：



**Tomcat并发优化**

在tomcat配置文件server.xml中的配置中，和连接数相关的参数有：

maxThreads： Tomcat使用线程来处理接收的每个请求。这个值表示Tomcat可创建的最大的线程数。默认值150。

acceptCount： 指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理。默认值10。

minSpareThreads： Tomcat初始化时创建的线程数。默认值25。

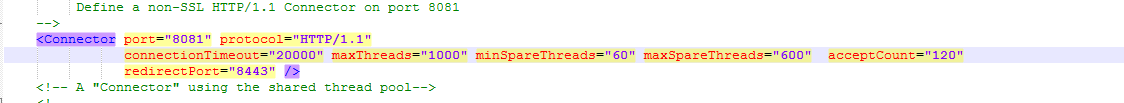
maxSpareThreads： 一旦创建的线程超过这个值，Tomcat就会关闭不再需要的socket线程。默认值75。

enableLookups： 是否反查域名，默认值为true。为了提高处理能力，应设置为false

connnectionTimeout： 网络连接超时，默认值60000，单位：毫秒。设置为0表示永不超时，这样设置有隐患的。通常可设置为30000毫秒。

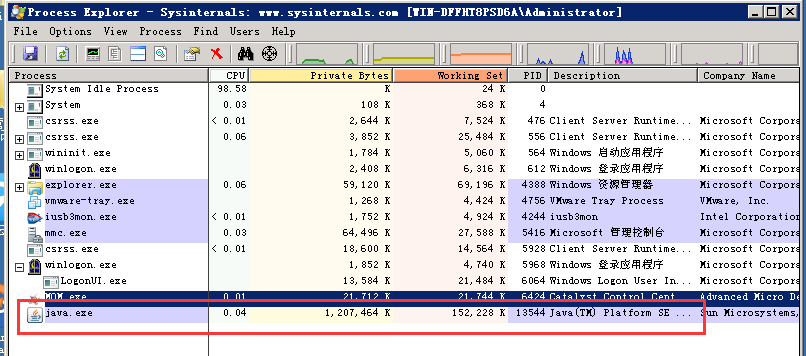
maxKeepAliveRequests： 保持请求数量，默认值100。 bufferSize： 输入流缓冲大小，默认值2048 bytes。

compression： 压缩传输，取值on/off/force，默认值off。 其中和最大连接数相关的参数为maxThreads和acceptCount。如果要加大并发连接数，应同时加大这两个参数。

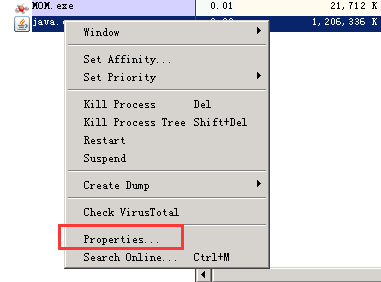


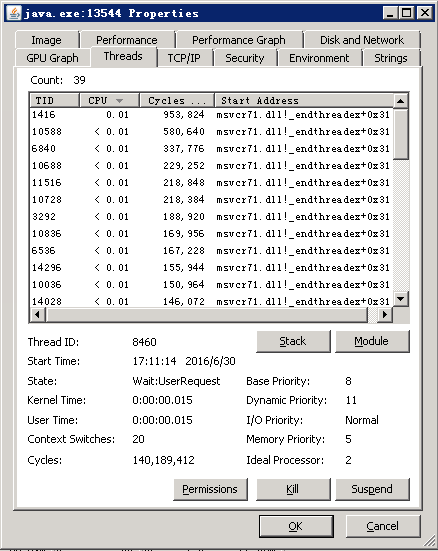
##### 工具：Process Explorer

1. 首先打开Process Explorer工具，确认好产品的运行进程ID（例如java.exe） **13544**



1. 选中java.exe右键打开Properties， 切换到Threads选项卡，查看对应的线程运行情况

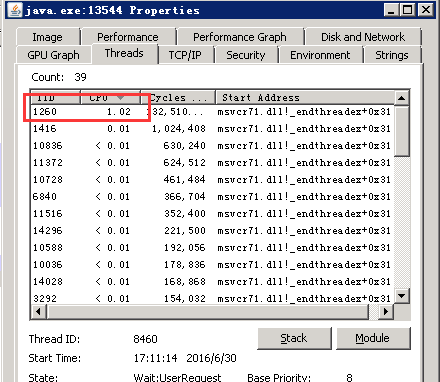




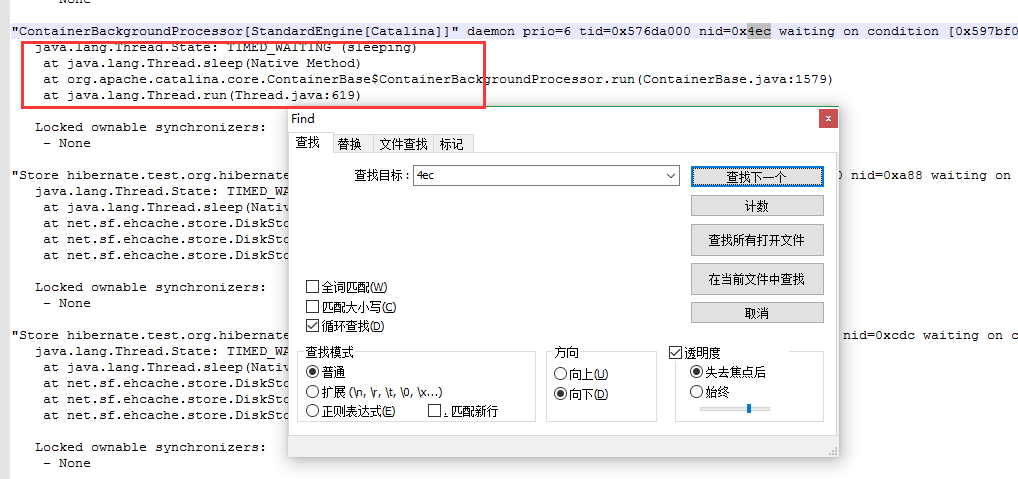
1. 利用java自带的jstack.exe命令导出进程运行的堆栈信息 （提示：这里可以根据CPU占用较高时多导出几次文件对比分析）

C:\Users\Administrator>jstack –l 进程ID > C:/13544.jstack

1. 例如1260线程占用CPU过高



1. 把1260转换成十六进制 （4ec）到刚才导出的文件13544.jstack中去搜索对应的代码



备注：

热点cpu问题的分析步骤：

1、找出java进程id; jps -m -l

2、找出该进程内最消耗cpu的线程：

3、线程id转化为十六进制；

4、使用jstack命令： jstack -l pid > D:\