# 介绍

## jdk、jre、jvm三者的关系

jdk > jre > jvm

jdk：全称 Java Development Kit

jre：全称 Java Runtime Environment

jvm：全称 Java Virtual Machine

javafx是applet应用程序插件

## 内存溢出场景模拟

### 实验代码：

|  |
| --- |
| **package** com.roocon.test1;  **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** Main {  **public** **static** **void** main(String[] args) {  List<Demo> demoList = **new** ArrayList<Demo>();  **while**(**true**){  demoList.add(**new** Demo());  }  }  } |

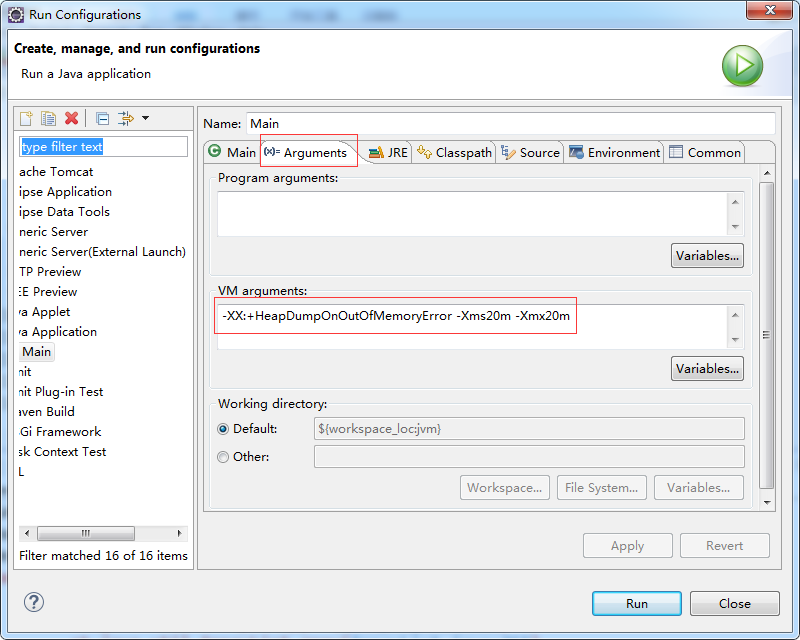
运行时间长了会出现内存溢出的错误：

|  |
| --- |
| Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  at java.util.Arrays.copyOf(Arrays.java:3210)  at java.util.Arrays.copyOf(Arrays.java:3181)  at java.util.ArrayList.grow(ArrayList.java:261)  at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:235)  at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:227)  at java.util.ArrayList.add(ArrayList.java:458)  at com.roocon.test1.Main.main(Main.java:10) |

### 找出问题：

1、导出堆快照

设置运行参数：

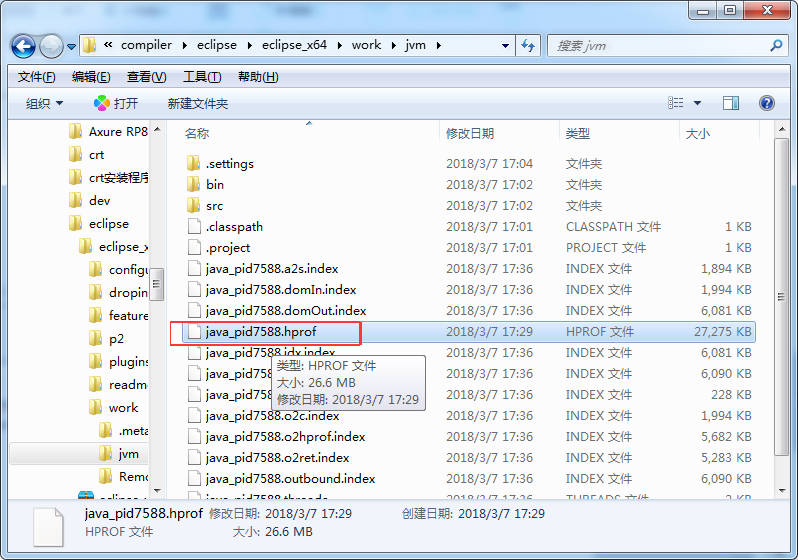


jvm中的参数

-XX:+HeapDumpOnOutOfMemoryError -Xms20m -Xmx20m

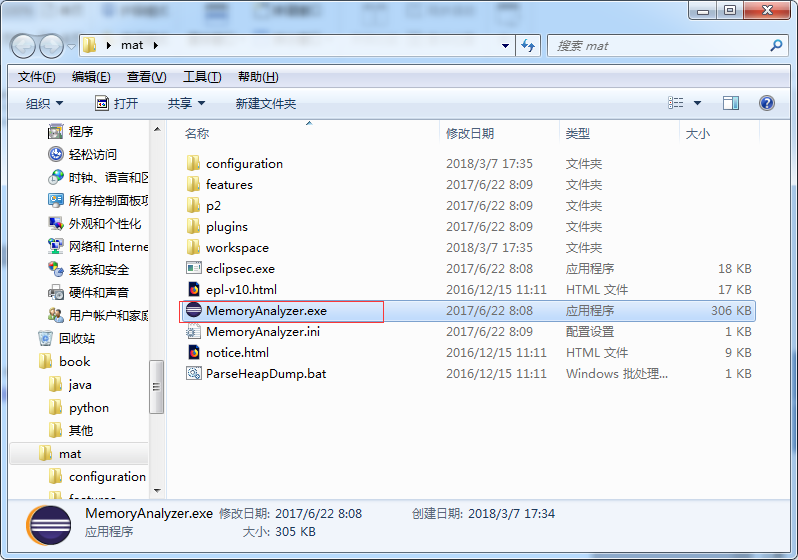
-Xms20m -Xmx20m 使堆内存只有20m

执行后在项目目录下生成一个堆快照文件

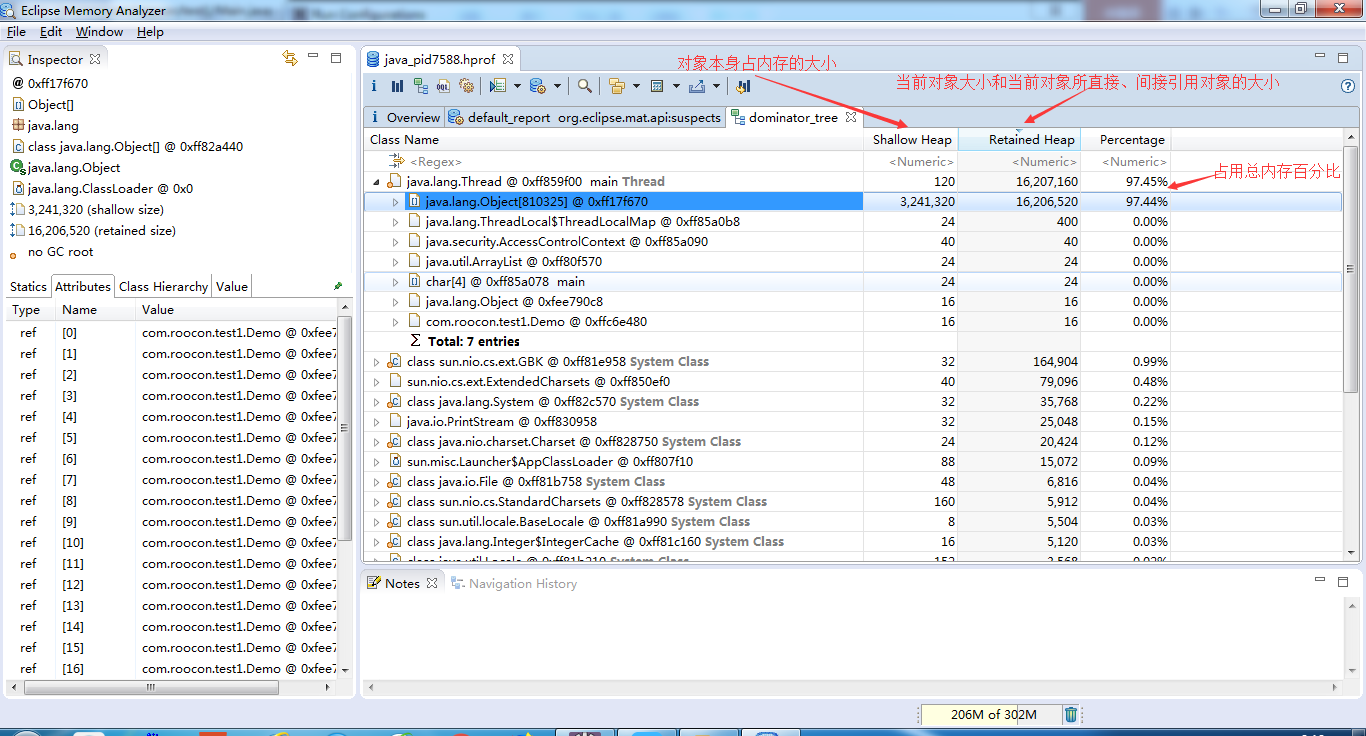


下载eclipse memory analyer，可以看到堆内存快照中的信息，下载地址：<http://www.eclipse.org/mat/downloads.php>

下载后点击下面的软件



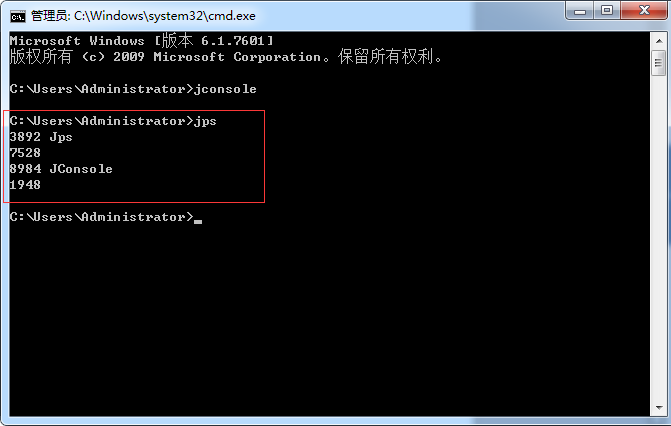
引入heap文件后可查看内存状态



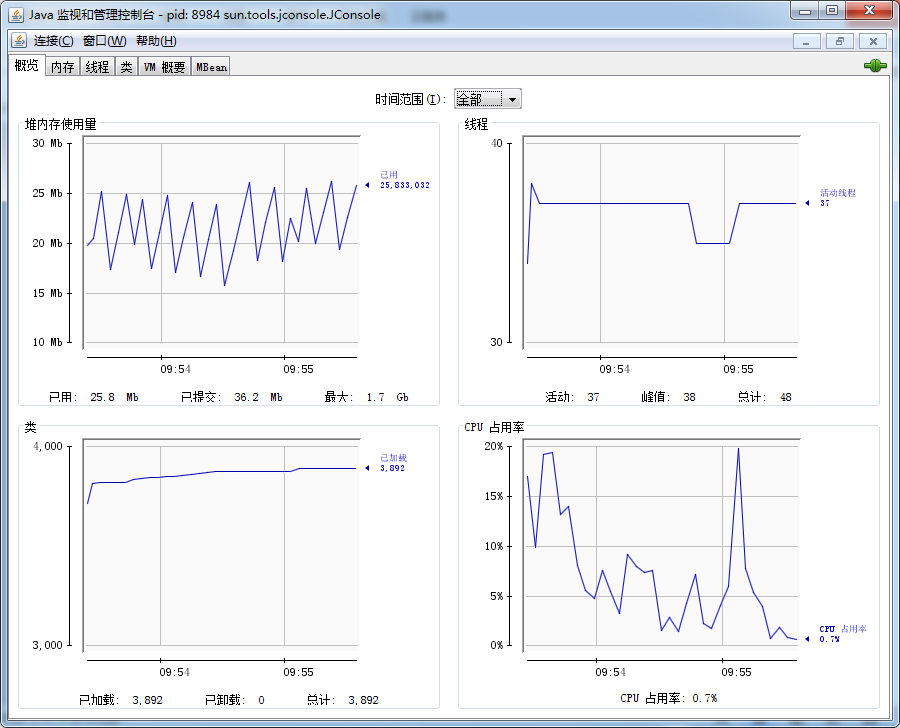
由分析工具定位到错误信息，定位错误位置，然后去修改

## jvm监控工具

jps：列出当前所有的java进程



jconsole:监控程序

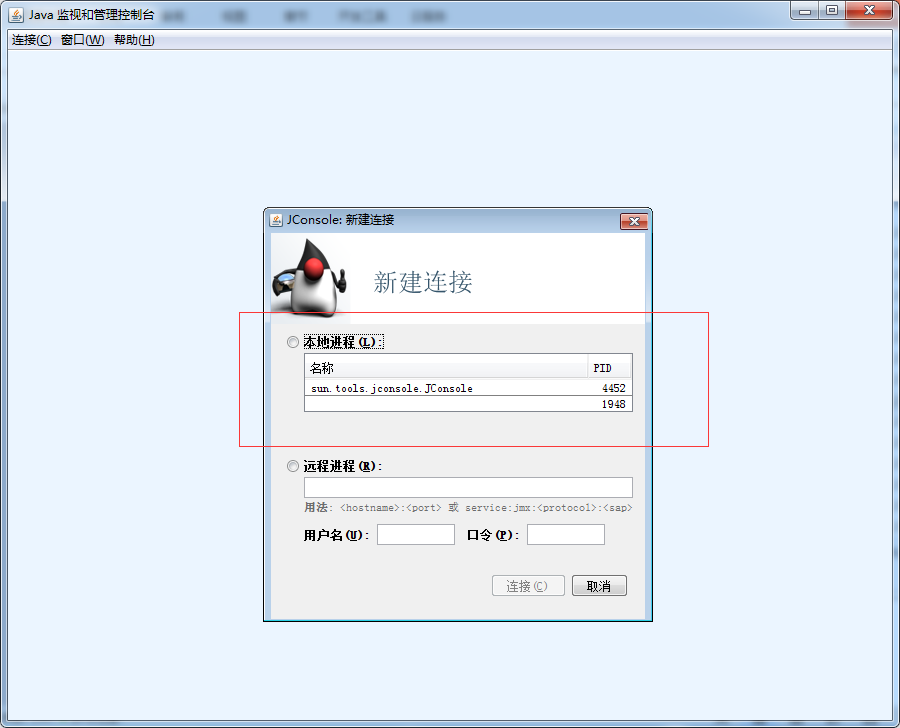


jconsole的使用：

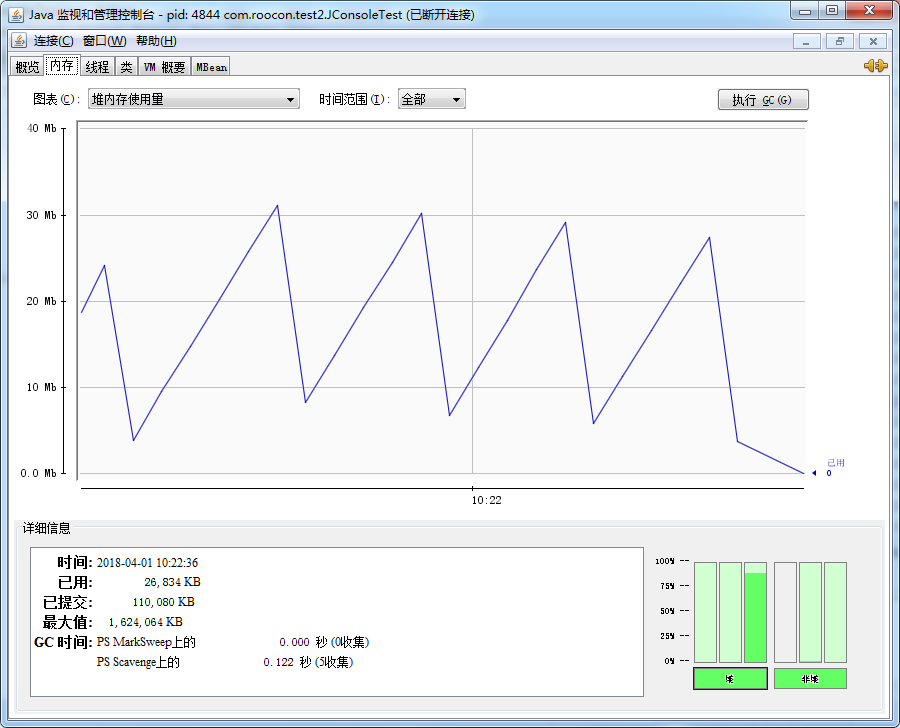
测试代码：

|  |
| --- |
| **package** com.roocon.test2;  **import** java.util.ArrayList;  **import** java.util.List;  **public** **class** JConsoleTest {  **public** JConsoleTest() {  **byte** []b1 = **new** **byte**[128 \* 1024];  }  **public** **static** **void** main(String[] args) {  **try** {  Thread.*sleep*(5000);  } **catch** (Exception e) {  e.printStackTrace();  }  System.***out***.println("start....");  *fill*(1000);  }    **private** **static** **void** fill(**int** n){  List<JConsoleTest> jConsoleTests = **new** ArrayList<>();  **for**(**int** i = 0; i < n; i++){  **try** {  Thread.*sleep*(100);  } **catch** (Exception e) {  e.printStackTrace();  }  jConsoleTests.add(**new** JConsoleTest());  }  System.***out***.println("end....");  }  } |

链接想要检测的进程



查看程序监控结果



## jdk8的新特性

Lambda表达式：

示例：

|  |
| --- |
| **package** com.roocon.test3;  **import** java.awt.event.ActionEvent;  **import** java.awt.event.ActionListener;  **import** javax.swing.JButton;  **import** javax.swing.JFrame;  **public** **class** Main **extends** JFrame {  **private** JButton jb;    **public** Main() {  **this**.setBounds(200, 200, 400, 200);  **this**.setTitle("lambda测试");    jb = **new** JButton("click");  //正常响应代码  /\*jb.addActionListener(new ActionListener() {    @Override  public void actionPerformed(ActionEvent e) {  System.out.println("clicked");    }  });\*/    //lambda表达式  jb.addActionListener(event -> System.***out***.println("hello"));    **this**.add(jb);    **this**.setVisible(**true**);  **this**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);  }    **public** **static** **void** main(String[] args) {  **new** Main();  }  } |

## java虚拟机内存管理

### java虚拟机栈

内存溢出异常演示（StackOverFlowError）

|  |
| --- |
| **package** com.roocon.test4;  **public** **class** StackTest {  **private** **void** tes() {  System.***out***.println("方法执行....");  tes();  }  **public** **static** **void** main(String[] args) {  **new** StackTest().tes();  }  } |

出现异常的现象：

Exception in thread "main" java.lang.StackOverflowError

at sun.nio.cs.UTF\_8$Encoder.encodeLoop(UTF\_8.java:691)

at java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:579)

at sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:271)

at sun.nio.cs.StreamEncoder.write(StreamEncoder.java:125)

at java.io.OutputStreamWriter.write(OutputStreamWriter.java:207)

at java.io.BufferedWriter.flushBuffer(BufferedWriter.java:129)

at java.io.PrintStream.write(PrintStream.java:526)

at java.io.PrintStream.print(PrintStream.java:669)

at java.io.PrintStream.println(PrintStream.java:806)

at com.roocon.test4.StackTest.tes(StackTest.java:6)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

at com.roocon.test4.StackTest.tes(StackTest.java:7)

### java堆

作用：存放对象实例、垃圾收集器管理的主要区域

方法放在栈中，方法中的对象放在堆中，对象名在栈中指向堆中的对象

# java虚拟机问题集

## HotSpot虚拟机如何为新生对象分配内存

1. 假如java堆中内存是绝对规整的，所有用过的内存都放在一边，空闲的内存放在另一边，中间放着一个指针作为分界点的指示器，那所分配内存就仅仅是把那个指针向空闲那边挪动一段与对象大小相等的距离，这种分配方式称为“指针碰撞”。
2. 如果java堆中的内存并不是规整的，已使用的内存和空闲内存相互交错，那就没有办法简单的进行指针碰撞了，虚拟机就必须维护一个列表，记录上那些内存块是可用的，在分配的时候从列表中找到一块足够大的空间划分个对象实例，并更新列表上的记录，这种分配方式称为“空闲列表”。

## 对象创建在虚拟机中是非常平凡的行为，在并发情况下，若两个对象同时申请同一地址，如何解决

1. 对分配内存空间的动作进行同步处理——实际上虚拟机采用CAS配上失败重试的方式保证更新操作的原子性。
2. 把内存分配的动作按照线程划分在不同的空间之中执行，即每个线程在java堆中预先分配一小块内存，称为本地线程分配缓冲（Thread Local Allocation Buffer, TLAB）。哪个线程要分配内存，就在哪个线程的TLAB上分配，只有TLAB用完并分配新的TLAB时，才需要同步锁定。