JVM

主讲老师:鲁班学院—华安

获取资料以及VIP咨询老师:嫦娥老师QQ：2746251334

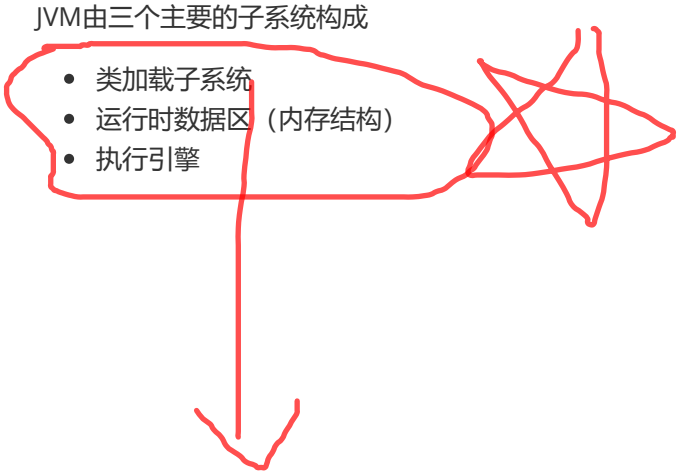
java虚拟机(java virtual machine，JVM)，一种能够运行java字节码的虚拟机。作为一种编程语言的虚拟机，实际上不只是专用于java语言，只要生成的编译文件匹配JVM对加载编译文件格式要求，任何语言都可以由JVM编译运行。比如kotlin、scala等。

jvm有很多，不只是Hotspot，还有JRockit、J9等等

JVM的基本结构

JVM由三个主要的子系统构成

- 类加载子系统
- 运行时数据区（内存结构）
- 执行引擎



类加载机制

动态连接，在运行中把符号引用(字符串常量)转换为直接引用

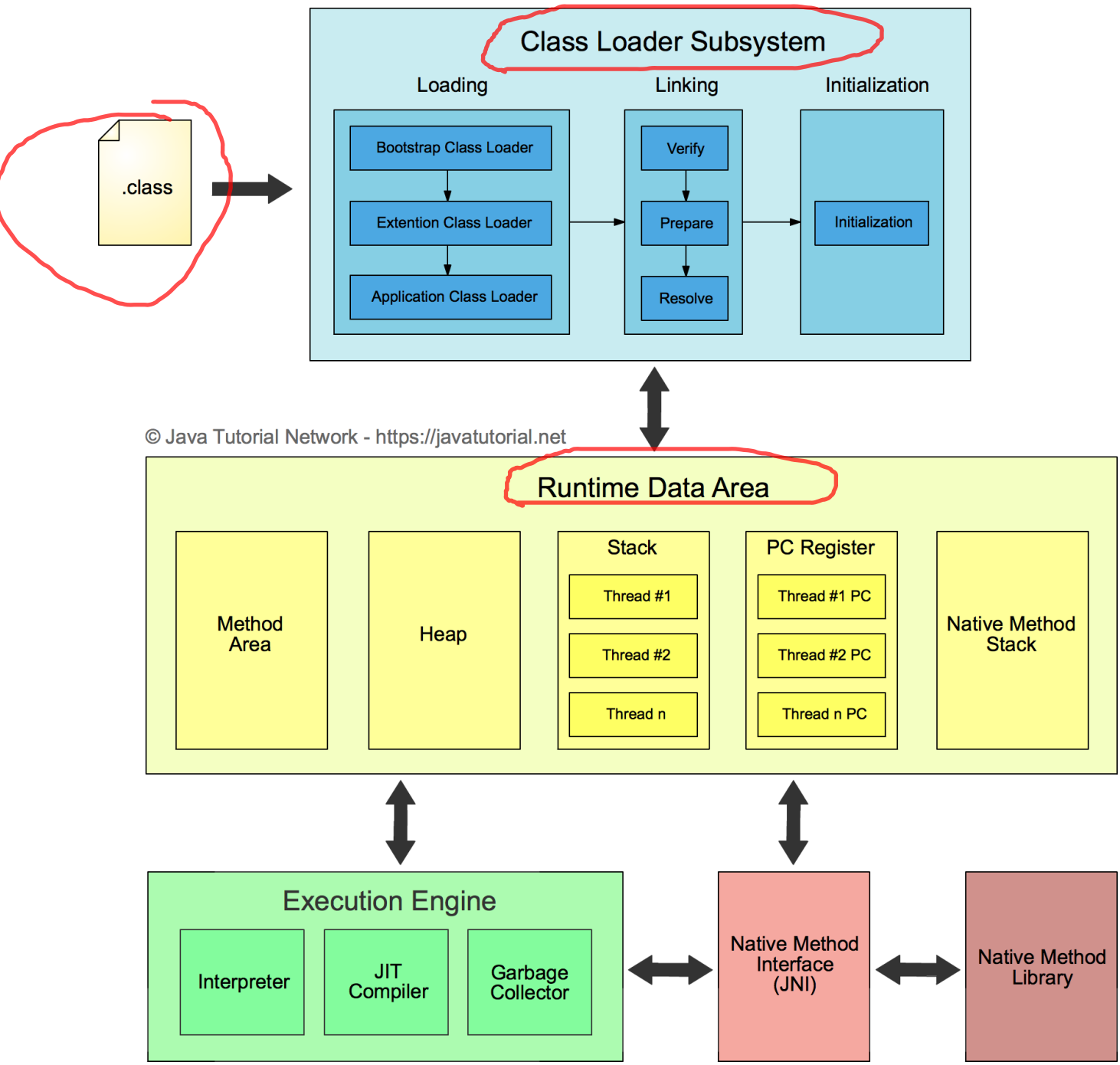
本地方法就是用c语言实现的，我们之间调用极可

程序计数器就是存放一个数，就是接下来你要执行的代码，为了找到代码，就为了找到你要执行的指令，程序计数器存放一个序号

进行运算的时候都是从操作数栈弹出两个值

方法出口就是记录一个指针，指向调用饿方法

n个栈帧，因为一个方法里面可以调用其他方法



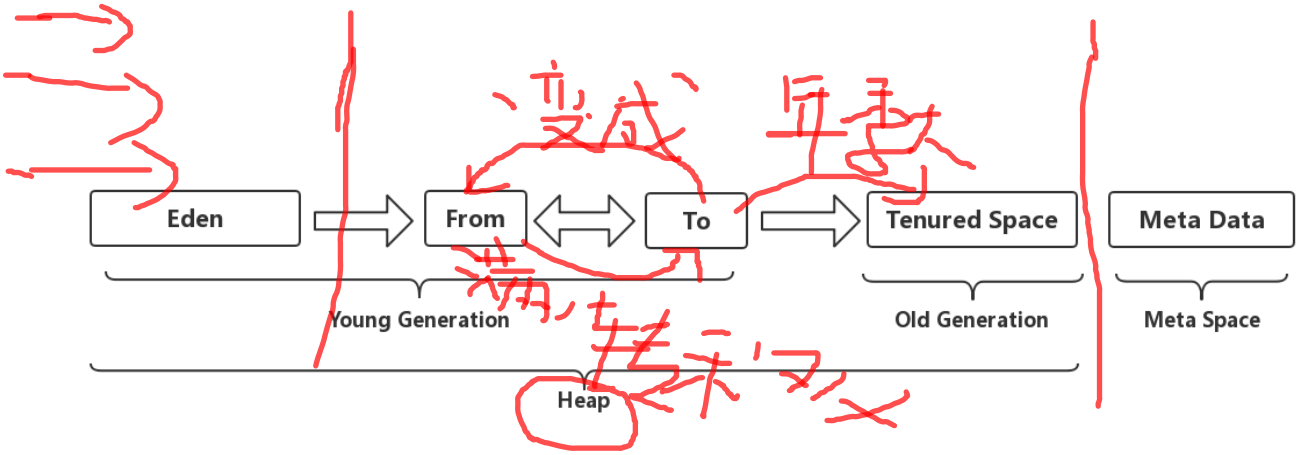
运行时数据区（内存结构）

1.方法区（Method Area）

类的所有字段和方法字节码，以及一些特殊方法如构造函数，接口代码也在这里定义。简单来说，所有定义的方法的信息都保存在该区域，静态变量+常量+类信息（构造方法/接口定义）+运行时常量池都存在方法区中，虽然Java虚拟机规范把方法区描述为堆的一个逻辑部分，但是它却有一个别名叫做Non-Heap（非堆），目的应该是为了和Java的堆区分开

2.堆（Heap）

虚拟机启动时自动分配创建，用于存放对象的实例，几乎所有对象都在堆上分配内存，当对象无法在该空间申请到内存是将抛出OutOfMemoryError异常。同时也是垃圾收集器管理的主要区域。



2.1 新生代（Young Generation）

类出生、成长、消亡的区域，一个类在这里产生，应用，最后被垃圾回收器收集，

结束生命。

新生代分为两部分：伊甸区（Eden space）和幸存者区（Survivor space），所有的类都是在伊甸区被new出来的。幸存区又分为From和To区。当Eden区的空间用完是，程序又需要创建对象，JVM的垃圾回收器将Eden区进行垃圾回收（Minor GC），将Eden区中的不再被其它对象应用的对象进行销毁。然后将Eden区中剩余的对象移到From Survivor区。若From Survivor区也满了，再对该区进行垃圾回收，然后移动到To Survivor区。

2.2 老年代（Old Generation）

新生代经过多次GC仍然存货的对象移动到老年区。若老年代也满了，这时候将发生Major GC（也可以叫Full GC）进行老年区的内存清理。若老年区执行了Full GC之后发现依然无法进行对象的保存，就会抛出OOM（OutOfMemoryError）异常

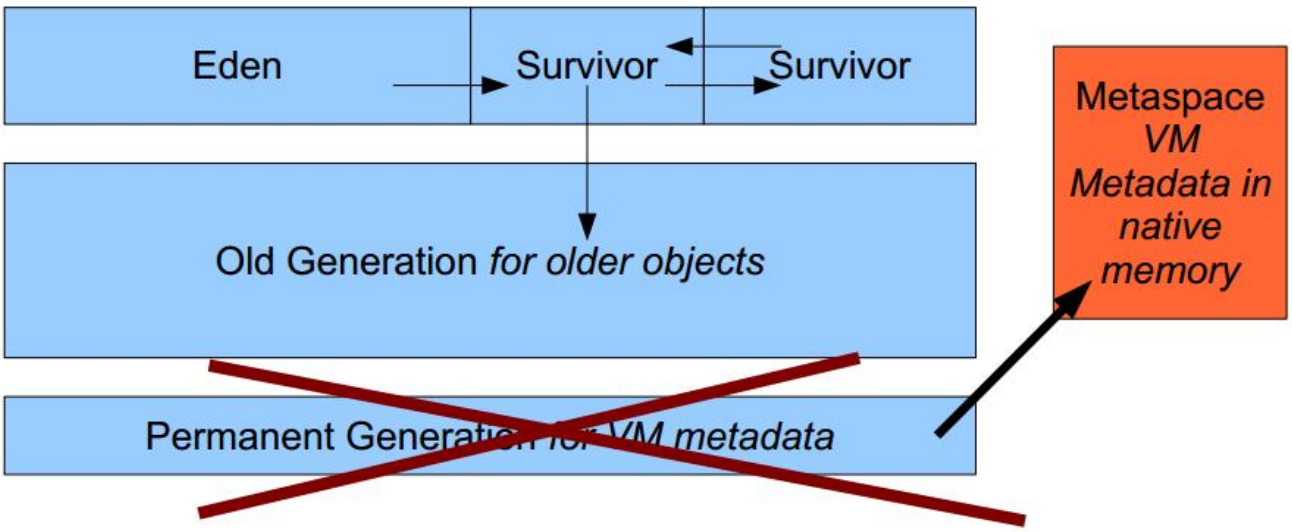
2.3 元空间（Meta Space）

在JDK1.8之后，元空间替代了永久代，它是对JVM规范中方法区的实现，区别在于元数据区不在虚拟机当中，而是用的本地内存，永久代在虚拟机当中，永久代逻辑结构上也属于堆，但是物理上不属于。

为什么移除了永久代?

参考官方解释http://openjdk.java.net/jeps/122

大概意思是移除永久代是为融合HotSpot与 JRockit而做出的努力，因为JRockit没有永久代，不需要配置永久代。



3.栈(Stack)

Java线程执行方法的内存模型，一个线程对应一个栈，每个方法在执行的同时都会创建一个栈帧（用于存储局部变量表，操作数栈，动态链接，方法出口等信息）不存在垃圾回收问题，只要线程一结束该栈就释放，生命周期和线程一致

4.本地方法栈(Native Method Stack)

和栈作用很相似，区别不过是Java栈为JVM执行Java方法服务，而本地方法栈为JVM执行native方法服务。登记native方法，在Execution Engine执行时加载本地方法库

5.程序计数器(Program Counter Register)

就是一个指针，指向方法区中的方法字节码（用来存储指向吓一跳指令的地址，也即将要执行的指令代码），由执行引擎读取下一条指令，是一个非常小的内存空间，几乎可以忽略不计

JDK性能调优监控工具

Jinfo

查看正在运行的Java程序的扩展参数

查看JVM的参数

```
D:\>jinfo -flags 7824
Attaching to process ID 7824, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.73-b02
Non-default VM flags: -XX:CICompilerCount=4 -XX:InitialHeapSize=134217728 -XX:MaxHeapSize=2124414976 -XX:MaxNewSize=707788800
-XX:MinHeapDeltaBytes=524288 -XX:NewSize=44564480 -XX:OldSize=89653248 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops
-XX:+UseFastUnorderedTimeStamps -XX:-UseLargePagesIndividualAllocation -XX:+UseParallelGC
Command line:
```

查看java系统属性

等同于System.getProperties()

```
D:\>jinfo -sysprops 7824
Attaching to process ID 7824, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.73-b02
java.runtime.name = Java(TM) SE Runtime Environment
java.vm.version = 25.73-b02
sun.boot.library.path = C:\Program Files\Java\jdk1.8.0_73\jre\bin
java.protocol.handler.pkgs = org.springframework.boot.loader
java.vendor.url = http://java.oracle.com/
java.vm.vendor = Oracle Corporation
path.separator = ;
file.encoding.pkg = sun.io
java.vm.name = Java HotSpot(TM) 64-Bit Server VM
sun.os.patch.level =
sun.java.launcher = SUN_STANDARD
user.script =
user.country = CN
user.dir = D:\
java.vm.specification.name = Java Virtual Machine Specification
PID = 7824
java.runtime.version = 1.8.0_73-b02
java.awt.graphicsenv = sun.awt.Win32GraphicsEnvironment
os.arch = amd64
java.endorsed.dirs = C:\Program Files\Java\jdk1.8.0_73\jre\lib\endorsed
line.separator =

java.io.tmpdir = C:\Users\colde\AppData\Local\Temp\
java.vm.specification.vendor = Oracle Corporation
user.variant =
os.name = Windows 10
sun.jnu.encoding = GBK
java.library.path = C:\Program Files\Java\jdk1.8.0_73\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOWS;D:\work\WorkS
oft\;C:\Program Files\Java\jdk1.8.0_73\bin;C:\Program Files\Java\jdk1.8.0_73\jre\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS
\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;D:\work\WorkSoft\apache-maven-3.6.0\bi
n\;D:\Soft\pycSafefile\x64;C:\Program Files (x86)\Pandoc\;D:\work\WorkSoft\gradle-4.9\bin;C:\Users\colde\AppData\Local\Microso
ft\WindowsApps;;.
spring.beaninfo.ignore = true
```

Jstat

jstat命令可以查看堆内存各部分的使用量，以及加载类的数量。命令格式：

jstat [-命令选项] [vmid] [间隔时间/毫秒] [查询次数]

Jmap

可以用来查看内存信息

堆的对象统计

```
jmap -histo 7824 > xxx.txt
```

如图：

```
 num     #instances         #bytes  class name
----------------------------------------------
   1:         18829      143011888  [I
   2:        680830      125590192  [C
   3:       1164734       37271488  java.util.concurrent.locks.AbstractQueuedSynchronizer$Node
   4:         22790       11492832  [B
   5:        451949       10846776  java.lang.String
   6:         74841        4789824  java.net.URL
   7:         42656        3753728  java.lang.reflect.Method
   8:        115310        3689920  org.springframework.boot.loader.jar.StringSequence
   9:         57398        3214288  java.util.LinkedHashMap
  10:         46245        2323200  [Ljava.lang.Object;
  11:         30449        2083584  [Ljava.util.HashMap$Node;
  12:         48879        1955160  java.util.LinkedHashMap$Entry
  13:         59480        1903360  java.util.concurrent.ConcurrentHashMap$Node
  14:         90053        1898440  [Ljava.lang.Class;
  15:         60960        1463040  java.lang.StringBuffer
  16:         12314        1359880  java.lang.Class
  17:         56533        1356792  org.springframework.boot.loader.jar.JarURLConnection$JarEntryName
  18:         32990        1055680  java.util.HashMap$Node
  19:         31574        1043656  [Ljava.lang.String;
  20:         27354         875328  java.lang.ref.WeakReference
  21:         11991         863352  java.lang.reflect.Field
  22:         10505         839184  [S
  23:           281         630064  [Ljava.util.concurrent.ConcurrentHashMap$Node;
  24:         12766         612768  java.util.HashMap
  25:         25378         609072  java.lang.StringBuilder
  26:          7258         522576  org.springframework.core.type.classreading.AnnotationMetadataReadingVisitor
  27:         12935         517400  java.lang.ref.SoftReference
  28:         12700         506776  [Ljava.lang.reflect.Method;
  29:         30525         488400  java.lang.Object
  30:         12199         487960  java.util.HashMap$KeyIterator
  31:          9820         471360  org.springframework.core.ResolvableType
  32:         29333         469328  java.util.LinkedHashSet
  33:         17418         418032  java.util.ArrayList
  34:          5615         404280  org.springframework.core.annotation.AnnotationAttributes
  35:          7189         402584  java.beans.MethodDescriptor
```

- Num：序号
- Instances：实例数量
- Bytes：占用空间大小
- Class Name：类名

堆信息

```
D:\>jmap -histo 7824 > log.txt

D:\>jmap -heap 7824
Attaching to process ID 7824, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.73-b02

using thread-local object allocation.
Parallel GC with 8 thread(s)

Heap Configuration:
   MinHeapFreeRatio         = 0
   MaxHeapFreeRatio         = 100
   MaxHeapSize              = 2124414976 (2026.0MB)
   NewSize                  = 44564480 (42.5MB)
   MaxNewSize               = 707788800 (675.0MB)
   OldSize                  = 89653248 (85.5MB)
   NewRatio                 = 2
   SurvivorRatio            = 8
   MetaspaceSize            = 21807104 (20.796875MB)
   CompressedClassSpaceSize = 1073741824 (1024.0MB)
   MaxMetaspaceSize         = 17592186044415 MB
   G1HeapRegionSize         = 0 (0.0MB)

Heap Usage:
PS Young Generation
Eden Space:
   capacity = 632815616 (603.5MB)
   used     = 329940176 (314.6554718017578MB)
   free     = 302875440 (288.8445281982422MB)
   52.138437746770144% used
From Space:
   capacity = 12582912 (12.0MB)
   used     = 8740536 (8.335624694824219MB)
   free     = 3842376 (3.6643753051757812MB)
   69.46353912353516% used
To Space:
   capacity = 13107200 (12.5MB)
   used     = 0 (0.0MB)
   free     = 13107200 (12.5MB)
   0.0% used
PS Old Generation
   capacity = 95944704 (91.5MB)
   used     = 44983840 (42.899932861328125MB)
   free     = 50960864 (48.600067138671875MB)
   46.885172526041664% used

26884 interned Strings occupying 3314192 bytes.
```

堆内存dump

```
D:\>jmap -dump:format=b,file=eureka.hprof 7824
Dumping heap to D:\eureka.hprof ...
Heap dump file created
```

jmap -dump:format=b,file=temp.hprof

也可以在设置内存溢出的时候自动导出dump文件（项目内存很大的时候，可能会导不出来）

1.-XX:+HeapDumpOnOutOfMemoryError

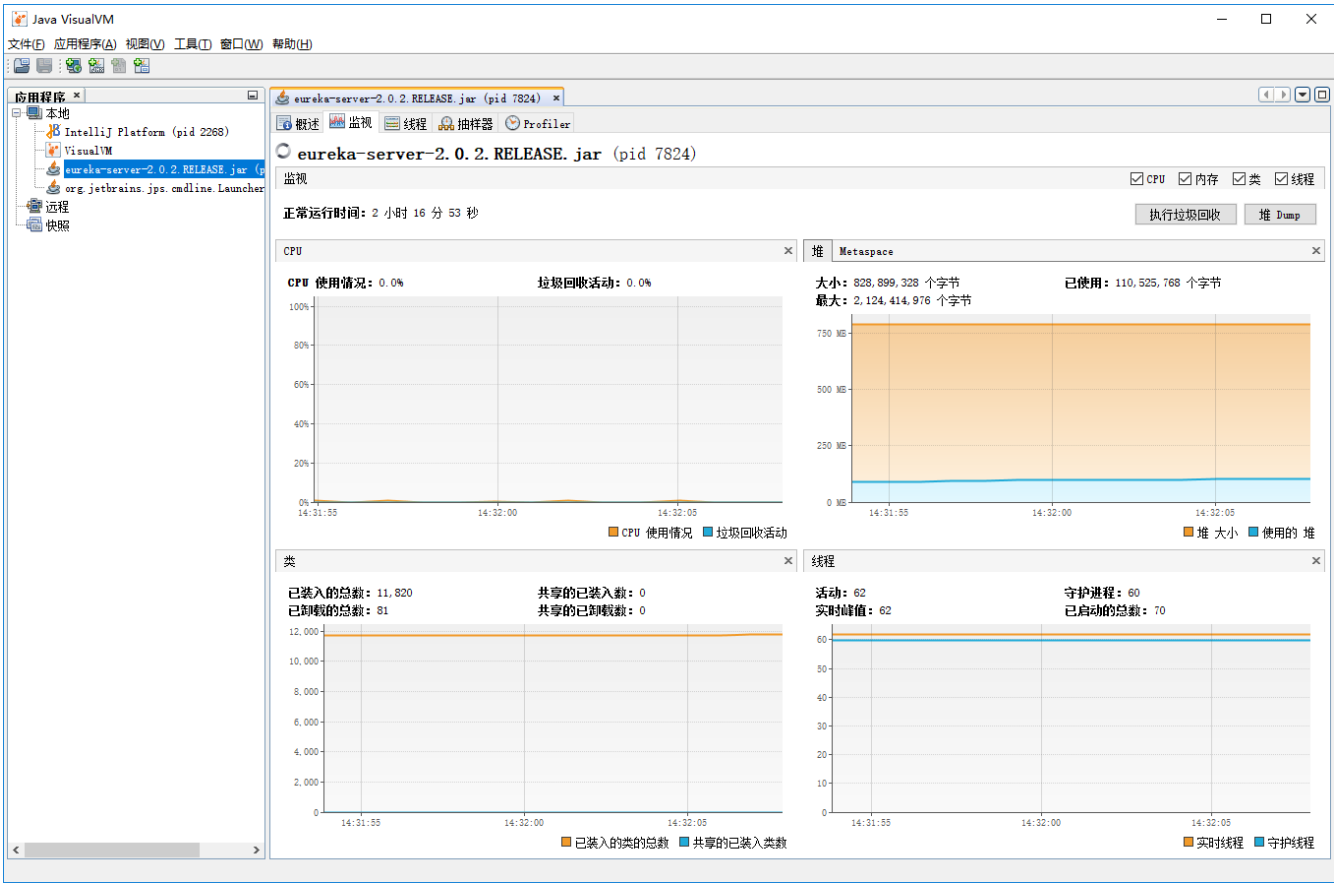
2.-XX:HeapDumpPath=输出路径

```
-Xms10m -Xmx10m -XX:+PrintGCDetails -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=d:\oomdump.dump
```

```java
public class OutOfMemoryDump {

    /**
     *  设置JVM参数
     *  -Xms10m
     *  -Xmx10m
     *  -XX:+PrintGCDetails
     *  -XX:+HeapDumpOnOutOfMemoryError
     *  -XX:HeapDumpPath=./（路径）
     */
    public static void main(String[] args) {
        List<Object> list = new ArrayList<>();
        int i = 0;
        while(true){
            list.add(new User(i++,UUID.randomUUID().toString()));
        }
    }
```

可以使用jvisualvm命令工具导入文件分析



Jstack

jstack用于生成java虚拟机当前时刻的线程快照。

```
D:\>jstack 7824
2019-05-26 15:01:56
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.73-b02 mixed mode):

"JMX server connection timeout 76" #76 daemon prio=5 os_prio=0 tid=0x000000001d359800 nid=0x3a7c in Object.wait() [0x000000002be6f000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at com.sun.jmx.remote.internal.ServerCommunicatorAdmin$Timeout.run(ServerCommunicatorAdmin.java:168)
        - locked <0x00000000ff3b0178> (a [I)
        at java.lang.Thread.run(Thread.java:745)

"RMI Scheduler(0)" #75 daemon prio=5 os_prio=0 tid=0x000000001d355800 nid=0x3ba8 waiting on condition [0x000000002bd6f000]
   java.lang.Thread.State: TIMED_WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x0000000083c9ad38> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2078)
        at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:1093)
        at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:809)
        at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)

"RMI TCP Accept-0" #73 daemon prio=5 os_prio=0 tid=0x00000000225db800 nid=0x3344 runnable [0x000000002ba6e000]
   java.lang.Thread.State: RUNNABLE
        at java.net.Dual1StackPlainSocketImpl.accept0(Native Method)
        at java.net.Dual1StackPlainSocketImpl.socketAccept(Dual1StackPlainSocketImpl.java:131)
        at java.net.AbstractPlainSocketImpl.accept(AbstractPlainSocketImpl.java:409)
        at java.net.PlainSocketImpl.accept(PlainSocketImpl.java:199)
        - locked <0x0000000083ac1830> (a java.net.SocksSocketImpl)
        at java.net.ServerSocket.implAccept(ServerSocket.java:545)
        at java.net.ServerSocket.accept(ServerSocket.java:513)
        at sun.management.jmxremote.Local1RMIServerSocketFactory$1.accept(Local1RMIServerSocketFactory.java:52)
        at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.executeAcceptLoop(TCPTransport.java:400)
        at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.run(TCPTransport.java:372)
        at java.lang.Thread.run(Thread.java:745)

"DestroyJavaVM" #72 prio=5 os_prio=0 tid=0x00000000225db000 nid=0x3a64 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"http-nio-8080-AsyncTimeout" #70 daemon prio=5 os_prio=0 tid=0x00000000225da000 nid=0x36bc waiting on condition [0x000000002af6f000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at org.apache.coyote.AbstractProtocol$AsyncTimeout.run(AbstractProtocol.java:1133)
        at java.lang.Thread.run(Thread.java:745)
```