1、**查看java进程**

jps -l

|  |
| --- |
| [root@10 credit-mobile-access]# jps -l  8567 sun.tools.jps.Jps  7010 com.alibaba.dubbo.container.Main  7255 /data/app/credit\_mobile\_access/boot/credit-mobile-access-deploy.jar |

2、**查看java 堆（heap）使用情况**jmap -heap 7255

|  |
| --- |
| **Attaching to process ID 31846, please wait...**  **Debugger attached successfully.**  **Server compiler detected.**  **JVM version is 24.71-b01**  **using thread-local object allocation.**  **Parallel GC with 4 thread(s)//GC 方式**  **Heap Configuration: //堆内存初始化配置**  **MinHeapFreeRatio = 0 //对应jvm启动参数-XX:MinHeapFreeRatio设置JVM堆最小空闲比率(default 40)**  **MaxHeapFreeRatio = 100 //对应jvm启动参数 -XX:MaxHeapFreeRatio设置JVM堆最大空闲比率(default 70)**  **MaxHeapSize = 2082471936 (1986.0MB) //对应jvm启动参数-XX:MaxHeapSize=设置JVM堆的最大大小**  **NewSize = 1310720 (1.25MB)//对应jvm启动参数-XX:NewSize=设置JVM堆的‘新生代’的默认大小**  **MaxNewSize = 17592186044415 MB//对应jvm启动参数-XX:MaxNewSize=设置JVM堆的‘新生代’的最大大小**  **OldSize = 5439488 (5.1875MB)//对应jvm启动参数-XX:OldSize=<value>:设置JVM堆的‘老生代’的大小**  **NewRatio = 2 //对应jvm启动参数-XX:NewRatio=:‘新生代’和‘老生代’的大小比率**  **SurvivorRatio = 8 //对应jvm启动参数-XX:SurvivorRatio=设置年轻代中Eden区与Survivor区的大小比值**  **PermSize = 21757952 (20.75MB) //对应jvm启动参数-XX:PermSize=<value>:设置JVM堆的‘永生代’的初始大小**  **MaxPermSize = 85983232 (82.0MB)//对应jvm启动参数-XX:MaxPermSize=<value>:设置JVM堆的‘永生代’的最大大小**  **G1HeapRegionSize = 0 (0.0MB)**  **Heap Usage://堆内存使用情况**  **PS Young Generation**  **Eden Space://Eden区内存分布**  **capacity = 33030144 (31.5MB)//Eden区总容量**  **used = 1524040 (1.4534378051757812MB) //Eden区已使用**  **free = 31506104 (30.04656219482422MB) //Eden区剩余容量**  **4.614088270399305% used //Eden区使用比率**  **From Space: //其中一个Survivor区的内存分布**  **capacity = 5242880 (5.0MB)**  **used = 0 (0.0MB)**  **free = 5242880 (5.0MB)**  **0.0% used**  **To Space: //另一个Survivor区的内存分布**  **capacity = 5242880 (5.0MB)**  **used = 0 (0.0MB)**  **free = 5242880 (5.0MB)**  **0.0% used**  **PS Old Generation //当前的Old区内存分布**  **capacity = 86507520 (82.5MB)**  **used = 0 (0.0MB)**  **free = 86507520 (82.5MB)**  **0.0% used**  **PS Perm Generation//当前的 “永生代” 内存分布**  **capacity = 22020096 (21.0MB)**  **used = 2496528 (2.3808746337890625MB)**  **free = 19523568 (18.619125366210938MB)**  **11.337498256138392% used**  **670 interned Strings occupying 43720 bytes.** |

|  |
| --- |
| **num #instances #bytes class name**  **编号 个数 字节 类名**  **----------------------------------------------**  **1: 7 1322080 [I**  **2: 5603 722368 <methodKlass>**  **3: 5603 641944 <constMethodKlass>**  **4: 34022 544352 java.lang.Integer**  **5: 371 437208 <constantPoolKlass>**  **6: 336 270624 <constantPoolCacheKlass>**  **7: 371 253816 <instanceKlassKlass>** |

1. **查看堆内存(histogram)中的对象数量及大小**。  
   **jmap -histo 3331**
2. **将内存使用的详细情况输出到文件  
   jmap -dump:format=b,file=heapDump 6900**

**总结**

**1.如果程序内存不足或者频繁GC，很有可能存在内存泄露情况，这时候就要借助Java堆Dump查看对象的情况。 2.要制作堆Dump可以直接使用jvm自带的jmap命令**

**3.可以先使用jmap -heap命令查看堆的使用情况，看一下各个堆空间的占用情况。**

**4.使用jmap -histo:[live]查看堆内存中的对象的情况。如果有大量对象在持续被引用，并没有被释放掉，那就产生了内存泄露，就要结合代码，把不用的对象释放掉。**

**5.也可以使用 jmap -dump:format=b,file=<fileName>命令将堆信息保存到一个文件中，再借助jhat命令查看详细内容**

**6.在内存出现泄露、溢出或者其它前提条件下，建议多dump几次内存，把内存文件进行编号归档，便于后续内存整理分析。**