JVM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：

```
-bash-4.2$ jps
77921 Bootstrap
125472 icac-bak.jar
41890 nacos-server.jar
18434 icac-coll.jar
845 apv-biz.jar
64365 icac-gateway.jar
125071 icac-auth.jar
5198 legal-urge-biz.jar
42447 nacos-server.jar
119824 Jps
128882 icac-upms.jar
94162 icac-coll-v2.jar
129045 icac-workflow.jar
127512 icac-job.war
90843 plmm-biz.jar
127130 icac-log.jar
88285 icac-sync.jar
-bash-4.2$ jinfo -flags 18434
Attaching to process ID 18434, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.144-b01
Non-default VM flags: -XX:CICompilerCount=3 -XX:InitialHeapSize=268435456 -XX:MaxHeapSize=268435456 -XX:MaxNewSize=89128960 -XX:MinHeapDeltaBytes=524288 -XX:
ers -XX:+UseCompressedOops -XX:+UseParallelGC
Command line:  -Xms256m -Xmx256m
```

查看java系统属性 jinfo -sysprops 18434

相当于System.getProperties()

```
-bash-4.2$ jinfo -sysprops 845
Attaching to process ID 845, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.144-b01
java.runtime.name = Java(TM) SE Runtime Environment
java.vm.version = 25.144-b01
sun.boot.library.path = /data/jdk/1.8.0_144/jre/lib/amd64
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 = unknown
sun.java.launcher = SUN_STANDARD
user.country = US
user.dir = /data/tomcat/instance/icac/apv-biz
java.vm.specification.name = Java Virtual Machine Specification
PID = 845
java.runtime.version = 1.8.0_144-b01
java.awt.graphicsenv = sun.awt.X11GraphicsEnvironment
os.arch = amd64
java.endorsed.dirs = /data/jdk/1.8.0_144/jre/lib/endorsed
line.separator =

java.io.tmpdir = /tmp
java.vm.specification.vendor = Oracle Corporation
os.name = Linux
http.maxRedirects = 5
sun.jnu.encoding = UTF-8
java.library.path = /usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
spring.beaninfo.ignore = true
java.specification.name = Java Platform API Specification
java.class.version = 52.0
sun.management.compiler = HotSpot 64-Bit Tiered Compilers
os.version = 3.10.0-693.el7.x86_64
user.home = /data/tomcat
user.timezone = Asia/Shanghai
```

```
os.version = 3.10.0-693.el7.x86_64
user.home = /data/tomcat
user.timezone = Asia/Shanghai
catalina.useNaming = false
java.awt.printerjob = sun.print.PSPrinterJob
file.encoding = UTF-8
@appId = icac-apv
java.specification.version = 1.8
catalina.home = /tmp/tomcat.1206763797159207035.9036
user.name = tomcat
java.class.path = /data/tomcat/instance/icac/apv-biz/apv-biz.jar
java.vm.specification.version = 1.8
sun.arch.data.model = 64
sun.java.command = /data/tomcat/instance/icac/apv-biz/apv-biz.jar
java.home = /data/jdk/1.8.0_144/jre
user.language = en
java.specification.vendor = Oracle Corporation
awt.toolkit = sun.awt.X11.XToolkit
java.vm.info = mixed mode
java.version = 1.8.0_144
java.ext.dirs = /data/jdk/1.8.0_144/jre/lib/ext:/usr/java/packages/lib/ext
sun.boot.class.path = /data/jdk/1.8.0_144/jre/lib/resources.jar:/data/jdk/1.8.0_144
_144/jre/lib/jsse.jar:/data/jdk/1.8.0_144/jre/lib/jce.jar:/data/jdk/1.8.0_144/jre/l
e/classes
java.awt.headless = true
java.vendor = Oracle Corporation
catalina.base = /tmp/tomcat.1206763797159207035.9036
file.separator = /
java.vendor.url.bug = http://bugreport.sun.com/bugreport/
sun.io.unicode.encoding = UnicodeLittle
sun.cpu.endian = little
sun.cpu.isalist =
-bash-4.2$
```

jstat

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] [时间间隔/毫秒] [查询次数]

```
-bash-4.2$ jstat
invalid argument count
Usage: jstat -help|-options
       jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

Definitions:
  <option>      An option reported by the -options option
  <vmid>        Virtual Machine Identifier. A vmid takes the following form:
                    <lvmid>[@<hostname>[:<port>]]
                Where <lvmid> is the local vm identifier for the target
                Java virtual machine, typically a process id; <hostname> is
                the name of the host running the target Java virtual machine;
                and <port> is the port number for the rmiregistry on the
                target host. See the jvmstat documentation for a more complete
                description of the Virtual Machine Identifier.
  <lines>       Number of samples between header lines.
  <interval>    Sampling interval. The following forms are allowed:
                    <n>["ms"|"s"]
                Where <n> is an integer and the suffix specifies the units as
                milliseconds("ms") or seconds("s"). The default units are "ms".
  <count>       Number of samples to take before terminating.
  -J<flag>      Pass <flag> directly to the runtime system.
-bash-4.2$ jstat -gc 845
 S0C    S1C    S0U    S1U      EC       EU        OC         OU       MC     MU    CCSC   CCSU   YGC     YGCT    FGC    FGCT     GCT
512.0  512.0  384.0   0.0   86016.0  20143.9   175104.0   27542.5   59284.0 56880.9 7168.0 6700.5   2138   11.454   3      1.344   12.798
-bash-4.2$
```

jmap

用来查看内存信息

堆的对象统计

jmap -histo 845 > xxx.txt

num:序号

instances 实例数量

bytes 占用空间大小

class name 类名

```
num     #instances         #bytes  class name
----------------------------------------------
  1:        170337       30826152  [C
  2:          7792       19074080  [I
  3:         30793       13339624  [B
  4:        104877        2517048  java.lang.String
  5:         18170        1598960  java.lang.reflect.Method
  6:         37073        1186336  java.util.concurrent.ConcurrentHashMap$Node
  7:         10321        1154568  java.lang.Class
  8:         17677        1123384  [Ljava.lang.Object;
  9:         20239         647648  java.util.HashMap$Node
 10:          8200         615440  [Ljava.util.HashMap$Node;
 11:         14097         563880  java.util.LinkedHashMap$Entry
 12:         30002         480032  java.lang.Object
 13:          8230         396888  [Ljava.lang.String;
 14:          6848         383488  java.util.LinkedHashMap
 15:           179         375840  [Ljava.util.concurrent.ConcurrentHashMap$Node;
 16:         12782         293112  [Ljava.lang.Class;
 17:          4859         272104  sun.nio.cs.UTF_8$Encoder
 18:         10448         250752  java.util.ArrayList
 19:          7726         185424  java.lang.StringBuilder
 20:             3         165936  [Lcom.alibaba.druid.sql.parser.SymbolTable$Entry;
 21:          3409         163632  java.util.HashMap
 22:          1448         162176  java.net.SocksSocketImpl
 23:          3334         160032  java.nio.HeapCharBuffer
 24:          3402         136080  java.lang.ref.SoftReference
 25:          5653         135672  org.springframework.core.MethodClassKey
 26:          1245         119520  org.springframework.beans.GenericTypeAwarePropertyDescriptor
 27:          3566         114112  java.util.Hashtable$Entry
 28:          3251         104032  java.lang.StackTraceElement
 29:          2546         101840  java.util.HashMap$KeyIterator
 30:             3          98352  [Lcom.alibaba.fastjson.util.IdentityHashMap$Entry;
 31:          2439          97560  com.sun.org.apache.xerces.internal.dom.DeferredTextImpl
 32:          2917          93344  java.lang.ref.WeakReference
 33:          2253          90120  java.util.TreeMap$Entry
 34:          1386          88704  com.mysql.jdbc.ConnectionPropertiesImpl$BooleanConnectionProperty
 35:          2652          84864  java.util.LinkedList
 36:           992          79360  java.lang.reflect.Constructor
 37:          1960          78400  java.lang.ref.Finalizer
 38:           334          77488  sun.net.www.protocol.http.HttpURLConnection
root.txt
```

堆内存 dump

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:\oom\oom.dump
```

linux 没有权限安装监控工具 只能项目运行时添加jvm参数。发生异常自动生成dump

从远程下载下来，离线分析dump文件

JvisualVM

文件——>装载——>堆 Dump(*.hprof, * . *)——>类

jtack

用于生成java虚拟机当前时刻的线程快照。

死锁案例：

```
public class DeadLockDemo2 {

    private static Object lock1 = new Object();
    private static Object lock2 = new Object();

    public static void main(String[] args) {
        ExecutorService service = Executors.newCachedThreadPool();
        service.execute(() -> {
            while (true) {
                synchronized (lock1) {
                    System.out.println("thread1 获取锁 lock1");
                    synchronized (lock2) {
                        System.out.println("thread1 获取锁 lock2");
                    }
                }
            }
        });

        service.execute(() -> {
            while (true) {
                synchronized (lock2) {
                    System.out.println("thread2 获取锁 lock2");
                    synchronized (lock1) {
                        System.out.println("thread2 获取锁 lock1");
                    }
                }
            }
        });

        service.shutdown();

    }

}
```


```
C:\Users\Administrator>jstack 244836
2019-09-04 16:56:42
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.171-b11 mixed mode):

"DestroyJavaVM" #13 prio=5 os_prio=0 tid=0x00000000031f3800 nid=0x1de70 waiting on
condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"thread2" #12 prio=5 os_prio=0 tid=0x000000001e09d800 nid=0x3fb24 waiting for monitor entry
[0x000000001eb3e000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.leh.jvm.DeadLockDemo1.lambda$main$1(DeadLockDemo1.java:47)
        - waiting to lock <0x000000076b390f08> (a java.lang.Object)
        - locked <0x000000076b390f18> (a java.lang.Object)
        at com.leh.jvm.DeadLockDemo1$$Lambda$2/1831932724.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:748)

"thread1" #11 prio=5 os_prio=0 tid=0x000000001e099000 nid=0x3c040 waiting for monitor entry
[0x000000001ea3f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.leh.jvm.DeadLockDemo1.lambda$main$0(DeadLockDemo1.java:29)
        - waiting to lock <0x000000076b390f18> (a java.lang.Object)
        - locked <0x000000076b390f08> (a java.lang.Object)
        at com.leh.jvm.DeadLockDemo1$$Lambda$1/990368553.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:748)

"Service Thread" #10 daemon prio=9 os_prio=0 tid=0x000000001de01000 nid=0xd128 runnable
[0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"C1 CompilerThread2" #9 daemon prio=9 os_prio=2 tid=0x000000001ddf9800 nid=0x3cbcc waiting
on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"C2 CompilerThread1" #8 daemon prio=9 os_prio=2 tid=0x000000001dd99000 nid=0x3cf20 waiting
on condition [0x0000000000000000]
```

```
   java.lang.Thread.State: RUNNABLE

"C2 CompilerThread0" #7 daemon prio=9 os_prio=2 tid=0x000000001dd95800 nid=0x3e414 waiting
on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Monitor Ctrl-Break" #6 daemon prio=5 os_prio=0 tid=0x000000001dd92000 nid=0x2fb4c runnable
[0x000000001e43e000]
   java.lang.Thread.State: RUNNABLE
        at java.net.SocketInputStream.socketRead0(Native Method)
        at java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
        at java.net.SocketInputStream.read(SocketInputStream.java:171)
        at java.net.SocketInputStream.read(SocketInputStream.java:141)
        at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:284)
        at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:326)
        at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)
        - locked <0x000000076b4cf0d8> (a java.io.InputStreamReader)
        at java.io.InputStreamReader.read(InputStreamReader.java:184)
        at java.io.BufferedReader.fill(BufferedReader.java:161)
        at java.io.BufferedReader.readLine(BufferedReader.java:324)
        - locked <0x000000076b4cf0d8> (a java.io.InputStreamReader)
        at java.io.BufferedReader.readLine(BufferedReader.java:389)
        at com.intellij.rt.execution.application.AppMainV2$1.run(AppMainV2.java:64)

"Attach Listener" #5 daemon prio=5 os_prio=2 tid=0x000000001c9e0000 nid=0x2cc50 waiting on
condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Signal Dispatcher" #4 daemon prio=9 os_prio=2 tid=0x000000001dd50800 nid=0x27b88 runnable
[0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Finalizer" #3 daemon prio=8 os_prio=1 tid=0x00000000032e9000 nid=0x3ef34 in Object.wait()
[0x000000001dd3e000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x000000076b208ed0> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:143)
        - locked <0x000000076b208ed0> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:164)
        at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:212)

"Reference Handler" #2 daemon prio=10 os_prio=2 tid=0x00000000032e3000 nid=0x11e08 in
Object.wait() [0x000000001dc3f000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x000000076b206bf8> (a java.lang.ref.Reference$Lock)
        at java.lang.Object.wait(Object.java:502)
        at java.lang.ref.Reference.tryHandlePending(Reference.java:191)
        - locked <0x000000076b206bf8> (a java.lang.ref.Reference$Lock)
        at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:153)

"VM Thread" os_prio=2 tid=0x000000001c997000 nid=0x1cf74 runnable
```

```
"GC task thread#0 (ParallelGC)" os_prio=0 tid=0x0000000003208800 nid=0x13e10 runnable

"GC task thread#1 (ParallelGC)" os_prio=0 tid=0x000000000320a800 nid=0x4524 runnable

"GC task thread#2 (ParallelGC)" os_prio=0 tid=0x000000000320c000 nid=0x2604 runnable

"GC task thread#3 (ParallelGC)" os_prio=0 tid=0x000000000320e800 nid=0x5860 runnable

"VM Periodic Task Thread" os_prio=2 tid=0x000000001de5d800 nid=0x2d1a8 waiting on condition

JNI global references: 317


Found one Java-level deadlock:
=============================
"thread2":
  waiting to lock monitor 0x000000001c99bb08 (object 0x000000076b390f08, a
java.lang.Object),
  which is held by "thread1"
"thread1":
  waiting to lock monitor 0x000000001c99e398 (object 0x000000076b390f18, a
java.lang.Object),
  which is held by "thread2"

Java stack information for the threads listed above:
===================================================
"thread2":
        at com.leh.jvm.DeadLockDemo1.lambda$main$1(DeadLockDemo1.java:47)
        - waiting to lock <0x000000076b390f08> (a java.lang.Object)
        - locked <0x000000076b390f18> (a java.lang.Object)
        at com.leh.jvm.DeadLockDemo1$$Lambda$2/1831932724.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:748)
"thread1":
        at com.leh.jvm.DeadLockDemo1.lambda$main$0(DeadLockDemo1.java:29)
        - waiting to lock <0x000000076b390f18> (a java.lang.Object)
        - locked <0x000000076b390f08> (a java.lang.Object)
        at com.leh.jvm.DeadLockDemo1$$Lambda$1/990368553.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:748)

Found 1 deadlock.
```

```
com.leh.lock.MyLock object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                                               VALUE
      0     4        (object header)     -->对象头                           01 00 00 00
(00000001 00000000 00000000 00000000) (1)
      4     4        (object header)     -->对象头                           00 00 00 00
(00000000 00000000 00000000 00000000) (0)
      8     4        (object header)     -->对象头                           43 c1 00 f8
(01000011 11000001 00000000 11111000) (-134168253)
     12     4        (loss due to the next object alignment)  -->对齐字节  (对象头 4 + 4 + 4
12字节 不是8的倍数 ，需补足16字节，于是补4个字节)
Instance size: 16 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```

00000001 00000000 00000000 00000000

00000000 00000000 00000000 00000000

01000011 11000001 00000000 11111000

12 * 8 = 96bit

ObjcetHeader = 96bit

32位操作系统：

普通对象头{

Mark Word (4byte = 32bit)

klass pointer/Class Metadata Address

}

数组对象头{

Mark Word (4byte = 32bit)

klass pointer/Class Metadata Address

lenth

}

文档地址：

`http://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html`

object header

```
Common structure at the beginning of every GC-managed heap object. (Every oop points to an
object header.) Includes fundamental information about the heap object's layout, type, GC
state, synchronization state, and identity hash code. Consists of two words. In arrays it
is immediately followed by a length field. Note that both Java objects and VM-internal
objects have a common object header format.
```

翻译：

每个gc管理的堆对象开头的公共结构。(每个oop都指向一个对象头。)包括堆对象的布局、类型、GC状态、同步状态和标识哈希码的基本信息。由两个词（属性）组成。在数组中，它后面紧跟着一个长度字段。

注意，Java对象和vm内部对象都有一个通用的对象头格式

klass pointer

The second word of every object header. Points to another object (a metaobject) which describes the layout and behavior of the original object. For Java objects, the "klass" contains a C++ style "vtable".

mark word

The first word of every object header. Usually a set of bitfields including synchronization state and identity hash code. May also be a pointer (with characteristic low bit encoding) to synchronization related information. During GC, may contain GC state bits.

64位操作系统：

object header { 可以理解为 有一个专门描述对象头的文件 XXX.java 有两个属性

Mark Word (8byte = 64bit) ---------- MarkOop.cpp文件

klass pointer /Class Metadata Address （ jvm未开启指针压缩时占64bit, 若jvm开启指针压缩 则32bit ）

}

`markOop.cpp`

```
//  64 bits:
//  --------
//  unused:25 hash:31 -->| unused:1   age:4    biased_lock:1 lock:2 (normal object)
//  JavaThread*:54 epoch:2 unused:1   age:4    biased_lock:1 lock:2 (biased object)
//  PromotedObject*:61 --------------------->| promo_bits:3 ----->| (CMS promoted object)
//  size:64 ----------------------------------------------------->| (CMS free block)
//
//  unused:25 hash:31 -->| cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && normal
object)
//  JavaThread*:54 epoch:2 cms_free:1 age:4    biased_lock:1 lock:2 (COOPs && biased
object)
//  narrowOop:32 unused:24 cms_free:1 unused:4 promo_bits:3 ----->| (COOPs && CMS promoted
object)
//  unused:21 size:35 -->| cms_free:1 unused:7 ------------------>| (COOPs && CMS free
block)
```

64个字节每一位表示的意思会随着对象状态的变化而表示不同的意思 ，从而节省jvm的空间

Hotspot 与 jvm的关系？ openjdk又是什么?

jvm ----------概念/标准 ----- 女朋友

hotspot ------sun-----产品/实现 ----- 冰冰

openjdk项目（代码） ------即hotspot源码 ---C++开发的

java -version == > java.exe -version

java.exe --------openjdk编译之后的

war包 -------project 不包含源码 而是class文件

下载的jdk是openjdk编译后的文件 而不包含源码

学并发，需要去读懂 jdk的源码 c++ 的代码

jvm规范 ----openjdk文档

其他产品：

j9 ---------- IBM

JR

taobaoVM

java 当中 那某个东西计算时不能拿位数计算，但可以拿字节

比如 31位 = 3byte + 7位