深入理解JVM（三）—配置参数

1. **参数类别**

JVM配置参数分为三类参数：

1、跟踪参数

2、堆分配参数

3、栈分配参数

这三类参数分别用于跟踪监控JVM状态，分配堆内存以及分配栈内存。

1. **跟踪参数**

跟踪参数用于跟踪监控JVM，往往被开发人员用于JVM调优以及故障排查。

1. 当发生GC时，打印GC简要信息

使用-XX:+PrintGC或-verbose:gc参数

这两个配置参数效果是一样的，都是在发生GC时打印出简要的信息，例如执行代码：

1: public static void main(String[] args) 2: { 3: byte[] bytes =null; 4: for(int i=0;i<100;i++){ 5: bytes = new byte[1 \* 1024 \* 1024]; 6: } 7: }

这个程序连续创建了100个1M的数组对象，使用-XX:+PrintGC或-verbose:gc参数执行该程序，即可查看到GC情况：

1: [GC (Allocation Failure) 32686K->1648K(123904K), 0.0007230 secs] 2: [GC (Allocation Failure) 34034K->1600K(123904K), 0.0009652 secs] 3: [GC (Allocation Failure) 33980K->1632K(123904K), 0.0005306 secs]

我们可以看到程序执行了3次GC（minor GC），这三次GC都是新生代的GC，因为这个程序每次创建新的数组对象，都会把新的对象赋给bytes变量，而老的对象没有任意对象引用它，老对对象会变的不可达，这些不可达的对象在新生代minor GC时候被回收掉。

32686K表示回收前，对象占用空间。1648K表示回收后，对象占用空间。123904K表示还有多少空间可用。0.0007230 secs表示这次垃圾回收花的时间。

1. 打印GC的详细信息以及堆使用详细信息

使用-XX:+PrintGCDetails参数

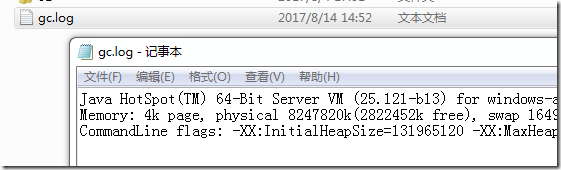
1: [GC (Allocation Failure) [PSYoungGen: 32686K->1656K(37888K)] 32686K->1664K(123904K), 0.0342788 secs] [Times: user=0.00 sys=0.00, real=0.03 secs] 2: [GC (Allocation Failure) [PSYoungGen: 34042K->1624K(70656K)] 34050K->1632K(156672K), 0.0013466 secs] [Times: user=0.00 sys=0.00, real=0.00 secs] 3: Heap 4: PSYoungGen total 70656K, used 43118K [0x00000000d6100000, 0x00000000dab00000, 0x0000000100000000) 5: eden space 65536K, 63% used [0x00000000d6100000,0x00000000d8985ac8,0x00000000da100000) 6: from space 5120K, 31% used [0x00000000da600000,0x00000000da796020,0x00000000dab00000) 7: to space 5120K, 0% used [0x00000000da100000,0x00000000da100000,0x00000000da600000) 8: ParOldGen total 86016K, used 8K [0x0000000082200000, 0x0000000087600000, 0x00000000d6100000) 9: object space 86016K, 0% used [0x0000000082200000,0x0000000082202000,0x0000000087600000) 10: Metaspace used 2669K, capacity 4486K, committed 4864K, reserved 1056768K 11: class space used 288K, capacity 386K, committed 512K, reserved 1048576K

我们看到除了打印GC信息之外，还显示了堆使用情况，堆分为新生代、老年代、元空间。注意这里没有永久区了，永久区在java8已经移除，原来放在永久区的常量、字符串静态变量都移到了元空间，并使用本地内存。

新生代当中又分为伊甸区（eden）和幸存区（from和to），从上面打印的内容可以看到新生代总大小为70656K，使用了43118K，细心的同学的可能会发现eden+from+to=65536K+5120K+5120K=75776 并不等于总大小70656K，这是为什么呢？这是因为新生代的垃圾回收算法是采用复制算法，简单的说就是在from和to之间来回复制（复制过程中再把不可达的对象回收掉），所以必须保证其中一个区是空的，这样才能有预留空间存放复制过来的数据，所以新生代的总大小其实等于eden+from（或to）=65536K+5120K=70656k。

1. 使用外部文件记录GC的日志

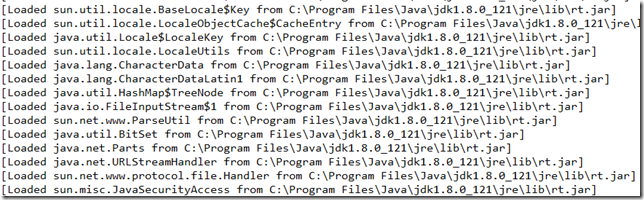
还有一个非常有用的参数，它可以把GC的日志记录到外部文件中，这在生产环境进行故障排查时尤为重要，当java程序出现OOM时，总希望看到当时垃圾回收的情况，通过这个参数就可以把GC的日志记录下来，便于排查问题，当然也可以做日常JVM监控。-Xloggc:log/gc.log



1. 监控类的加载

-XX:+TraceClassLoading

使用这个参数可以监控java程序加载的类：



1. **堆配置参数**

指定最大堆，最小堆：Xmx、Xms

这两个参数是我们最熟悉最常用的参数，可以用以下代码打印出目前内存使用的情况：

|  |
| --- |
| public static void main(String[] args) {  System.out.println("最大堆："+  Runtime.getRuntime().maxMemory()/1024/1024+"M");  System.out.println("空闲堆："+  Runtime.getRuntime().freeMemory()/1024/1024+"M");  System.out.println("总的堆："+  Runtime.getRuntime().totalMemory()/1024/1024+"M");  } |

最大堆也就是Xmx参数指定的大小，表示java程序最大能使用多少内存大小，如果超过这个大小，那么java程序会报：out of memory（OOM错误），空闲堆表示程序已经分配的内存大小减去已经使用的内存大小，而总的堆表示目前程序已经配置到多少内存大小，一般而言程序一启动，会按照-Xms5m先分配5M的空间，这时总的堆大小就是5M。

指定新生代内存大小:Xmn，例如我们指定-Xmx20m -Xms5m -Xmn2m -XX:+PrintGCDetails

新生代（eden+from+to）和老年代（不包含永久区）的比值：-XX:NewRatio

例如我们设置参数：-Xmx20m -Xms20m -XX:NewRatio=4 -XX:+PrintGCDetails（注意这里改参数为4表示新生代和老年代比值为1:4）

Eden区与Survivor区（from、to）的大小比值：-XX:SurvivorRatio（如设置为8,则两个Survivor区与一个Eden区的比值为2:8,一个Survivor区占整个年轻代的1/10）

例如设置参数-Xmx20m -Xms20m -Xmn8m -XX:SurvivorRatio=6 -XX:+PrintGCDetails

其他还有-XX:+HeapDumpOnOutOfMemoryError、-XX:+HeapDumpPath这两个参数可以实现在发生OOM异常时把堆栈信息打印到外部文件。

1. **堆分配参数的总结**

根据实际事情调整新生代和幸存代的大小

官方推荐新生代占堆的3/8

幸存代占新生代的1/10

在OOM时，记得Dump出堆，确保可以排查现场问题

1. **永久区分配参数**

-XX:PermSize -XX:MaxPermSize

用于设置永久区的初始空间和最大空间，他们表示一个系统可以容纳多少个类型，一般空间比较小。在java1.8以后，永久区被移到了元数据区，使用本地内存，所以这两个参数也不建议再使用。

1. **栈大小分配参数**

栈大小参数为-Xss，通常只有几百k，决定了函数调用的深度，每个线程都有自己独立的栈空间。如果函数调用太深，超过了栈的大小，则会抛出java.lang.StackOverflowError，通常我们遇到这种错误，不是去调整-Xss参数，而是应该去调查函数调用太深的原理，是否使用递归，能不能保证递归出口等。

1. **来源**

http://www.cnblogs.com/leefreeman/p/7364030.html