作为高级开发，你懂这些 JVM 参数吗？

搜云库技术团队 昨天

前言

大家都知道，jvm在启动的时候，会执行默认的一些参数。一般情况下，这些设置的默认参数应对一些平常的项目也够用了。但是如果项目特别大了，需要增加一下堆内存的大小、或者是系统老是莫明的挂掉，想查看下gc日志来排查一下错误的原因，都需要咱们手动设置这些参数。

各个参数介绍

1.verbose:gc

表示，启动jvm的时候，输出jvm里面的gc信息。格式如下：

```
[Full GC 178K->99K(1984K),  0.0253877 secs]
```

解读： Full GC 就表示执行了一次Full GC的操作，178K 和99K 就表示执行GC前内存容量和执行GC后的内存容量。1984K就表示内存总容量。后面那个是执行本次GC所消耗的时间，单位是秒。

2.-XX:+printGC

这个打印的GC信息跟上个一样，就不做介绍了。

3.-XX:+PrintGCDetails

打印GC的详细信息。格式如下：

```
-Heap
- def new generation   total 13824K, used 11223K [0x27e80000, 0x28d80000, 0x28d80000)
-  eden space 12288K,  91% used [0x27e80000, 0x28975f20, 0x28a80000)
-  from space 1536K,   0% used [0x28a80000, 0x28a80000, 0x28c00000)
-  to   space 1536K,   0% used [0x28c00000, 0x28c00000, 0x28d80000)
- tenured generation   total 5120K, used 0K [0x28d80000, 0x29280000, 0x34680000)
-   the space 5120K,   0% used [0x28d80000, 0x28d80000, 0x28d80200, 0x29280000)
- compacting perm gen  total 12288K, used 142K [0x34680000, 0x35280000, 0x38680000)
-   the space 12288K,   1% used [0x34680000, 0x346a3a90, 0x346a3c00, 0x35280000)
-    ro space 10240K,  44% used [0x38680000, 0x38af73f0, 0x38af7400, 0x39080000)
-    rw space 12288K,  52% used [0x39080000, 0x396cdd28, 0x396cde00, 0x39c80000)
```

解读：new generation 就是堆内存里面的新生代。total的意思就是一共的，所以后面跟的就是新生代一共的内存大小。used也就是使用了多少内存大小。0x开头的那三个分别代表的是 底边界，当前边界，高边界。也就是新生代这片内存的起始点，当前使用到的地方和最大的内存地点。

eden space 这个通常被翻译成伊甸园区，是在新生代里面的，一些创建的对象都会先被放进这里。后面那个12288K就表示伊甸园区一共的内存大小，91% used，很明显，表示已经使用了百分之多少。后面的那个0x跟上一行的解释一样。

from space 和to space 是幸存者的两个区。也是属于新生代的。他两个区的大小必须是一样的。因为新生代的GC采用的是复制算法，每次只会用到一个幸存区，当一个幸存区满了的时候，把还是活的对象复制到另个幸存区，上个直接清空。这样做就不会产生内存碎片了。

tenured generation 就表示老年代。

compacting perm 表示永久代。**由于这两个的格式跟前面我介绍的那个几乎一样，我就不必介绍了。**

4.-XX:+PrintGCTimeStamps

打印GC发生的时间戳。格式如下：

```
289.556: [GC [PSYoungGen: 314113K->15937K(300928K)] 405513K->107901K(407680K), 0.0178568 s
293.271: [GC [PSYoungGen: 300865K->6577K(310720K)] 392829K->108873K(417472K), 0.0176464 se
```

解读：289.556表示从jvm启动到发生垃圾回收所经历的的时间。GC表示这是新生代GC（Minor GC）。PSYoungGen表示新生代使用的是多线程垃圾回收器Parallel Scavenge。314113K->15937K(300928K)]这个跟上面那个GC格式一样，只不过，这个是表示的是新生代，幸存者区。后面那个是整个堆的大小，GC前和GC后的情况。Times这个显而易见，代表GC的所消耗的时间，用户垃圾回收的时间和系统消耗的时间和最终真实的消耗时间。

5.-X:loggc:log/gc.log

这个就表示，指定输出gc.log的文件位置。（我这里写的log/gc.log就表示在当前log的目录里，把GC日志写到叫gc.log的文件里。）

6.-XX:+PrintHeapAtGC

表示每次GC后，都打印堆的信息。（这个打印的基本格式跟上面第二条的基本类似，我也就不比多说了。）

7.-XX:+TraceClassLoading

监控类的加载。格式如下：

```
•[Loaded java.lang.Object from shared objects file]
•[Loaded java.io.Serializable from shared objects file]
•[Loaded java.lang.Comparable from shared objects file]
•[Loaded java.lang.CharSequence from shared objects file]
•[Loaded java.lang.String from shared objects file]
•[Loaded java.lang.reflect.GenericDeclaration from shared objects file]
•[Loaded java.lang.reflect.Type from shared objects file]
```

使用这个参数就能很清楚的看到那些类被加载的情况了。

8.-XX:+PrintClassHistogram

跟踪参数。这个按下Ctrl+Break后，就会打印一下信息：

```
num     #instances         #bytes  class name
----------------------------------------------
1:        890617      470266000  [B
2:        890643       21375432  java.util.HashMap$Node
3:        890608       14249728  java.lang.Long
4:            13        8389712  [Ljava.util.HashMap$Node;
5:          2062         371680  [C
6:           463          41904  java.lang.Class
```

–分别显示：序号、实例数量、总大小、类型。

这里面那个类型，B和C的其实就是byte和char类型。

9.-Xmx -Xms

这个就表示设置堆内存的最大值和最小值。这个设置了最大值和最小值后，jvm启动后，并不会直接让堆内存就扩大到指定的最大数值。而是会先开辟指定的最小堆内存，如果经过数次GC后，还不能，满足程序的运行，才会逐渐的扩容堆的大小，但也不是直接扩大到最大内存。

10.-Xmn

设置新生代的内存大小。

11.-XX:NewRatio

新生代和老年代的比例。比如：1：4，就是新生代占五分之一。

12.-XX:SurvivorRatio

设置两个Survivor区和eden区的比例。比如：2：8 ，就是一个Survivor区占十分之一。

13.-XX:+HeapDumpOnOutMemoryError

发生OOM时，导出堆的信息到文件。

14.-XX:+HeapDumpPath

表示，导出堆信息的文件路径。

15.-XX:OnOutOfMemoryError

当系统产生OOM时，执行一个指定的脚本，这个脚本可以是任意功能的。比如生成当前线程的dump文件，或者是发送邮件和重启系统。

16.-XX:PermSize -XX:MaxPermSize

设置永久区的内存大小和最大值。永久区内存用光也会导致OOM的发生。

17.-Xss

设置栈的大小。栈都是每个线程独有一个，所有一般都是几百k的大小。

总结

以上就是我整理的一些jvm设置的参数，当然不止这些。我这只是介绍了些常用的参数。希望能够帮到大家，由于能力有限，如有错误的地方敬请谅解。

来源：https://dwz.cn/ERri3dmC

近期技术热文

1、 试试 IDEA 解决 Maven 依赖冲突的高能神器！

2、 天真！这简历一看就是包装过的！

3、 用了这么久 IDEA ，有个牛逼功能叫后缀补全！

4、 count(1)、count(*) 与 count(列名) 的执行区别

5、 Java 里的 for (;;) 与 while (true)，哪个更快？

6、 SpringBoot 并发登录人数控制，附踢人功能


搜云库技术团队
网站：https://tech.souyunku.com
技术、架构、资料、工作 、内推
专注于分享最有价值的互联网技术干货文章
搜云库
技术团队