译者：不悔 QQ：845486067

原文：https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html

内存泄漏的常见现象就是java.lang.OutOfMemoryError异常。这个异常常见抛出的情况为：当需要分配对象的时候，堆空间不足。在这种情况下，垃圾收集器无法获取一个有效的空间来容纳一个新的对象，同时堆的大小也无法进行扩展。

另外，这种错误也有可能发生在native memory(操作系统内存)没有足够的空间装载一个java class的情况.还有一个比较少见的情况，当垃圾收集器耗费了大量的时间，但是仅仅只有很少的内存被释放。

 当一个 java.lang.OutOfMemoryError异常被抛出，相应的堆栈信息也会打印出来。

 java.lang.OutOfMemoryError也可能在native library code请求分配空间时，空间不足的情况下发生（例如，交换空间不足（swap space））

 诊断一个java.lang.OutOfMemoryError的异常最先要做的是确定异常产生的原因，到底是因为java堆满了还是因为native堆满了？

1. **Exception in thread thread\_name: java.lang.OutOfMemoryError: Java heap space**
   1. 造成的原因：java无法给新的对象分配空间。这种错误不一定是内存泄漏造成的。
      1. 可能是配置原因：当应用被设置了一个特定的堆容量（或者是默认的容量），但是这个容量又不足以支撑这个应用。
      2. 可能是对象的引用一直未释放，这使它在垃圾回收的时候无法被清理。
      3. 过度使用finalizers。当一个类拥有一个finalize 方法，GC的时候该类型的对象不会被垃圾回收器清理掉。相反的是，垃圾回收完毕后，这个对象会进入队列等待被回收，这个回收操作会发生在后面不确定的时间。在Oracle Sun的实现中，回收是由一个守护线程来执行的，该守护线程来处理队列中的的对象 。当回收线程的处理速度跟不上对象入队的速度的时候，就可能会导致堆内存被占满，从而抛出异常。还有一种情况是，当应用创建一个高优先级的线程（注：守护线程的优先级低），这会导致队列中对象增长的速度大于守护线程处理的速度
2. **Exception in thread thread\_name: java.lang.OutOfMemoryError: GC Overhead limit exceeded（GC 开销超出限额了）**
   1. 造成的原因：当完成一次垃圾收集，如果程序花费了98%的时间在垃圾回收上，但是回收的堆空间小于2%，并且已经做过5次连续垃圾回收的操作，然后这个异常将会被抛出。这个异常通常情况下抛出的原因是：存活的数据几乎占满了java堆的空间导致空闲的空间太小。
   2. 解决方法：增大堆的空间。该异常可以通过命令行进行关闭： -XX:-UseGCOverheadLimit（注：关闭这个参数仅仅是不抛出这种异常，而不是不抛异常）
3. **Exception in thread thread\_name: java.lang.OutOfMemoryError: Requested array size exceeds VM limit （请求分配的数组大小超过vm的限制）**
   1. 例如：当应用需要分配一个512M的数组，但是最大堆内存只有256M
   2. 解决方法
      1. 扩大堆内存
      2. 可能是程序有bug，要检查请求分配的数组大小是否合理
4. **Exception in thread thread\_name: java.lang.OutOfMemoryError: Metaspace**
   1. 造成的原因：metaspace中class metadata相关的空间被耗尽。metaspace中class metadata的空间受限于MaxMetaSpaceSize。当class metadata所需要的内存超过MaxMetaSpaceSize,该异常会抛出
   2. 解决方法：
      1. 增大MaxMetaSpaceSize
      2. 由于MetaSpace和java堆都是分配在相同的地址空间中（注：无论是堆还是非堆，大家都是使用的同一个物理机器上的内存），当减少Java堆的容量的时候，MetaSpace将获得更多的使用空间。如果Java堆有多余的空间，这是一个折中的方法。
5. **Exception in thread thread\_name: java.lang.OutOfMemoryError: request size bytes for reason. Out of swap space?**
   1. 造成的原因：native内存分配失败或者接近耗尽，一般是发起内存分配的模块抛出这个异常。
   2. 解决方法：当该异常信息抛出的时候，VM会调用fatal error（致命错误）处理机制（生成一个包含线程、进程、系统信息的日志文件）。在native内存耗尽的情况下，日志中的堆内存以及内存映射信息（memory map）可能是有用的。
6. **Exception in thread thread\_name: java.lang.OutOfMemoryError: Compressed class space**
   1. 造成的原因：在64位平台上一个class metadata 的指针可以用32位来表示（通过UseCompressedOops）.它是通过UseCompressedClassPointers命令来控制的。当UseCompressedClassPointers使用时，classmetadata的可用内存将等于CompressedClassSpaceSize。当UseCompressedClassPointers所需要的空间超过了CompressedClassSpaceSize，这个异常就会被抛出。
   2. 解决方法：增大CompressedClassSpaceSize，关闭UseCompressedClassPointers。备注：CompressedClassSpaceSize的是有一个限定范围的。比如说 -XX：CompressedClassSpaceSize=4g,超出了限制的范围，将会返回一个 CompressedClassSpaceSize of 4294967296 is invalid; must be between 1048576 and 3221225472（CompressedClassSpaceSize 的参数4294967296 是无效的；必须在1048576和3221225472之间（换算一下是1M到3G之间））
   3. 备注：有多种类型的class metadata，包括kclass metadata 和其他 metadata。只有klass metadata 是存放在CompressedClassSpaceSize限定的空间里。其他metadata存放在Metaspace
7. **Exception in thread thread\_name: java.lang.OutOfMemoryError: reason stack\_trace\_with\_native\_method**
   1. 造成的原因：当错误堆栈的栈顶是一个native方法，这表明native方法内存分配失败。这与其他错误信息不同的是，它是在JNI或者native方法中被检测出的这个异常，而不是在JVM代码层面。
   2. 解决方法：你需要通过操作系统的一些本地工具来做进一步诊断