**Java 常见的几种 OOM**

**1、StackOverflowError（栈空间溢出）**

public class StackOverflowErrorDemo {

public static void main(String[] args) {

main(args); // Exception in thread "main" java.lang.StackOverflowError

}

}

上面这种 OOM 比较好理解，在 main 方法中循环调用 main 方法，循环产生的大量形参都会在栈空间进行创建，当超过栈空间的大小，就会导致栈空间溢出，发生 OOM。

﻿

**2、Java Heap Space（堆空间溢出）**

public class JavaHeapSpaceDemo {

public static void main(String[] args) {

// 我配置了虚拟机参数 -Xms10m -Xmx10m 初始化堆内存和最大堆内存都是 10m

byte[] b = new byte[20 \* 1024 \* 1024]; // 这里 new 了 20m 的字节数组

// Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

}

}

上面的这个 OOM 也比较好理解，我给 JVM 设置的初始化堆内存和最大堆内存大小都是 10M，然后我在 main 方法中创建一个 20M 大小的字节数组，很显然一下子就超过堆内存大小了，直接发生 OOM。

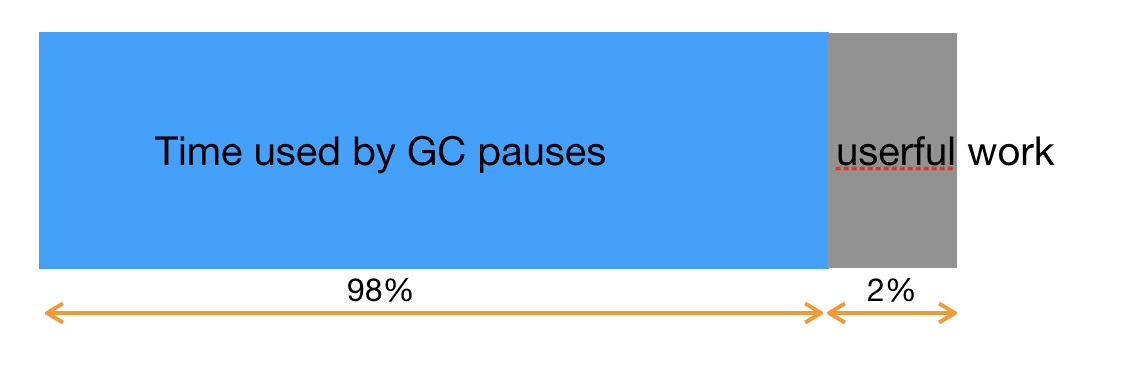
﻿

**3、GC overhead limit exceeded（GC 回收时间过长）**

GC 回收时间过长时会抛出 OutOfMemoryError 。过长的定义是，**超过 98% 的时间用来做 GC** ，并且**回收了不到 2% 的堆内存**，连续多次 GC 都只回收了不到 2% 的极端情况下才会抛出。假如不抛出 GC overhead limit 错误会发生什么情况呢？

﻿

那就是 GC 清理的这么点内存很快会再次填满，迫使 GC 再次执行，这样就形成恶性循环，CPU 使用率一直是 100%，而 GC 却没有任何成果。



﻿

public class GCOverheadDemo {

/\*\*

\* JVM 参数配置： -Xms10m -Xmx10m -XX:+PrintGCDetails -XX:MaxDirectMemorySize=5m

\*/

public static void main(String[] args) {

int i = 0;

List<String> list = new ArrayList<String>();

try {

while (true) {

list.add(String.valueOf(++i).intern());

}

} catch (Throwable e) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*" + i);

e.printStackTrace();

throw e;

}

}

}

// 执行结果,进行了很多次 GC，最后抛出下面错误

// Exception in thread "main" java.lang.OutOfMemoryError: GC overhead limit exceeded

// 随便找一条 GC 的日志来看：

// [Full GC (Ergonomics) [PSYoungGen: 2047K->2047K(2560K)] [ParOldGen: 6998K->6998K(7168K)] 9046K->9046K(9728K), [Metaspace: 3314K->3314K(1056768K)], 0.0266330 secs] [Times: user=0.05 sys=0.00, real=0.03 secs]

// 发现一直在做 GC ，但是 GC 效果却不明显。连续多次 GC 都回收不了多少内存，只有抛出 error 否则，恶性循环。

﻿

**4、Direct buffer memory（本机直接内存溢出）**

1️⃣ 写 NIO 程序经常使用 ByteBuffer 来读取或者写入数据，这是一种基于通道（Channel）与缓冲区（Buffer）的 I/O 方式。它可以使用 Native 函数库直接分配堆外内存，然后通过一个存在在 Java 堆里面的 DirectByteBuffer 对象作为这块内存的引用进行操作。这样能在一些场景中显著提高性能，因为避免了在 Java 堆和 Native 堆中来回复制数据。

﻿

2️⃣ ByteBuffer.allocate(capability) 第一种方式是分配 JVM 堆内存，属于 GC 管辖范围，由于需要拷贝所以速度相对较慢。

﻿

3️⃣ ByteBuffer.allocateDirect(capability) 第二种方式是分配 OS 本地内存，不属于 GC 管辖范围，由于不需要内存拷贝，所以速度相对较快。

﻿

4️⃣ 但如果不断分配本地内存，堆内存很少使用，那么 JVM 就不需要执行 GC ，DirectByteBuffer 对象们就不会被回收，这时候堆内存充足，但本地内存可能已经用光了，再次尝试分配本地内存就会出现 OutOfMemory ，程序直接崩溃。

public class DirectBufferMemoryDemo {

public static void main(String[] args) {

// 如果什么都不配置 JVM 内存，大概是本地内存的 1/4

System.out.println("配置的 maxDirectMemory" + (sun.misc.VM.maxDirectMemory()/ (double)1024 / 1024) + "MB");

// 这里调用的是 jdk 包中 rt.jar 包中的方法 sun.misc.VM.maxDirectMemory()

try {

Thread.sleep(3000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// -Xms10m -Xmx10m -XX:MaxDirectMemorySize=5m （把系统堆内存设置成 5MB）

// 配置 5MB 实际使用 6MB

ByteBuffer buffer = ByteBuffer.allocateDirect(6 \* 1024 \* 1024);

}

// 直接内存我给它调到了 5M ，但是分配了 6M 内存，内存用光了，就报 OutOfMemory 了

// 最后运行结果 Exception in thread "main" java.lang.OutOfMemoryError: Direct buffer memory

}

**Tips:**

1、可以使用 -XX:MaxDirectMemorySize 来指定本机直接内存的大小

2、在 NIO 程序中，使用 ByteBuffer.allocateDirect(capability) 分配的是直接内存，可能会导致堆内存溢出。

﻿

**5、unable to create new native thread（不能创建一个本地线程）**

高并发请求服务器时，经常出现如下异常：java.lang.OutOfMemoryError:unable to create new native thread

﻿

准确的说 native thread 异常与对应的平台有关。

﻿

**导致原因：**

* 你的应用创建了太多线程，一个应用进程创建多个线程，超过系统承载极限
* 你的服务器并不允许你的应用程序创建这么多线程，linux 系统默认允许单个进程可以创建的线程数是 1024 个，你的应用创建超过这个数量，就会报 java.lang.OutOfMemoryError:unable to create new native thread

﻿

**解决办法：**

* 想办法降低你应用程序创建线程的数量，分析应用是否真的需要创建这么多线程，如果不是，修改代码将线程数降到最低
* 对于有的应用，确实需要创建很多的线程，远超过 linux 系统的默认 1024 个线程的限制，可以通过修改 linux 服务器配置，扩大 linux 默认限制

public class UnableCreateNewThreadDemo {

public static void main(String[] args) {

for (int i = 1; ;i++) { // 无限 for 循环

System.out.println("-------i =" + i);

new Thread(new Runnable() {

@Override

public void run() {

try {

Thread.sleep(Integer.MAX\_VALUE);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}, "" + i).start();

}

}

}

// 这样就会有无数个线程被创建，被创建之后 sleep 在那，不会停止，一旦达到 linux 系统默认线程限制，

//就会报 Exception in thread "main" java.lang.OutOfMemoryError: unable to create new native thread

﻿

**通过扩大服务器线程限制解决方法：**

vim /etc/security/limits.d/90-nproc.conf

\* soft nproc 1024

root soft nproc unlimited

heping soft nproc 3000

# 假如说想要用 heping 这个用户运行，然后希望他生成的线程多一些，可以编辑这个配置文件，

#在下面加一行，然后把数字调大点