JVM OOM异常会导致JVM退出吗?

出处： https://mp.weixin.qq.com/s/8j8YTcr2qhVActLGzOqe7Q

https://blog.csdn.net/h2604396739/article/details/91441248

先分析一道面试题

JVM 堆内存溢出后，其他线程是否可继续工作?

答：这道题其实很有难度，涉及的知识点有jvm内存分配、作用域、gc等，不是简单的是与否的问题。

由于题目中给出的OOM，java中OOM又分很多类型；比如：堆溢出（“java.lang.OutOfMemoryError: Java heap space”）、永久带溢出（“java.lang.OutOfMemoryError:Permgen space”）、不能创建线程（“java.lang.OutOfMemoryError:Unable to create new native thread”）等很多种情况。

本文主要是分析堆溢出对应用带来的影响。

先说一下答案，答案是还能运行 。

代码如下：

```
public class JvmThread {

    public static void main(String[] args) {
        new Thread(() -> {
            List<byte[]> list = new ArrayList<byte[]>();
            while (true) {
                System.out.println(new Date().toString() + Thread.currentThread() + "==");
                byte[] b = new byte[1024 * 1024 * 1];
                list.add(b);
                try {
                    Thread.sleep(1000);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();

        // 线程二
        new Thread(() -> {
            while (true) {
                System.out.println(new Date().toString() + Thread.currentThread() + "==");
                try {
                    Thread.sleep(1000);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
}
```

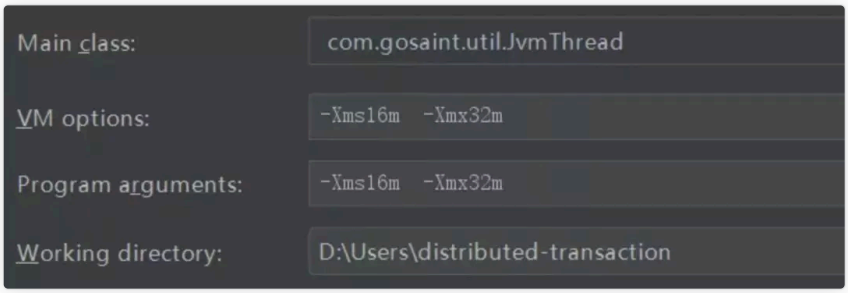
结果展示：

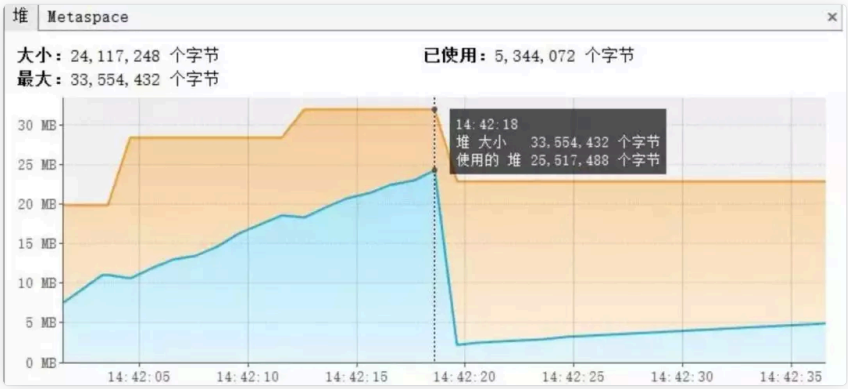
```
Wed Nov 07 14:42:18 CST 2018Thread[Thread-1,5,main]==
Wed Nov 07 14:42:18 CST 2018Thread[Thread-0,5,main]==
```

主题色彩

```
at com.gosaint.util.JvmThread.lambda$main$0(JvmThread.java:21)
at com.gosaint.util.JvmThread$$Lambda$1/521645586.run(Unknown Source)
at java.lang.Thread.run(Thread.java:748)
Wed Nov 07 14:42:20 CST 2018Thread[Thread-1,5,main]==
Wed Nov 07 14:42:21 CST 2018Thread[Thread-1,5,main]==
Wed Nov 07 14:42:22 CST 2018Thread[Thread-1,5,main]==
```

JVM启动参数设置：





上图是JVM堆空间的变化。我们仔细观察一下在14:42:05~14:42:25之间曲线变化，你会发现使用堆的数量，突然间急剧下滑！这代表这一点，当一个线程抛出OOM异常后，它所占据的内存资源会全部被释放掉，从而不会影响其他线程的运行！

讲到这里大家应该懂了，此题的答案为一个线程溢出后，进程里的其他线程还能照常运行。注意了，这个例子我只演示了堆溢出的情况。如果是栈溢出，结论也是一样的，大家可自行通过代码测试。

总结：其实发生OOM的线程一般情况下会死亡，也就是会被终结掉，该线程持有的对象占用的heap都会被gc了，释放内存。因为发生OOM之前要进行gc，就算其他线程能够正常工作，也会因为频繁gc产生较大的影响。

一、问题来源

一次生产事故，由于一次性从数据库查询过多数据导致***线程*** OOM：Java heap space 异常（千万级表，JVM堆内存2G），但是在线程OOM发生时，java进程却没有立即挂掉。

不符合所谓发生OOM，程序就会挂的“预期”，因此进行深入了解。

二、OOM与异常

说到底OutOfMemoryError也只是一个java中的异常而已，属于Error一系非检查异常：


```
Object
    Throwable
```

堆内存不够与异常的关系

线程发生OOM Java heap space，首先是堆空间不够了，然后再由jvm在申请分配空间的方法调用上抛出OOM异常。

对于线程，它会像处理普通异常一样，处理OutOfMemoryError。

三、异常与线程

线程是资源调度的基本单位，Java在设计线程时充分考虑了线程的独立性。在异常方面，保持了线程异常的独立性，在线程执行中发生的异常，都由线程自身解决，不会抛出到执行它的线程。

在线程的实现上，也保证了这种独立性。

检查异常不可抛

线程的实现java.lang.Runnable实现了java.lang.Runnable接口，线程通过其run方法运行，方法签名如下：

```
public abstract void run();
```

由java语法保证了实现Runnable接口或者继承Thread类的子类，其run方法也不能声明抛出任何检查异常（checked exception）。因此在线程方法执行中发生的任何检查异常，必须在线程中处理。

默认异常处理器

除了检查异常，java中还有非检查异常（unchecked exception），这种异常无需显式声明也能沿着方法调用链向上抛出。线程对于这种未处理的异常，提供了默认异常处理器：

```
/**
* Dispatch an uncaught exception to the handler. This method is
* intended to be called only by the JVM. (将未被捕获的异常分发给处理器。这个方法只被JVM调用)
*/
private void dispatchUncaughtException(Throwable e) {
   getUncaughtExceptionHandler().uncaughtException(this, e);
}
```

Thread的 `init()` 方法线程至少有一个默认异常处理器，兜底的异常处理器是当前线程父线程的线程组 `ThreadGroup` ，可以看到线程组是有能力处理异常的：

```
public class ThreadGroup implements Thread.UncaughtExceptionHandler {}
```

线程通过这两种机制，保证内部发生的异常，在线程内解决，而不会抛出给启动线程的外部线程。

四、JVM退出条件

java虚拟机退出的条件是：虚拟机内不存在非守护线程。

线程发生未处理的异常（未处理异常由默认异常处理器处理）会导致线程结束，而与JVM的退出毫无关系。

OOM也是一种异常，它的发生也不会导致JVM退出。以下实例说明：

```
static class OOMObject {
}

// 为快速发生oom，设置堆大小; VM args: -Xms20m -Xmx20m
public static void main(String[] args) throws InterruptedException {
     new Thread(() -> {
         List<OOMObject> list = new ArrayList<>();
         while (true) {
             list.add(new OOMObject());
         }
```

```
        System.out.println(Thread.currentThread().getName() + " continuing...");
        Thread.sleep(1000L);
    }
}
```

线程抛出java.lang.OutOfMemoryError: Java heap space后，main线程依旧会循环打印main continuing...。
线程中发生OOM异常如此，发生其他异常也如此，不影响其他线程，也不会导致JVM退出。

五、OOM与JVM退出

OOM的发生表示了此刻JVM堆内存告罄，不能分配出更多的资源，或者gc回收效率不可观。一个线程的OOM，在一定程度的并发下，若此时其他线程也需要申请堆内存，那么其他线程也会因为申请不到内存而OOM，甚至连锁反应导致整个JVM的退出。

以上示例没有导致JVM退出的原因在于，线程通过往局部变量集合中不断加入对象，产生OOM。线程因异常退出后，集合中的对象由于引用不可达，会被gc，这样就有了足够的堆内存供其他线程使用。

若示例中的list是一个“全局”的类static变量，那么即使线程退出，内存也得不到释放。这时其他线程如果不断再申请堆内存资源，就会造成连锁反应导致JVM退出。

分类: JVM

« 上一篇: Linux系统下如何优雅地关闭Java进程?
» 下一篇: JVM 内存溢出详解（栈溢出，堆溢出，持久代溢出、无法创建本地线程）