**堆内内存：**

堆内内存 = 新生代+老年代+持久代

1. **堆内存**完全由**JVM负责分配和释放**，如果程序没有缺陷代码导致内存泄露，那么就不会遇到java.lang.OutOfMemoryError这个错误。

使用**堆外内存，就是为了能直接分配和释放内存**，提高效率。堆外内存意味着把内存对象分配在Java虚拟机的堆以外的内存，**这些内存直接受操作系统管理**（而不是虚拟机）。JDK5.0之后，**代码中能直接操作本地内存**的方式有2种：

1.使用未公开的Unsafe。

2.NIO包下**ByteBuffer。java.nio.DirectByteBuffer对象进行堆外内存的管理和使用，它会在对象创建的时候就分配堆外内存。**

**堆外内存的好处是：**

**1、可以扩展至更大的内存空间。比如超过1TB甚至比主存还大的空间。   
2、理论上能减少GC暂停时间。**  
3、可以在进程间共享**，减少JVM间的对象复制，**使得JVM的分割部署更容易实现。 。因为堆内在flush到远程时，会先复制到直接内存（非堆内存），然后在发送；而**堆外内存相当于省略掉了这个工作**。

4、它的**持久化存储可以支持快速重启**，同时还能够在测试环境中重现生产数据。

**堆外内存缺点：**

而福之祸所依，自然也有不好的一面：   
　　**1 堆外内存难以控制，如果内存泄漏，那么很难排查**  
　　2 **堆外内存相对来说，不适合存储很复杂的对象。一般简单的对象或者扁平化的比较适合。**

JVM可以使用的内存分外2种：**堆内存和堆外内存**.

    堆内存完全由JVM负责分配和释放，如果程序没有缺陷代码导致内存泄露，那么就不会遇到java.lang.OutOfMemoryError这个错误。

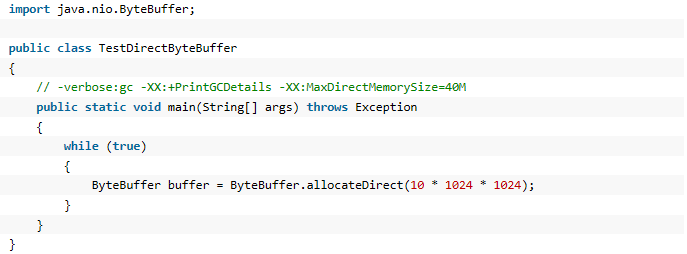
    使用堆外内存，就是为了能直接分配和释放内存，提高效率。JDK5.0之后，代码中能直接操作本地内存的方式有2种：使用未公开的Unsafe和NIO包下ByteBuffer。

    关于Unsafe对象的简介和获取方式,（不安全，不建议使用）

    使用ByteBuffer分配本地内存则非常简单，直接ByteBuffer.allocateDirect(10 \* 1024 \* 1024)即可。

    C语言的内存分配和释放函数malloc/free，必须要一一对应，否则就会出现内存泄露或者是野指针的非法访问。java中我们需要手动释放获取的堆外内存吗？

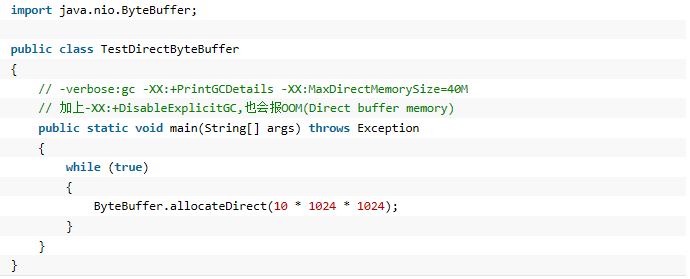
**我们一起来看看NIO中提供的ByteBuffer**

****

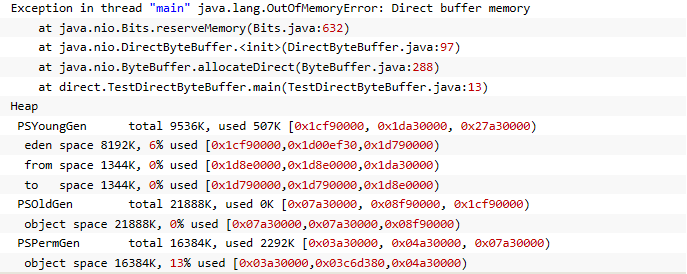
      我们将**最大堆外内存设置成40M，运行这段代码会发现：程序可以一直运行下去，**不会报OutOfMemoryError。如果使用了-verbose:gc -XX:+PrintGCDetails，会发现程序频繁的进行垃圾回收活动。那么**DirectByteBuffer究竟是如何释放堆外内存的？**

    我们修改下JVM的启动参数，重新运行之前的代码：

https://images2015.cnblogs.com/blog/37237/201601/37237-20160106221257403-45840365.png



      与之前的JVM启动参数相比，增加了-XX:+DisableExplicitGC，这个参数作用是禁止代码中显示调用GC。代码如何显示调用GC呢，通过System.gc()函数调用。如果加上了这个JVM启动参数，那么代码中调用System.gc()没有任何效果，相当于是没有这行代码一样。



      显然堆内存（包括新生代和老年代）内存很充足，但是堆外内存溢出了。也就是说NIO直接内存的回收，需要依赖于System.gc()。**如果我们的应用中使用了java nio中的direct memory，那么使用-XX:+DisableExplicitGC一定要小心，存在潜在的内存泄露风险**。

     我们知道java代码无法强制JVM何时进行垃圾回收，也就是说垃圾回收这个动作的触发，完全由JVM自己控制，它会挑选合适的时机回收堆内存中的无用java对象。

代码中显示调用System.gc()，只是建议JVM进行垃圾回收，但是到底会不会执行垃圾回收是不确定的，可能会进行垃圾回收，也可能不会。什么时候才是合适的时机呢？一般来说是，系统比较空闲的时候（比如JVM中活动的线程很少的时候），还有就是内存不足，不得不进行垃圾回收。

我们例子中的根本矛盾在于：**堆内存由JVM自己管理，堆外内存必须要由我们自己释放；堆内存的消耗速度远远小于堆外内存的消耗，但要命的是必须先释放堆内存中的对象，才能释放堆外内存，但是我们又不能强制JVM释放堆内存。**

**Direct Memory的回收机制：**Direct Memory是受GC控制的，例如ByteBuffer bb = ByteBuffer.allocateDirect(1024)，这段代码的执行会在堆外占用1k的内存，**Java堆内只会占用一个bb对象的指针引用的大小，堆外的这1k的空间只有当bb对象被回收时，才会被回收，这里会发现一个明显的不对称现象，就是堆外可能占用了很多，而堆内没占用多少，导致还没触发GC，那就很容易出现Direct Memory造成物理内存耗光。**

Direct ByteBuffer分配出去的内存其实也是由GC负责回收的，而不像Unsafe是完全自行管理的，Hotspot在GC时会扫描Direct ByteBuffer对象是否有引用，如没有则同时也会回收其占用的堆外内存。