# 介绍

对象的内存分配，从概念上讲，应该都是在堆上分配（而实际上也有可能经过即时编译后被拆散为标量类型并间接地在栈上分配）。在经典分代的设计下，新生对象通常会分配在新生代中，少数情况下（例如对象大小超过一定阈值）也可能会直接分配在老年代。对象分配的规则并不是固定的，《Java虚拟机规范》并未规定新对象的创建和存储细节，这取决于虚拟机当前使用的是哪一种垃圾收集器，以及虚拟机中与内存相关的参数的设定。

本次使用HotSpot虚拟机，以客户端模式运行。由于并未指定收集器组合，因此，本节验证的实际是使用Serial加Serial Old客户端默认收集器组合下的内存分配和回收的策略，这种配置和收集器组合也许是开发人员做研发时的默认组合（其实现在研发时很多也默认用服务端虚拟机了），但在生产环境中一般不会这样用。

# 对象优先在Eden分配

大多数情况下，对象在新生代Eden区中分配。当Eden区没有足够空间进行分配时，虚拟机将发起一次Minor GC。

HotSpot虚拟机提供了-XX：+PrintGCDetails这个收集器日志参数，告诉虚拟机在发生垃圾收集行为时打印内存回收日志，并且在进程退出的时候输出当前的内存各区域分配情况。在实际的问题排查中，收集器日志常会打印到文件后通过工具进行分析。

在代码的testAllocation()方法中，尝试分配三个2MB大小和一个4MB大小的对象，在运行时通过-Xms20M、-Xmx20M、-Xmn10M这三个参数限制了Java堆大小为20MB，不可扩展，其中10MB分配给新生代，剩下的10MB分配给老年代。-XX：Survivor-Ratio=8决定了新生代中Eden区与一个Survivor区的空间比例是8∶1，从输出的结果也清晰地看到“eden space 8192K、from space 1024K、to space 1024K”的信息，新生代总可用空间为9216KB（Eden区+1个Survivor区的总容量）。

|  |
| --- |
| private static final int \_1MB = 1024 \* 1024;  /\*\*  \* VM参数：-verbose:gc -Xms20M -Xmx20M -Xmn10M -XX:+PrintGCDetails -XX:SurvivorRatio=8  \*/  public static void testAllocation() {  byte[] allocation1, allocation2, allocation3, allocation4;  allocation1 = new byte[2 \* \_1MB];  allocation2 = new byte[2 \* \_1MB];  allocation3 = new byte[2 \* \_1MB];  allocation4 = new byte[4 \* \_1MB]; // 出现一次Minor GC  } |

运行结果：

|  |
| --- |
| [GC [DefNew: 6651K->148K(9216K), 0.0070106 secs] 6651K->6292K(19456K), 0.0070426 secs] [Times: user=0.00 sys=0.0  Heap  def new generation total 9216K, used 4326K [0x029d0000, 0x033d0000, 0x033d0000)  eden space 8192K, 51% used [0x029d0000, 0x02de4828, 0x031d0000)  from space 1024K, 14% used [0x032d0000, 0x032f5370, 0x033d0000)  to space 1024K, 0% used [0x031d0000, 0x031d0000, 0x032d0000)  tenured generation total 10240K, used 6144K [0x033d0000, 0x03dd0000, 0x03dd0000)  the space 10240K, 60% used [0x033d0000, 0x039d0030, 0x039d0200, 0x03dd0000)  compacting perm gen total 12288K, used 2114K [0x03dd0000, 0x049d0000, 0x07dd0000)  the space 12288K, 17% used [0x03dd0000, 0x03fe0998, 0x03fe0a00, 0x049d0000)  No shared spaces configured. |

执行testAllocation()中分配allocation4对象的语句时会发生一次Minor GC，这次回收的结果是新生代6651KB变为148KB，而总内存占用量则几乎没有减少（因为allocation1、2、3三个对象都是存活的，虚拟机几乎没有找到可回收的对象）。

产生这次垃圾收集的原因是为allocation4分配内存时，发现Eden已经被占用了6MB，剩余空间已不足以分配allocation4所需的4MB内存，因此发生Minor GC。垃圾收集期间虚拟机又发现已有的三个2MB大小的对象全部无法放入Survivor空间（Survivor空间只有1MB大小），所以只好通过分配担保机制提前转移到老年代去。

这次收集结束后，4MB的allocation4对象顺利分配在Eden中。因此程序执行完的结果是Eden占用4MB（被allocation4占用），Survivor空闲，老年代被占用6MB（被allocation1、2、3占用）。通过GC日志可以证实这一点。

# 大对象直接进入老年代

大对象就是指需要大量连续内存空间的Java对象，最典型的大对象便是那种很长的字符串，或者元素数量很庞大的数组，本节例子中的byte[]数组就是典型的大对象。大对象对虚拟机的内存分配来说就是一个不折不扣的坏消息，比遇到一个大对象更加坏的消息就是遇到一群“朝生夕灭”的“短命大对象”，我们写程序的时候应注意避免。

在Java虚拟机中要避免大对象的原因是，在分配空间时，它容易导致内存明明还有不少空间时就提前触发垃圾收集，以获取足够的连续空间才能安置好它们，而当复制对象时，大对象就意味着高额的内存复制开销。

HotSpot虚拟机提供了-XX：PretenureSizeThreshold参数，指定大于该设置值的对象直接在老年代分配，这样做的目的就是避免在Eden区及两个Survivor区之间来回复制，产生大量的内存复制操作。

|  |
| --- |
| private static final int \_1MB = 1024 \* 1024;  /\*\*  \* VM参数：-verbose:gc -Xms20M -Xmx20M -Xmn10M -XX:+PrintGCDetails -XX:SurvivorRatio=8  \* -XX:PretenureSizeThreshold=3145728  \*/  public static void testPretenureSizeThreshold() {  byte[] allocation;  allocation = new byte[4 \* \_1MB]; //直接分配在老年代中  } |

运行结果：

|  |
| --- |
| Heap  def new generation total 9216K, used 671K [0x029d0000, 0x033d0000, 0x033d0000)  eden space 8192K, 8% used [0x029d0000, 0x02a77e98, 0x031d0000)  from space 1024K, 0% used [0x031d0000, 0x031d0000, 0x032d0000)  to space 1024K, 0% used [0x032d0000, 0x032d0000, 0x033d0000)  tenured generation total 10240K, used 4096K [0x033d0000, 0x03dd0000, 0x03dd0000)  the space 10240K, 40% used [0x033d0000, 0x037d0010, 0x037d0200, 0x03dd0000)  compacting perm gen total 12288K, used 2107K [0x03dd0000, 0x049d0000, 0x07dd0000)  the space 12288K, 17% used [0x03dd0000, 0x03fdefd0, 0x03fdf000, 0x049d0000)  No shared spaces configured. |

执行代码中的testPretenureSizeThreshold()方法后，我们看到Eden空间几乎没有被使用，而老年代的10MB空间被使用了40%，也就是4MB的allocation对象直接就分配在老年代中，这是因为-XX：PretenureSizeThreshold被设置为3MB（就是3145728，这个参数不能与-Xmx之类的参数一样直接写3MB），因此超过3MB的对象都会直接在老年代进行分配。

注意　-XX:PretenureSizeThreshold参数只对Serial和ParNew两款新生代收集器有效，HotSpot的其他新生代收集器，如Parallel Scavenge并不支持这个参数。如果必须使用此参数进行调优，可考虑ParNew加CMS的收集器组合。

# 长期存活的对象将进入老年代