链滴

JVM (四) 堆内存模型及内存分配策略

作者：wky181

原文链接：https://ld246.com/article/1571622389511

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190530.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="JVM-堆内存模型">JVM 堆内存模型</h2>
JVM 的堆分为两部分，分别为新生代和老年代

新生代：
新生代又分为两个区。一个伊甸（Eden）区，一个幸存者（survivor）区，幸存者区又分为 so 和 s1
区，新生代采用的是复制算法，因为新生的对象很多，大部分都是朝生息死的，所以采用复制算法最
理，新生的对象都要先放在伊甸区，在 Eden 区满的时候，就要进行 Minor GC 了。将 Eden 区存活
对象复制到 so 或者 s1 区，然后清空 Eden 区。之后如果 Eden 区再满的时候，就把 Eden 和非空的
存者区里存活的对象放入另一个幸存者区，然后清空，so 和 s1 互相交替，如此循环。
参数控制 <code>-XX:SurvivorRatio</code> 用来设置新生代中 Eden 空间和 from/to 空间的比例
br>
默认 <code>-XX:SurvivorRatio=8</code> Eden:From:To 8:1:1
老年代
经过一定次数的 Minor GC 后，在新生代存活下来的对象会进入老年代，大对象也会直接进入老年代
老年代采用的是标记-清除法或者标记-整理法。因为老年代大部分都是存活时间比较长的对象。当老
代满的时候，就会进行 Full GC。其回收速度一般会比 Minor GC 慢十倍以上 。
参数控制 <code> -XX:MaxTenuringThreshold</code> 设置对象经过几次 GC 进入老年代，默认
5
参数控制 <code> -XX: PretenureSizeThreshold</code> 设置指定大小 如果超过这个大小对象直
进入老年代 (前提使用 ParNew 或者 Serial 收集器)。
参数控制 <code>-XX: NewRatio</code> 设置新生代和老年代的比例

<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2019/10/image-fd40504c.png?imageView2/2/interlace/1/format/jpg"></
>
<h2 id="JVM-常用参数">JVM 常用参数</h2>
<table>
<thead>
<tr>
<th>JVM 参数</th>
<th>说明</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:+PrintGC</td>
<td>使用这个参数，虚拟机启动后，只要遇到 GC 就会打印日志</td>
</tr>
<tr>
<td>-XX:MaxTenuringThreshold</td>
<td>设置对象经过几次 GC 进入老年代，默认 15</td>
</tr>
<tr>
<td>-XX:+PrintGCDetails</td>
<td>可以查看详细信息，包括各区情况</td>
</tr>
<tr>
<td>-Xms</td>
<td>设置 java 程序启动时初始堆大小</td>
</tr>
<tr>
<td>-Xmx</td>

<td>设置程序能获得的最大堆大小</td>
</tr>
<tr>
<td>-Xss</td>
<td>指定每个线程的最大深度</td>
</tr>
<tr>
<td>-Xmn</td>
<td>可以设置新生代的大小</td>
</tr>
<tr>
<td>-XX:SurvivorRatio</td>
<td>用来设置新生代中 Eden 空间和 from/to 空间的比例</td>
</tr>
<tr>
<td>-XX: NewRatio</td>
<td>设置新生代和老年代的比例</td>
</tr>
<tr>
<td>-XX:+PrintCommandLineFlags</td>
<td>将虚拟机显式和隐式的参数输出</td>
</tr>
<tr>
<td>-XX:-UseTLAB</td>
<td>禁止为本地线程分配缓冲</td>
</tr>
</tbody>
</table>
<h2 id="对象先进入伊甸区">对象先进入伊甸区</h2>
参数控制 <code>-XX:+PrintGC -Xms20m -Xmx20m -Xmn10m -XX:+PrintGCDetails -XX:+PintCommandLineFlags -XX:SurvivorRatio=8 -XX:-UseTLAB</code>
<pre><code class="highlight-chroma">public class Lecture02 {
    private static final int _1MB = 1024 * 1024;

    public static void main(String[] args) {
        byte[] byte1, byte2, byte3, byte4;
        byte1 = new byte[2 * _1MB];
        byte2 = new byte[2 * _1MB];
        byte3 = new byte[2 * _1MB];
        byte4 = new byte[3 * _1MB];
    }
}
</code></pre>
GC 日志如下：
<pre><code class="highlight-chroma">[GC (Allocation Failure) [DefNew: 8096K->670K(9216K), 0.0060545 secs] 8096K->68

4K(19456K), 0.0060974 secs] [Times: user=0.00 sys=0.02, real=0.01 secs]
Heap
 def new generation total 9216K, used 3746K [0x00000000fec00000, 0x00000000ff600000, 0x00000000ff600000

  eden space 8192K, 37% used [0x00000000fec00000, 0x00000000fef01168, 0x00000000ff400000)
  from space 1024K, 65% used [0x00000000ff500000, 0x00000000ff5a79e8, 0x00000000ff600000)
  to   space 1024K, 0% used [0x00000000ff400000, 0x00000000ff400000, 0x00000000ff500000)
 tenured generation total 10240K, used 6144K [0x00000000ff600000, 0x0000000100000000, 0x0000000100000000)
</code></pre>
在执行到最后一行代码的时候，此时 Eden 区已经放入了 3 个 2M 的对象，而 Eden 的最大空间大约是 8M。再最后一个 3M 的对象想进入 Eden 区时发现空间不足，所以发生了一次 Minor GC，是因为 Eden 区的三个对象都是 2M，而 s1 或 s0 空间只有 1M。所以三个对象直接进入了老年代。后一个 3M 对象进入 Eden 区。从 GC 日志上可以看出 <code> eden space 8192K, 37% used</code>。Eden 区用了大约 3M,<code>tenured generation total 10240K, used 6144K</code> 老年用了大约 6M。如果把 <code>byte4 = new byte[3 * _1MB];</code>
这一行代码注释掉，就不会发生 GC 了。注意禁用缓存，不然会影响结果。

<h2 id="大对象直接进入老年代">大对象直接进入老年代</h2>
当对象大小超过指定大小，就会直接进入老年代，用参数 <code>-XX:PretenureSizeThreshold</code> 设置指定的大小
参数控制：<code>-XX:PretenureSizeThreshold=2m -XX:+PrintGC -Xms20m -Xmx20m -Xmn1m -XX:+PrintGCDetails -XX:+PrintCommandLineFlags -XX:SurvivorRatio=8 -XX:-UseTLAB -XX:UseSerialGC</code>
<pre><code class="highlight-chroma">public class Lecture03 {
    private static final int _1MB = 1024 * 1024;

    public static void main(String[] args) {
        byte[] b = new byte[5 * _1MB];
    }
}
</code></pre>
打印日志如下：
<pre><code class="highlight-chroma">Heap
 def new generation total 9216K, used 1923K [0x00000000fec00000, 0x00000000ff600000, 0x00000000ff600000

  eden space 8192K, 23% used [0x00000000fec00000, 0x00000000fede0f40, 0x00000000ff400000)
  from space 1024K, 0% used [0x00000000ff400000, 0x00000000ff400000, 0x00000000ff500000)
  to   space 1024K, 0% used [0x00000000ff500000, 0x00000000ff500000, 0x00000000ff600000)
 tenured generation

n total 10240K, used 5120K [0x00000000ff600000, 0x0000000100000000, 0x000000010000000)
</code></pre>
没有发生 GC, tenured generation total 10240K, used 5120K，老年代使用了大约 5M,注意前提使用 ParNew 或者 Serial 收集器和禁用线程缓存，不然影响结果。
<h2 id="对象到达指定年龄-进入老年代">对象到达指定年龄，进入老年代</h2>
<code>-XX:MaxTenuringThreshold</code>,设置对象经过几次 GC 进入老年代，默认 15，经过一次 GC 对像的年龄就加一，当到达指定年龄就会进入老年代。我测试的指定大小为 3
参数控制：<code>-XX:+PrintGC -Xms20m -Xmx20m -Xmn10m -XX:+PrintGCDetails -XX:+PrintCommandLineFlags -XX:SurvivorRatio=8 -XX:MaxTenuringThreshold=3 -XX:PretenureSizeThreshold=10m -XX:-UseTLAB</code>
<pre><code class="highlight-chroma">public class Lecture04 {
    private static final int _1MB = 1024 * 1024;

    public static void main(String[] args) {
        //设置的长期活对象，不要超过整个s0区的大小，否则会直接进入老年代
        byte[] b1 = new byte[_1MB/4];
        //将Eden 区满，触发MinorGC
        for (int i = 0; i < 20; i++) {
            for (int j = ; j < 10; j++) {
                byte[] b = new byte[_1MB];
            }
        }
    }
}

</code></pre>
GC 部分日志如下
<pre><code class="highlight-chroma">[GC (Allocation Failure) [PSYoungGen: 7328K->1016K(9216K)] 7328K->1024K(19456K), 0.0012449 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 8184K->1016K(9216K)] 8192K->1024K(19456K), 0.0007303 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 8184K->984K(9216K)] 8192K->992K(19456K), 0.0006177 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 8152K->0K(9216K)] 8160K->948K(19456K), 0.0007460 secs] [Times: ser=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 7168K->0K(9216K)] 8116K->948K(19456K), 0.0003422 secs] [Times: ser=0.00 sys=0.00, real=0.00 secs]
[GC (Allocation Failure) [PSYoungGen: 7168K->0K(9216K)] 8116K->948K(19456K), 0.0002871 secs] [Times:

ser=0.00 sys=0.00, real=0.00 secs]
</code></pre>
可见在经历三次 GC 后，存活的对象都进入老年代了，新生代已经没有对象了，因为达到了指定龄，进入老年代了。