链滴

一些容易理解的 JVM 参数调优

作者：craig

原文链接：https://ld246.com/article/1549033952882

来源网站：链滴

JVM调优有许多参数优化，下面整理了一些我自己能够理解的参数，后面慢慢补充

-XX:AutoBoxCacheMax

-XX:+AlwaysPreTouch

CMSInitiatingOccupancyFraction

MaxTenuringThreshold

ExplicitGCInvokesConcurrent

-Xmx, -Xms

NewRatio

-XX:AutoBoxCacheMax

JAVA进程启动的时候,会加载rt.jar这个核心包的,rt.jar包里的Integer自然也是被加载到JVM中,Intege里面有一个IntegerCache缓存,如下:

```
private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer cache[];
```

原文链接：一些容易理解的 JVM 参数调优

```
        static {
            // high value may be configured by property
            int h = 127;
            String integerCacheHighPropValue =
                sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
            if (integerCacheHighPropValue != null) {
                int i = parseInt(integerCacheHighPropValue);
                i = Math.max(i, 127);
                // Maximum array size is Integer.MAX_VALUE
                h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
                cache[k] = new Integer(j++);
        }

        private IntegerCache() {}
    }
```

IntegerCache有一个静态代码块,JVM在加载Integer这个类时,会优先加载静态的代码。当JVM进程启完毕后, -128 ~ +127 范围的数字会被缓存起来,调用valueOf方法的时候,如果是这个范围内的数字,则接从缓存取出。

超过这个范围的,就只能构造新的Integer对象了。

```
public static Integer valueOf(int i) {
        assert IntegerCache.high >= 127;
        if (i >= IntegerCache.low && i <= IntegerCache.high)
            return IntegerCache.cache[i + (-IntegerCache.low)];
        return new Integer(i);
    }
```

因此可以根据实际情况把AutoBoxCacheMax的值设置的大写,比如江南白衣推荐的

```
-XX:AutoBoxCacheMax=20000
```

-XX:+AlwaysPreTouch

JAVA进程启动的时候,虽然我们可以为JVM指定合适的内存大小,但是这些内存操作系统并没有真正的配给JVM,而是等JVM访问这些内存的时候,才真正分配,这样会造成以下问题。

1、GC的时候,新生代的对象要晋升到老年代的时候,需要内存,这个时候操作系统才真正分配内存,这样会加大young gc的停顿时间;

2、可能存在内存碎片的问题。

可以在JVM启动的时候,配置

```
-XX:+AlwaysPreTouch
```

参数,这样JVM就会先访问所有分配给它的内存,让操作系统把内存真正的分配给JVM.后续JVM就可以畅的访问内存了。

JAVA 1.7用的垃圾收集算法还是CMS,下文提到的参数都是针对CMS的。

CMSInitiatingOccupancyFraction

java垃圾回收算法之-CMS(并发标记清除),垃圾收集线程会跟应用的线程一起并行的工作,万一垃圾收线程在工作的时候,老年代内存不足怎么办?因此最好还是提前启动CMS来收集垃圾(CMS GC)。

可以通过设置

```
CMSInitiatingOccupancyFraction=75
```

那么当老年代堆空间的使用率达到75%的时候就开始执行垃圾回收,CMSInitiatingOccupancyFractio默认值是92%,这个就太大了。

CMSInitiatingOccupancyFraction参数必须跟下面两个参数一起使用才能生效的。

```
-XX:+UseConcMarkSweepGC
-XX:+UseCMSInitiatingOccupancyOnly
```

MaxTenuringThreshold

新生代是使用copy算法来进行垃圾回收的,可以参看

java垃圾回收算法之-coping复制

默认情况下,当新生代执行了15次young gc后,如果还有对象存活在Survivor区中,那么就可以直接将这对象晋升到老年代,但是由于新生代使用copy算法,如果Survivor区存活的对象太久的话,Survivor区存的对象就越多,这个就会影响copy算法的性能,使得young gc停顿的时间加长,建议设置成6。

```
-XX:MaxTenuringThreshold=6
```

ExplicitGCInvokesConcurrent

如果系统使用堆外内存,比如用到了Netty的DirectByteBuffer类,那么当想回收堆外内存的时候,需要调用

```
System.gc()
```

而这个方法将进行full gc,整个应用将会停顿,如果是使用CMS垃圾收集器,那么可以设置

```
-XX:+ExplicitGCInvokesConcurrent
```

这个参数来改变System.gc()的行为,让其从full gc --> CMS GC,CMS GC是并发收集的,且中间执行的程中,只有部分阶段需要stop the world。

注意:设置了ExplicitGCInvokesConcurrent,那就不要设置DisableExplicitGC参数来禁掉System.gc()。

-Xmx, -Xms

这两个一般都是设置4个g

NewRatio

GC最多的还是发生在新生代的young gc,所以可以提高一下新生代在整个堆的占用比例,建议设置为对

分,尽量避免young gc

-XX:NewRatio=1