[**深入理解JVM(六)——JVM性能调优实战**](https://blog.csdn.net/u010425776/article/details/51232463)

<https://blog.csdn.net/u010425776/article/details/51232463>

## 如何在高性能服务器上进行JVM调优？

为了充分利用高性能服务器的硬件资源，有两种JVM调优方案，它们都有各自的优缺点，需要根据具体的情况进行选择。

### 1. 采用64位操作系统，并为JVM分配大内存

我们知道，如果JVM中堆内存太小，那么就会频繁地发生垃圾回收，而垃圾回收都会伴随不同程度的程序停顿，因此，如果扩大堆内存的话可以减少垃圾回收的频率，从而避免程序的停顿。

因此，人们自然而然想到扩大内存容量。而32位操作系统理论上最大只支持4G内存，64位操作系统最大能支持128G内存，因此我们可以使用64位操作系统，并使用64位JVM，并为JVM分配更大的堆内存。但问题也随之而来。

堆内存变大后，虽然垃圾收集的频率减少了，但每次垃圾回收的时间变长。如果对内存为14G，那么每次Full GC将长达数十秒。如果Full GC频繁发生，那么对于一个网站来说是无法忍受的。

因此，对于使用大内存的程序来说，一定要减少Full GC的频率，如果每天只有一两次Full GC，而且发生在半夜， 那完全可以接受。

要减少Full GC的频率，就要尽量避免太多对象进入老年代，可以有以下做法：

* 确保对象都是“朝生夕死”的   
  一个对象使用完后应尽快让他失效，然后尽快在新生代中被Minor GC回收掉，尽量避免对象在新生代中停留太长时间。
* 提高大对象直接进入老年代的门槛   
  通过设置参数-XX:PretrnureSizeThreshold来提高大对象的门槛，尽量让对象都先进入新生代，然后尽快被Minor GC回收掉，而不要直接进入老年代。

#### 注意：使用64位JDK的注意点

1. 64位JDK支持更大的堆内存，但更大的堆内存会导致一次垃圾回收时间过长。
2. 现阶段，64位JDK的性能普遍比32位JDK低。
3. 堆内存过大无法在发生内存溢出时生成内存快照   
   若将堆内存设为10G，那么当堆内存溢出时就要生成10G的大文件，这基本上是不可能的。
4. 相同程序，64位JDK要比32位JDK消耗更大的内存

### 2. 使用32位JVM集群

针对于64位JDK种种弊端，我们更多选择使用32位JDK集群来充分利用高性能机器的硬件资源。

#### 如何实现？

在一台服务器上运行多个服务器程序，这些程序都运行在32位的JDK上。然后再运行个服务器作为反向代理服务器，由它来实现负载均衡。   
由于32位JDK最多支持2G内存，因此每个虚拟结点的堆内存可以分配1.6G，一共运行10个虚拟结点的话，这台物理服务器可以拥有16G的堆内存。 

#### 有啥弊端？

1. 多个虚拟节点竞争共享资源时容易出现问题   
   如多个虚拟节点共同竞争IO操作，很可能会引起IO异常。
2. 很难高效地使用资源池   
   如果每个虚拟节点使用各自的资源池，那么无法实现各个资源池的负载均衡。如果使用集中式资源池，那么又存在竞争的问题。
3. 每个虚拟节点最大内存为2G

## 别忘了直接内存也可能导致内存溢出！

### 问题描述

有个小型网站，使用32位JDK，堆1.6G。运行期间发现老是出现内存溢出。为了判断是否是堆内存溢出，在程序运行前添加参数：-XX:+HeapDumpOnOutOfMemeryError(添加这个参数后当堆内存溢出时就会输出异常日至)。但当再次发生内存溢出时，没有生成相关异常日志。从而可以判定，不是堆内存发生溢出。 

### 问题分析

我们可以发现，在32位JDK中，将1.6G分配给了堆，还有一部分分配给了JVM的其它内存，只有少于0.4G的内存为非JVM内存。我们知道，如果使用了NIO，那么JVM会在JVM内存之外分配内存空间，这部分内存也叫“直接内存”。因此，如果程序中使用了NIO，那么就要小心“直接内存”不足时发生内存溢出异常了！ 

### 直接内存的垃圾回收过程

直接内存虽然不是JVM内存空间，但它的垃圾回收也有JVM负责。直接内存的垃圾回收发生在Full GC时，只有当老年代内存满时，垃圾收集器才会顺便收集一下直接内存中的垃圾。   
如果直接内存已满，但老年代没满，这时直接内存先是抛出异常，相应的catch块中调用System.gc()。由于System.gc()只是建议JVM回收，JVM可能不马上回收内存，那么这时直接内存就抛出内存溢出异常，使得程序终止。

## JVM崩溃的原因

当内存溢出时，JVM仅仅会终止当前运行的程序，那么什么时候JVM会崩溃呢？ 

### 什么是异步请求？

我们知道，Web服务器和客户端采用HTTP通信，而HTTP底层采用TCP通信。异步通信就是当客户端向服务器发送一个HTTP请求后，将这个请求的TCP连接委托给其它线程，然后它转而做别的事，那条被委托的线程保持TCP连接，等待服务器的回信。当收到服务器回信后，再将收到的数据转交给刚才的线程。这个过程就是异步通信过程。 

### 异步请求如何造成JVM崩溃？

如果一个Web应用使用了较多的异步请求(AJAX)，每次主线程发送完请求后都将TCP连接交给一条新的线程去等待服务器回信，那么如果网络不流畅时，这些受委托的线程迟迟等不到服务器的回信，因此保持着TCP连接。当TCP连接过多时，超过JVM的承受能力，JVM就发生崩溃。

## 如何处理大对象？

大对象对于JVM来说是个噩耗。如果对象过大，当前新生代的剩余空间装不下它，那么就需要使用分配担保机制，将当前新生代的对象都复制到老年代中，给大对象腾出空间。分配担保涉及到大量的复制，因此效率很低。

那么，如果将大对象直接放入老年代，虽然避免了分配担保过程，但该对象只有当Full GC时才能被回收，而Full GC的代价是高昂的。如果大对象过多时，老年代很快就装满了，这时就需要进行Full GC，如果Full GC频率过高，程序就会变得很卡。

**因此，对于大对象，有如下几种处理方法：   
1. 在写程序的时候尽量避免大对象   
从源头降低大对象的出现，尽量选择空间利用率较高的数据结构存储。   
2. 尽量缩短大对象的有效时间   
对象用完后尽快让它失效，好让垃圾收集器尽快将他回收，避免因在新生代呆的时间过长而进入老年代。**