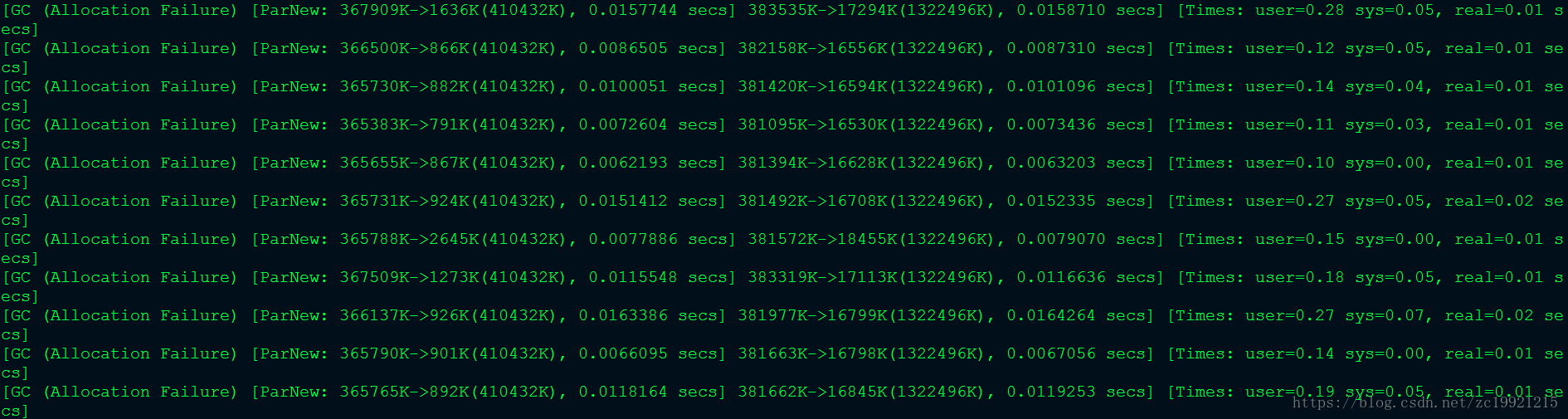
**GC(Allocation Failure)引发的一些JVM知识点梳理**

<https://blog.csdn.net/zc19921215/article/details/83029952>

日前查看某个程序的日志，发现一直在报GC相关的信息，不确定这样的信息是代表正确还是不正确，所以正好借此机会再复习下GC相关的内容：



以其中一行为例来解读下日志信息：

[GC (Allocation Failure) [ParNew: 367523K->1293K(410432K), 0.0023988 secs] 522739K->156516K(1322496K), 0.0025301 secs] [Times: user=0.04 sys=0.00, real=0.01 secs]

GC：

表明进行了一次垃圾回收，前面没有Full修饰，表明这是一次Minor GC ,注意它不表示只GC新生代，并且现有的不管是新生代还是老年代都会STW。

Allocation Failure：

表明本次引起GC的原因是因为在年轻代中没有足够的空间能够存储新的数据了。

ParNew：

    表明本次GC发生在年轻代并且使用的是ParNew垃圾收集器。ParNew是一个Serial收集器的多线程版本，会使用多个CPU和线程完成垃圾收集工作（默认使用的线程数和CPU数相同，可以使用-XX：ParallelGCThreads参数限制）。该收集器采用复制算法回收内存，期间会停止其他工作线程，即Stop The World。

367523K->1293K(410432K)：单位是KB

三个参数分别为：GC前该内存区域(这里是年轻代)使用容量，GC后该内存区域使用容量，该内存区域总容量。

0.0023988 secs：

    该内存区域GC耗时，单位是秒

522739K->156516K(1322496K)：

三个参数分别为：堆区垃圾回收前的大小，堆区垃圾回收后的大小，堆区总大小。

0.0025301 secs：

该内存区域GC耗时，单位是秒

[Times: user=0.04 sys=0.00, real=0.01 secs]：

    分别表示用户态耗时，内核态耗时和总耗时

分析下可以得出结论：

    该次GC新生代减少了367523-1293=366239K

    Heap区总共减少了522739-156516=366223K

    366239 – 366223 =17K，说明该次共有17K内存从年轻代移到了老年代，可以看出来数量并不多，说明都是生命周期短的对象，只是这种对象有很多。

    我们需要的是尽量避免Full GC的发生，让对象尽可能的在年轻代就回收掉，所以这里可以稍微增加一点年轻代的大小，让那17K的数据也保存在年轻代中。

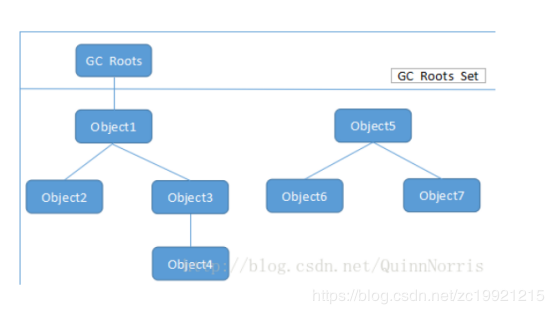
GC时，用什么方法判断哪些对象是需要回收：

引用计数法(已经不用了)

可达性分析法

前一种简而言之就是给对象添加一个引用计数器，有其他地方引用时这个计数器+1，引用失效时-1，为0时就可以删除掉了。但是它不能解决循环引用的问题，所以一般使用的都是后一种算法。

可达性分析法的基本思路就是通过一系列名为GC Roots的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链(Reference Chain)，当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的，那就可以回收掉了。



GC Roots一般都是些堆外指向堆内的引用，例如：

JVM栈中引用的对象

方法区中静态属性引用的对象

方法区中常量引用的对象

本地方法栈中引用的对象

以CMS为例，补充一些知识点介绍：

复制算法介绍：

因为新生代对象生命周期一般很短，现在一般将该内存区域划分为三块部分，一块大的叫Eden,两块小的叫Survivor。他们之间的比例一般为8:1:1。

使用的时候只使用Eden + 一块Survivor。用Eden区用满时会进行一次minor gc，将存活下面的对象复制到另外一块Survivor上。如果另一块Survivor放不下(对应虚拟机参数为 XX:TargetSurvivorRatio，默认50，即50%)，对象直接进入老年代。

（使用CMS时，默认的新生代收集器是ParNew）(有时新生代GC时，需要找到老年代中引用的新生代对象，这个时候会用到一种叫“卡表”的技术，避免老年代的全表扫描，具体怎么操作的暂时还不知道……)

Survivor区的意义：

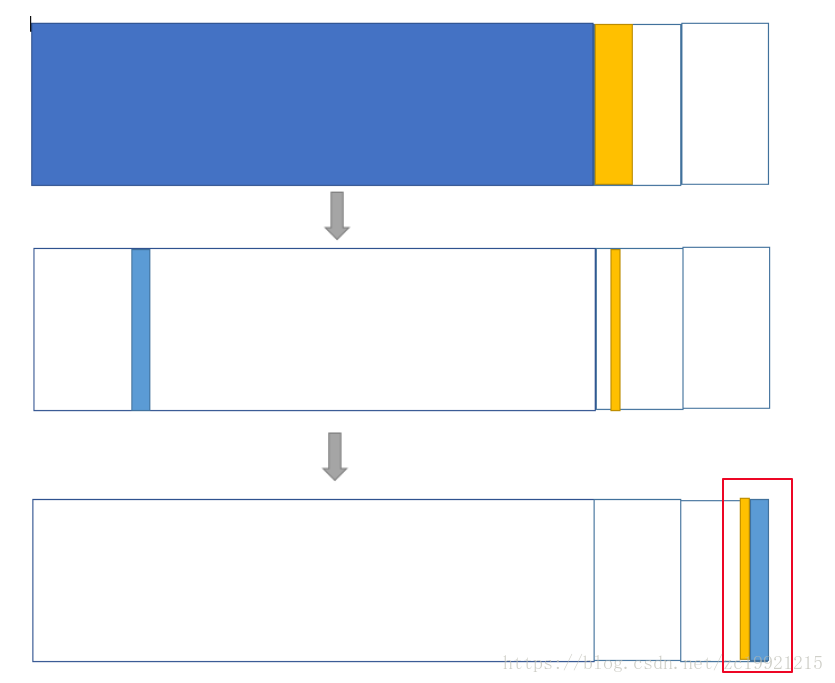
    如果没有survivor,Eden每进行一次minor gc，存活的对象就会进入老年代，老年代很快被填满就会进入major gc。由于老年代空间一般很大，所以进行一次gc耗时要长的多！尤其是频繁进行full GC，对程序的响应和连接都会有影响！

    Survivor存在就是减少被送到老年代的对象，进而减少Full gc的发生。默认设置是经历了16次minor gc还在新生代中存活的对象才会被送到老年代。

为什么要有两个Survivor：

    主要是为了解决内存碎片化和效率问题。如果只有一个Survivor时，每触发一次minor gc都会有数据从Eden放到Survivor，一直这样循环下去。注意的是，Survivor区也会进行垃圾回收，这样就会出现内存碎片化问题。如下图所示：

    碎片化会导致堆中可能没有足够大的连续空间存放一个大对象，影响程序性能。如果有两块Survivor就能将剩余对象集中到其中一块Survivor上，避免碎片问题。如下图所示：



Minor GC和Full GC的区别以及触发条件：

    Minor gc:

对于复制算法来说，当年轻代Eden区域满的时候会触发一次minor gc，将Eden和Survivor的对象复制到另外一块Survivor上，并且某个对象存活的时间超过一定minor gc次数直接进入老年代(默认15次，对应虚拟机参数 -XX:+MaxTenuringThreshold)。

    Full gc:（又叫major gc）

用于回收老年代。当老年代空间不足或者直接调用System.gc(不一定有用)时，会进行一次Full gc。（HotSpot还有一些其他复杂的触发条件，JDK8之前HotSpot的JVM中还有一个永久代(Perm区)，如果永久代内存不足也会触发Full gc。永久代主要存放一些class和元数据的信息 ---- 对应JVM规范中的方法区）

 一次Full gc很有可能会由一次minor gc触发，也可能是无法找到一块连续的空间分配给大对象而触发。

PS:JDK8中HotSpot为什么要取消永久代

    JDK8取消了永久代，新增了一个叫元空间(Metaspace)的区域，对应的还是JVM规范中的方法区。区别在于元空间使用的并不是JVM中的内存，而是使用本地内存。

    而这么做的原因大致有以下几点：

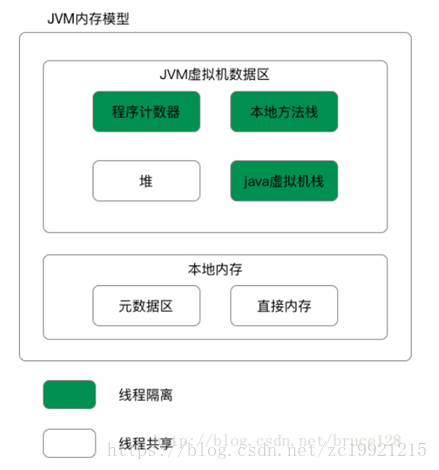
    1、字符串存在永久代中，容易出现性能问题和内存溢出。

　　2、类及方法的信息等比较难确定其大小，因此对于永久代的大小指定比较困难，太小容易出现永久代溢出，太大则容易导致老年代溢出。

　　3、永久代会为 GC 带来不必要的复杂度，并且回收效率偏低。

　　4、Oracle 可能会将HotSpot 与 JRockit 合二为一。

补充下JDK8内存模型图：



参考：

https://blog.csdn.net/antony9118/article/details/51425581(两个Survivor的意义)

https://zhidao.baidu.com/question/1111800566588999699.html(Full GC什么时候触发)

https://blog.csdn.net/l1394049664/article/details/81486470#%E4%BA%94%E3%80%81java8%E5%86%85%E5%AD%98%E6%A8%A1%E5%9E%8B%E5%9B%BE(JDK8内存模型图)

https://blog.csdn.net/quinnnorris/article/details/75040538(可达性分析算法解析)

https://segmentfault.com/a/1190000007726689(卡表是什么)

---------------------

作者：淡定一生2333

来源：CSDN

原文：https://blog.csdn.net/zc19921215/article/details/83029952

版权声明：本文为博主原创文章，转载请附上博文链接！