GC机制

# GC机制

GC：garbage collection

# 网上解释

Garbage Collection简称为GC,是垃圾回收的意思。**内存处理器**是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃。**Java语言提供的GC功能可以自动的检测对象是否超过作用域，从而达到自动回收内存的目的，java语言没有提供释放已分配内存的显示操作方法，资源回收工作全部交由GC来完成，程序员不能精确的控制垃圾回收的时机**。

GC在实现垃圾回收时的基本原理：

**Java的内存管理实际就是对象的管理，其中包括对像的分配和释放。**对于程序员来说，分配对象使用new关键字，释放对象时只是将对象赋值为null，让程序员不能够再访问到这个对象，该对象被称为“不可达”。**GC将负责回收所有“不可达”对象的内存空间。**

   对于GC来说，当程序员创建对象时，GC就开始监控这个对象地址、大小以及使用情况。**通常GC采用有向图的方式记录并管理堆中的所有对象，通过这种方式确定哪些对象是“可达”的，哪些对象是“不可达”的。**当GC确定一些对象为“不可达时”GC就有责任回收这些内存空间，但为了GC能够在不同的平台上实现，java规范对GC的很多行为都没有进行严格的规定。例如**对于采用什么类型的回收算法、什么时候进行回收等重要问题都没有明确的规定，因此不同的JVM实现着不同的的实现算法**，这也给JAVA程序员的开发带来了很多不确定性。

     根据GC的工作原理，可以通过一些技巧和方式让GC运行更快，高效而又合理。编程建议如下：

**1、尽早释放无用对象的引用**，特别注意一些复杂对象，如数组，队列等。对于此类对象，GC回收它们的效率一般较低，如果程序允许，应尽早将不用的引用对象赋为null，这样可以加速GC的工作。

**2、尽量少用finalize函数。**finalize是java提供给程序员用来释放对象或资源的函数，但是它会加大GC的工作量，因此尽量少采用finalize函数回收资源。

**当程序有一定的等待时间，程序员可以手动执行System.gc(),通知GC运行，但是java语言规范并不保证GC一定会执行。**

# finalize是什么？

Object类中定义的方法：finalize()方法：



这个一般有垃圾回收器调用，不是程序员调用。

finalize是方法名，java技术允许使用**finalize()方法**在**垃圾收集器**从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象的调用的。它是在Object类定义的，因此所有的类都继承了它。子类覆盖finalize()方法以整理系统资源或者执行其他清理工作。**finalize()方法是在垃圾收集器删除对象之前对这个对象的调用的。**

# 基本概念：

GC主要用于**管理JVM的堆区**，所以先来介绍一下JVM的内存分配。

## **程序计数器(Program Conuter Register)**

        程序计数器是一块较小的内存空间，它是当前线程执行字节码的行号指示器，字节码解释工作器就是通过改变这个计数器的值来选取下一条需要执行的指令。它是线程私有的内存，也是唯一一个没有OOM异常的区域。

## **Java虚拟机栈区(Java Virtual Machine Stacks)**

        也就是通常所说的栈区，它描述的是Java方法执行的内存模型，每个方法被执行的时候都创建一个栈帧(Stack Frame)，用于存储局部变量表、操作数栈、动态链接、方法出口等。每个方法被调用到完成，相当于一个栈帧在虚拟机栈中从入栈到出栈的过程。此区域也是线程私有的内存，可能抛出两种异常：如果线程请求的栈深度大于虚拟机允许的深度将抛出StackOverflowError；如果虚拟机栈可以动态的扩展，扩展到无法动态的申请到足够的内存时会抛出OOM异常。

## **本地方法栈(Native Method Stacks)**

        本地方法栈与虚拟机栈发挥的作用非常相似，区别就是虚拟机栈为虚拟机执行Java方法，本地方法栈则是为虚拟机使用到的Native方法服务。

## **堆区(Heap)**

**所有对象实例和数组都在堆区上分配，堆区是GC主要管理的区域。**堆区还可以细分为新生代、老年代，新生代还分为一个Eden区和两个Survivor区。此块内存为所有线程共享区域，当堆中没有足够内存完成实例分配时会抛出OOM异常。

## **方法区(Method Area)**

       方法区也是所有线程共享区，用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。GC在这个区域很少出现，这个区域内存回收的目标主要是对常量池的回收和类型的卸载，回收的内存比较少，所以也有称这个区域为永久代(Permanent Generation)的。当方法区无法满足内存分配时抛出OOM异常。

## **运行时常量池(Runtime Constant Pool)**

        运行时常量池是方法区的一部分，用于存放编译期生成的各种字面量和符号引用。

# 如何学习GC机制？需要搞清楚哪些地方？

针对GC的原理机制，主要搞清楚下面三个问题。

1、什么时候回收？

2、哪些需要回收？

3、怎么回收？

**1、什么时候回收？**上面提到GC主要管理的是堆区，堆区主要分为新生代和老年代。大概解释一下这两个区域。

**新生代**：分为一个Eden和两个Survivor区。**新new的对象都放在这里，很快消亡**。

**老年代**：新new的大对象直接丢到这里(为了避免在Eden区和两个Survivor区发生大量的内存拷贝)，其余就是在新生代多次回收没被干掉过来变成老家伙的对象了。

①对**象优先分配到新生代的Eden区，当不够空间的时候进行一次Minor GC，清理频率很高。**

②**Full GC发生在老年代，当不够空间的时候进行一次Full GC，伴随着也会进行一次Minor GC。**

③**进行Minor GC时，会判断每次变成晋升到老年代的对象平均值是否大于老年代剩余空间，如果大于，则进行一次Full GC，如果小于就会去判断HandlePromotionFailure设置是否允许担保失败，如果允许，则进行Minor GC，不允许则改为Full GC。**

**2、哪些需要回收？**

为了下面内容更好理解，首先来了解一下finalize方法。

**什么是finalize()方法？**

**每次进行GC之前系统都会调用一次finalize()方法，用以清理所有活动并且释放资源。**

什么时候调用finalize()方法？

1、GC调用之前，例如运行System.gc();（**调用System.gc()只是建议JVM去执行，是否执行还得JVM去判断**）

2、**程序退出时，每个对象都会调用finalzie**

3、**显式调用finalize**

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

**引用根搜索算法（GC ROOT Tracing）**，当一个对象没有任何引用连接的时候，则说明对象不可达，即对象不可用，这个时候就需要进行GC清理。

**判断对象是否可达的依据是有没必要执行finalize()方法。**如果finalize()方法没有被覆盖或者已经被系统调用过一次了(每个对象生命周期内只能调用一次)，则被不可达，需要进行GC清理，否则进行自救，恢复引用连接。

**3、怎么回收？**

不同区域回收算法不同。

**新生代：停止-复制。**

**老年代：标记-清理、标记-整理。**

新生代：新生代分为一个Eden区、两个Survivor区(Survivor0、Survivor1)。回收时先把Eden存活对象复制到Survivor0区，清空Eden区，当Survivor0区满了以后，把Eden和Survivor0区的存活对象复制到Survivor1区，清空Eden区和Survivor0区，之后交换Survivor0和Survivor1区，保持Survivor1区是空的，如此往复。

老年代：这两个没什么说的，字面理解。