**[JVM的堆和栈以及GC算法的介绍](https://blog.csdn.net/qq_20641565/article/details/60332593)**

<https://blog.csdn.net/qq_20641565/article/details/60332593>

JVM就是java虚拟机，我们可以把它理解成一个操作系统，每个不同的平台都有不同的JVM，比如linux系统和windows系统，就是因为这个原因所以java程序就有了一个很突出的特性就是 **跨平台性**

其中JVM中的堆和栈这两个东西以及它的垃圾回收机制是我们平时遇到得最多的，那么下面就介绍一下JVM的堆栈以及GC。

* **1.JVM的堆栈**

栈：在jvm中栈用来存储一些对象的引用、局部变量以及计算过程的中间数据，在方法退出后那么这些变量也会被销毁。它的存储比堆快得多，只比CPU里的寄存器慢

堆：用来存储程序中的一些对象，比如你用new关键字创建的对象，它就会被存储在堆内存中，但是这个对象在堆内存中的首地址会存储在栈中。

栈内存在JVM中默认是1M，可以通过下面的参数进行设置

-Xss

最小堆内存在JVM中默认物理内存的64分之1，最大堆内存在JVM中默认物理内存4分之一，且建议最大堆内存不大于4G，并且设置-Xms=-Xmx避免每次GC后，调整堆的大小，减少系统内存分配开销

-Xms

-Xmx

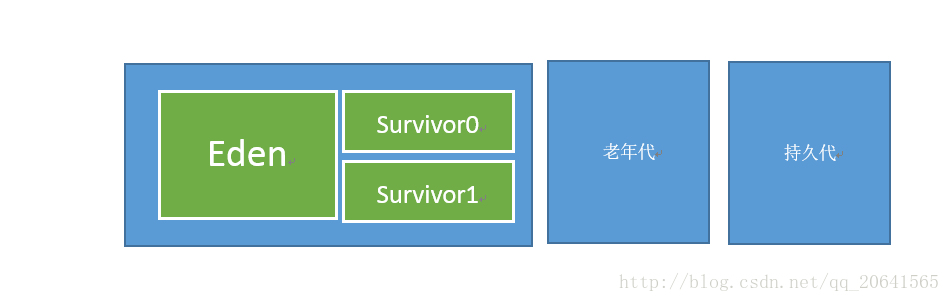
**在jvm的堆内存中有三个区域：**

1.年轻代：用于存放新产生的对象。

2.老年代：用于存放被长期引用的对象。

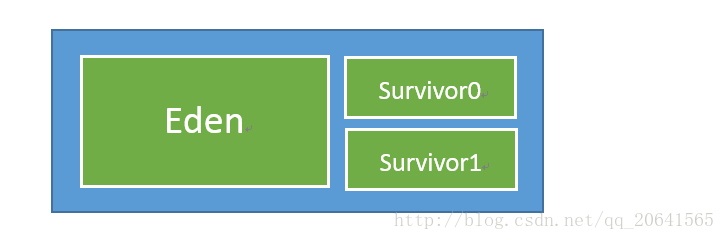
3.持久带：用于存放Class，method元信息。

如图：



* **一.年轻代**

年轻代中包含两个区：Eden 和survivor，并且用于存储新产生的对象，其中有两个survivor区如图：



可以使用参数配置年轻代的大小，如果配置它为100M那么就相当于2\*survivor+Eden = 100M

-Xmn

* 1

可以配置Eden 和survivor区的大小，这里配置的是比值，jvm中默认为8，意思就是Eden区的内存比上survivor的内存等于8，如果年轻代的Xmn配置的100M，那么Eden就会被分配80M内存，每个survivor分配10M内存

-XX:SurvivorRatio

* 1

还可以配置年轻代和老年代的比值，这里需要注意：老年代的内存就是通过这个比值设置，jvm没有给你直接设置老年代内存大小的参数；如果整个堆内存设为100M并且在这里设置年轻代和老年代的比值为7，如果持久代占用了10M，那么100M-10M=90M这里的90M就是老年代和年轻代的内存总和，且年轻代占用（90/（7+1）\*7）的内存，老年代就占用（90/（7+1）\*1）的内存。

-XX:NewRatio

* 1
* **二.老年代**

年轻代在垃圾回收多次都没有被GC回收的时候就会被放到老年代，以及一些大的对象（比如缓存，这里的缓存是弱引用），这些大对象可以不进入年轻代就直接进入老年代（1.防止新生代有大量剩余的空间，而大对象创建导致提前发生GC；2.防止在eden区和survivor区的大对象复制造成性能问题），这个可以通过如下参数设置，表示单个对象超过了这个值就会直接到老年带（默认为0）：

-XX:PretenureSizeThreshold

* 1

并且大的数组对象也会直接放到老年代，比如array和arrayList（底层用数组实现），因为数组需要连续的空间存储数据。

* **三.持久代**

持久代用来存储class，method元信息，大小配置和项目规模，类和方法的数量有关，一般配置128M就够了，设置原则是预留30%空间，它可以通过如下参数进行大小配置：

-XX: PermSize

-XX: MaxPermSize

持久代也可能会被GC回收，如果持久代理的常量池没有被引用以及一些无用的类信息和类的Class对象也会被回收。

* **四.JVM内存垃圾收集算法**

**1.引用计数器的方法**

类似于Objective-C的内存回收，在OC语言不使用arc机制时，在创建对象，且被引用时候也会对计数器加1，使用完成后会调用release方法就会对计数器减1

jvm的这个算法也和上面的机制类似，但是他不能解决对象循环引用问题，也不好解决精确的计算，因为java程序开发，内存回收是对我们透明的，而OC是在代码层面自己去手动控制

-**2.根搜索算法**

有向图算法，从GC Roots开始向下面搜索，搜索走过的路径叫做引用链接，当一个对象到达GC Roots没有任何引用链时，则这个对象就是不可达对象，就可以被回收，但是回收的时候会筛查出覆盖了finalze()方法且该对象finalze()方法没有被虚拟机调用过放入F-Queue队列然后执行F-Queue队列中对象的finalze()方法后如果该对象重新与某个GC Roots对象相关联，那么会将该对象从回收队列中移除

比如jvm中栈中指向堆中对象的指针就可以理解成一种GC Roots，因为栈中保存的是对象的指针指向的是堆中对象的首地址，当栈中的指针没有了那么堆中的对象就是不可达对象，就会被GC回收

* **五.jvm内存垃圾回收算法**

**1.复制算法 （用于新生代）**

用于新生代从Eden区到survivor0或者survivor1移动的时候

从根集合扫描如果对象被引用，就会被copy到survivor0或者survivor1，然后剩下的都是不可达对象，就可以被回收掉，然后S0或者S1的对象就会到老年代中，在存活对象比较少的时候很高效并且不会产生内存碎片，就是内存需要额外划分一块survivor区出来

**2.标记清除算法 （用于老年代）**

就是使用根搜索算法扫描，如果可达就标记，当扫描完成后，就对未标记的对象进行回收，它不需要像复制算法一样，需要一个新的内存区，而且不会对对象进行copy，这种对对象存活的多的情况下很高效，但是这样会产生内存碎片

**3.标记整理压缩算法 （用于老年代）**

就是在标记清除算法之后对内存碎片进行整理，只是他会对没有清理的对象进行移动，代价高

* **六.内存容量配置建议**

响应时间优先以及吞吐量优先类型的年轻代就尽可能设置大，老年代就可以设置小一点，因为在年轻代设置大了，那么年轻代的GC收集频率就会小，而且减少到达老年代的对象，最后尽量使老年代里面管理的对象是可用率高的，这样的话就会减少在年轻带的GC频次，并且防止进入老年代的对象过多触发FGC而stop-the-world