

JVM 内存包含如下几个部分：

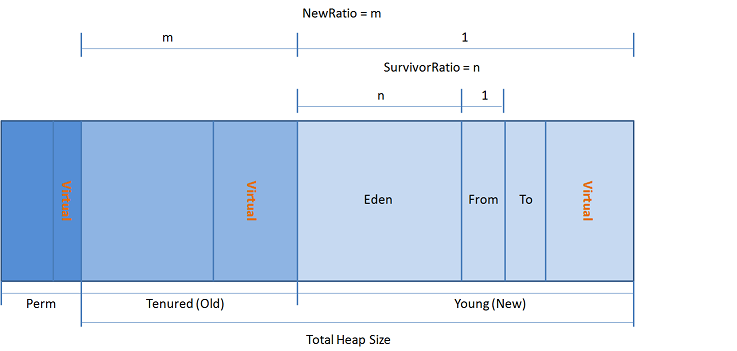
* Heap Memory 存放Java对象
* Non-Heap Memory 存放类加载信息和其它meta-data
* Other 存放JVM 自身代码等

在JVM启动时，就已经保留了固定的内存空间给Heap内存，这部分内存并不一定都会被JVM使用，但是可以确定的是这部分保留的内存不会被其他进程使用。这部分内存大小由 -Xmx参数指定。而另一部分内存在JVM启动时就分配给JVM，作为JVM的初始Heap内存使用。影响这个的参数是 -Xms。

默认空余堆内存小于40%时，JVM 就会增大堆直到-Xmx的最大限制，可以由-XX:MinHeapFreeRatio指定。  
默认空余堆内存大于70%时，JVM 会减少堆直到-Xms的最小限制，可以由-XX:MaxHeapFreeRatio指定。

可以通过-XX:MaxPermSize设置Non-Heap大小.

**GC 的年代划分**



如果 -Xms指定的值比-Xmx的小，那么两者的差值就是Virtual内存值。随着程序的运行，Eden区、 Tenured区和Perm区会逐渐使用保留的Virtual空间。

JVM内存模型中Heap区分两大块，一块是 NEW Generation,另一块是Old Generation. 在NewGeneration中，有一个叫Eden的空间，主要是用来存放新生的对象，还有两个Survivor Spaces（from,to），它们的大小总是一样，它们用来存放每次垃圾回收后存活下来的对象。在OldGeneration中，主要存放应用程序中生命周期长的内存对象。在NewGeneration块中，垃圾回收一般用Copying的算法，速度快。每次GC的时候，存活下来的对象首先由Eden拷贝到某个SurvivorSpace, 当Survivor Space空间满了后, 剩下的live对象就被直接拷贝到OldGeneration中去。因此，每次GC后，Eden内存块会被清空。在OldGeneration块中，垃圾回收一般用mark-compact的算法，速度慢些，但减少内存要求.  
垃圾回收分多级，0级为全部(Full)的垃圾回收，会回收OLD段中的垃圾；1级或以上为部分垃圾回收，只会回收NEW中的垃圾，内存溢出通常发生于OLD段或Perm段垃圾回收后，仍然无内存空间容纳新的Java对象的情况。

还有个Permanent Generation，主要用来放JVM自己的反射对象，比如类对象和方法对象等。关于这个区，它还提供**String pool**，看下面的例子：

**[java]** [view plaincopyprint?](http://blog.csdn.net/autofei/article/details/7456213)

1. String first = "abc";
2. String second = new String ("abc");

  
第一个对象存贮在Permanent Generation，而第二个对象存储在Heap里面。所以：

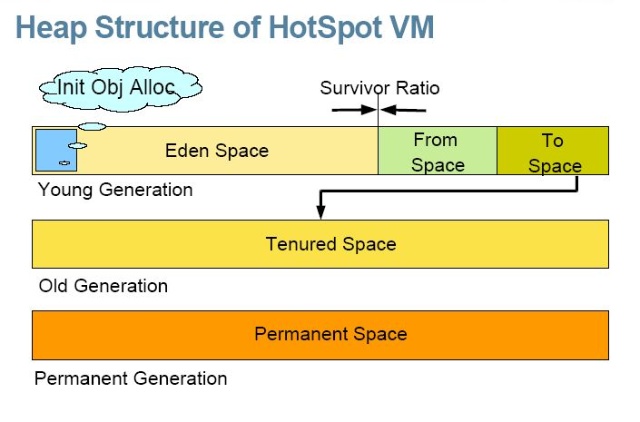
**[java]** [view plaincopyprint?](http://blog.csdn.net/autofei/article/details/7456213)

1. String s = "abc";
2. String p = "abc";

  
对象s和p指向同一个对象，这样效率大大提高。

内存申请过程如下：

1. JVM 会试图为相关Java对象在Eden中初始化一块内存区域
2. 当Eden空间足够时，内存申请结束。否则到下一步
3. JVM 试图释放在Eden中所有不活跃的对象（这属于1或更高级的垃圾回收）,释放后若Eden空间仍然不足以放入新对象，则试图将部分Eden中活跃对象放入Survivor区
4. Survivor区被用来作为Eden及OLD的中间交换区域，当OLD区空间足够时，Survivor区的对象会被移到Old区，否则会被保留在Survivor区
5. 当OLD区空间不够时，JVM 会在OLD区进行完全的垃圾收集（0级）
6. 完全垃圾收集后，若Survivor及OLD区仍然无法存放从Eden复制过来的部分对象，导致JVM无法在Eden区为新对象创建内存区域，则出现”out of memory错误”



具体算法请参考：[JDK5.0中JVM堆模型、GC垃圾收集详细解析](http://blog.csdn.net/sfdev/article/details/4483442)