运行时常量池

是方法区的一部分。基本类型的包装类引用的对象放在常量池中（如Integer i1 = 40;）

jvm怎样判断一个对象是否可回收？

Java采用可达性分析算法：当一个对象到GC Roots没有任何引用链相连时，则此对象可以被回收。

GC Roots对象引用包括:

栈中的局部变量（指向jvm堆里的对象）

类静态变量

运行时常量池里的引用（如Integer i1 = 40;）

字符串常量池中的引用

JVM内存分代模型：新生代、老年代、永久代

堆内存分为新生代和老年代，new的对象会放入新生代

新生代：

HotSpot把新生代划分为三块：1个Eden区（80%），2个Survivor区（各占10%）。

最开始，对象只在Eden进行分配

当Eden区满了，触发Minor GC，将Eden区中的存活对象转移到Survivor1中，同时清空Eden

下一次Minor GC，将Eden存活对象和Survivor1中存活对象转移到Survivor2中，然后清空Eden和Survivor1

循环往复

老年代

一个实例对象在新生代中，成功在15次垃圾回收之后，还是没有被回收到，那么就会被转移到老年代

永久代：就是JVM内存模型中提到的“方法区”

新生代对象何时会进入老年代？

* 新生代对象年龄超过一定阈值（默认15）
* 动态年龄判断

Survivor区的存活对象年龄从小到大进行累加，当累加到 X 年龄时的总和大于 Survivor区空间的50%时，那么比X大的 都会晋升到老年代

* 大对象
* Survivor区空间不足

Minor GC之后存活对象多于一块空的Survivor区域，则把这些对象全部迁移到老年代去

Minor GC

只清理新生代

当Eden区满时，触发Minor GC

FULL GC

会对整个堆进行整理，包括新生代、老年代、永久代

触发条件：

1. 调用System.gc()，系统建议执行Full GC，但是不必然执行。
2. 老年代空间不足

触发Minor GC之前,发现之前进入老年代的平均大小大于老年代的可用内存

Minor GC之后，存活对象进入老年代，可用空间不够

创建大对象，Eden区域当中放不下这个大对象，会直接保存在老年代当中，老年代空间也不足

1. 方法区空间不足

JVM 出现 fullGC 很频繁，怎么去线上排查问题？

垃圾回收算法

1.复制算法：用于新生代中对象的回收。基本思路就是：将新生代内存划分为两块，每次只使用其中的一块，当一块内存用完了，将存活的对象移动到另外一块上面，然后在把已使用过的内存空间一次清理掉

2.标记整理算法：

垃圾回收器（垃圾收集器）

ParNew：只清理新生代，复制算法。多线程，会造成停顿（Stop-The-World）

CMS：清理老年代，标记清除（Mark-Sweep）算法，存在内存碎片化问题

G1：清理新生代和老年代。可以把回收时间控制在设置的预期停顿时间范围。

非常适合大内存的机器；对于响应时效性要求高的系统，G1也非常合适