# GC

## java内存划分：

线程共享的：堆区，方法区。

***1.堆区：***存放对象实例，是理解Java GC机制最重要的区域，没有之一。1.8之后删除永久代。

***2.方法区：***用于存储虚拟机加载的：静态变量+常量+类信息+运行时常量池 （类信息：类的版本、字段、方法、接口、构造函数等描述信息 ）线程私有的：虚拟机栈，本地方法栈，程序计数器。默认最小值为16MB，最大值为64MB，可以通过-XX:PermSize 和 -XX:MaxPermSize 参数限制方法区的大小。

***1.虚拟机栈：***jvm执行的java方法。

***2.本地方法栈：***jvm执行的native方法。HotSpot将虚拟机栈和本地方法栈合并。

***3.程序计数器：***每个线程都有一个程序计算器，就是一个指针，指向方法区中的方法字节码（下一个将要执行的指令代码），由执行引擎读取下一条指令，是一个非常小的内存空间，几乎可以忽略不记。由于java多线程是通过线程轮流切换并分配执行时间来实现的，任何时刻只会执行一条线程中的指令。因此为了切换线程后依旧能恢复到正确的位置，需要一个独立的计数器。

## 执行对象

自动释放不再被引用的对象所占的堆内存，整理碎片空间。

1. 引用计数：有一个引用就+1，释放-1，0就回收，但是无法解决对象相互引用的问题。
2. 可达性分析：从GC Roots开始向下搜索，没有任何引用链相连时，则证明此对象是不可用的。不可达对象，需要被回收。

GC Roots包括：

a．虚拟机栈中引用的对象。

b．方法区中类静态属性实体引用的对象。

c．方法区中常量引用的对象。

d．本地方法栈中JNI（本地接口）引用的对象。

## 优缺点

优点：可以提高生产效率（减少释放空间的编程时间）和增加程序的健壮性（不再因为错误的释放空间导致程序崩溃）。

缺点：影响程序的性能，因为要动态的监视和释放。在明确的内存不再使用内存比起来，这个活动占用了更多的cpu时间。程序员缺乏安排cpu对释放对象的控制。

## 执行时机

程序调用System.gc()。

Minor GC：

Full GC：

# utp和tcp

1：UDP是面向报文的，发送方的UDP对应用层交下来的报文，不合并，不拆分，只是在其上面加上首部后就交给了下面的网络层，也就是说无论应用层交给UDP多长的报文，它制统统发送，一次发送一个。而对接收方，接到后直接去除首部，交给上面的应用层就完成任务了。因此，它需要应用层控制报文的大小

2：TCP是面向字节流的，它把上面应用层交下来的数据看成无结构的字节流来发送，知可以想象成流水形式的，发送方TCP会将数据放入“蓄水池”（缓存区），等到可以发送的时候就发送，不能发送就等着，TCP会根据当前网络的拥塞状态来确定每个报文段的道大小。

3：每一条TCP连接只能是点到点的，UDP支持一对一，一对多和多对多的交互通信。

4：TCP是流模式，UDP是数据报模式。

# netty

# mysql树

# hashmap的扩容和红黑树

# redis原理

# 锁原理