JVM分区

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，如果执行的是Java方法，记录的是正在执行的虚拟机字节码指令的地址，如果为native方法，值为null。

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、方法出口等信息。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，局部变量表中存储了编译期可知的各种基本数据类型、对象引用以及returnAddress类型

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。

在目前已经发布的的1.7中，字符串常量池已经从永久代中移出

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。他的一个重要特征是具备动态性，并非只有预先置入Class文件中常量池的内容才能进入方法区运行时常量池，运行期间也可以放入新的常量，例如String类的intern（）方法。

对象的创建过程

遇到new指令-类加载检查（检查指令参数是否能在常量池中定位到一个类的符号引用，并检查所代表的类是否已被加载、解析和初始化过）-为对象分配内存（内存大小在类加载后已完全确定，存在指针碰撞和空闲列表两种方式，取决于java堆是否规整，为解决线程安全问题：进行同步处理（CAS配上失败重试），另一种为每个线程预先分配本地线程分配缓冲（TLAB））-分配内存空间初始化为零值-对对象进行必要的设置（信息存于对象头之中）

对象的访问定位

1.句柄访问（在堆中划分一块内存来作为句柄池） 2.直接指针访问

JDK1.8 垃圾回收：

完全移除了永久代，但类的元数据信息（metadata）还在，只不过不再是存储在连续的堆空间上，而是移动叫做元空间的本地内存中。

1.8 内存模型：https://blog.csdn.net/bruce128/article/details/79357870

Minor GC、Major GC和Full GC

- 清理Eden区和 Survivor区叫Minor GC。
- 清理Old区叫Major GC。
- 清理整个堆空间—包括年轻代和老年代叫Full GC。

Minor GC的触发条件：当Eden区满时，会触发

Full GC触发条件：

- 调用System.gc()时，系统会通知建议执行Full GC，但是不必然执行。
- 老年代空间不足

- 方法区空间不足
- 通过Minor GC 后进入老年代的平均大小大于老年代的可用内存。

确定对象是否活着的方法

- **引用计数法：**添加引用计数器：当引用他时，计数器值加一，当失效时，减一，计数器为0的对象不可再被使用（很难解决对象之间循环引用的问题）
- **可达性分析算法（根搜索方法）：**通过一系列的称为“GC Roots”对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链，当一个对象到GC Roots没有任何引用链时（GC Roots 的对象有：虚拟机栈中引用的对象，方法区中类静态属性引用的对象，方法区中常量引用的对象，本地方法栈中JNI引用的对象）它将会第一次标记并且进行一次筛选，筛选条件是此对象有没有必要执行finalize()方法，当对象没有覆盖finalize()方法或者finalize()方法已经被虚拟机调用执行过一次，这两种情况都被视为没有必要执行finalize()方法，对于没有必要执行finalize()方法的将会被GC，对于有必要执行的，对象在finalize()方法中可能会自救，也就是重新与引用链上的任何一个对象建立关联即可。

四种引用类型

- 强引用：new 的对象，只要强引用存在，垃圾收集器永远不会回收掉被引用的对象
- 软引用：非必需对象，在将要发声内存溢出异常之前，才把这些对象列进回收范围之中进行第二次回收
- 弱引用：非必需对象，所关联对象只能生存到下一次垃圾收集发生之前
- 虚引用：完全不会对对象生存时间构成影响，无法通过虚引用取得对象实例，设置关联的唯一目的是能在这个对象被收集器回收时收到一个系统通知

GC停顿：GC进行时必须停止所有java执行线程，不可以出现分析过程中对象引用关系还在不断变化的情况，该点不满足的话分析结果准确性无法得到保证
安全点：程序并非在所有地方都可以暂停开始GC,只有到达安全点时才暂停:主动式中断：GC需要中断线程时，设置一个标志，各个线程执行时主动去轮询这个标志，为真时自己中断挂起，轮询标志地点和安全点重合
安全区域：对于程序“不执行”时，即引用关系不发生变化时，在任意地方开启GC都安全.

内存分配回收策略

- 对象优先在Eden分配
- 大对象（需要大量连续内存空间的java对象）直接进入老年代
- 长期存活的对象将进入老年代
- 动态对象年龄判定
- 空间分配担保