1、关键字说明：

1.1、GC Roots

找到“GC Roots”也是要花很长的时间，然而这里又有新的解决方法，就是通过采用一个OopMap的数据结构来记录系统中存活的“GC Roots”，在类加载完成的时候，虚拟机就把对象内什么偏移量上是什么类型的数据计算出来保存在OopMap，通过解释OopMap就可以找到堆中的对象，这些对象就是GC Roots。而不需要一个一个的去判断某个内存位置的值是不是引用。这种方式也叫准确式GC。

1.2、Stop the World

GC停顿，将非GC线程相关的用户线程临时挂起，实现对象状态的一致性，不能存在GC过程中，对象的状态飘忽不定的状态，做到准确GC。

1.3、Safe Point

程序并非在所有地方都能停下来开始GC，只有到达某个特定的位置才可以暂停，这个特定的位置就称为Safe Point。抢断式中断，首先爸所有线程全部中断，如果有线程并不处于安全点，那就恢复这个线程，让它跑到安全点上---抢先式中断；不直接对线程操作，使用标志信息，各个线程论询这个标志，发现标志为真是就挂起。轮询标志的地方和安全点是重合的—主动式中断。

1.4、Safe Region

被扩展的Safe Point，在这个区域内被认为是引用关系是不改变的，可以随时进行GC。

2、常见的垃圾回收算法

2.1、标记-清除法

是最基础的算法，后续的算法都是在此基础上不断完善和改进完成的。这个算法主要有两个不足：1、效率问题，不管是标记还是清除，效率都不高；2、空间问题，标记清除之后，会留下很多不连续的内存碎片，而空间碎片的增多不利于大对象的分配，可能会提前触发一次GC。

2.2、复制算法

将内存空间划分为大小想相等的两块，每次仅对其中的一块进行操作，当其中的一块使用完了，就将存活的对象复制到另一块上，随后将使用过的那块一次性清除干净。

这样看，一次只操作一块内存，将近50%的空间，显得太浪费了。其实，研究表明，新生代中的对象98%的属于朝生夕死的对象，所以并不需要按照1:1的形式来划分，而是按照这个比例进行划分：8:1:1的形式；

两个问题：1、为什么两块survivor区？2、survivor满了怎么办（分配担保）？

2.3、标记-整理法

标记整理算法的“标记”过程和标记-清除算法一致，只是后面并不是直接对可回收对象进行整理，而是让所有存活的对象都向一段移动，然后直接清理掉端边界意外的内存。

3、垃圾收集器

前面对几种常见的垃圾收集算法进行了简单的说明，下面所要讲到的垃圾收集器其实可看做是垃圾收集算法的具体实现。

3.1、Serial 收集器

一款采用复制算法思想实现的串行收集器，是一款发展较久的基本的收集器，一般用于新生代的垃圾收集；

3.2、parNew 收集器

是Serial收集器的多线程版本，二者在实现上共用了很多的代码；同样采用复制算法进行收集；server模式下的虚拟机首选的垃圾收集器；

3.3、**Parallel Scavenge收集器**

年轻代并行垃圾收集器，同样采用了复制算法进行实现；看起来和刚才介绍的parNew收集器是一致，其实Parallel Scavenge收集器更注重吞吐量。

先讲下什么是吞吐量。吞吐量的计算公式是：吞吐量=运行用户代码时间/(运行用户代码时间+GC消耗时间)；就是说，GC时间越短，吞吐量越大。调整吞吐量涉及两个参数，第一个参数，最大停顿时间，单位毫秒；每次发生GC时，会尽量在这个时间内进行。当然，这个值并不是越小越好，停顿时间缩短是以牺牲新生代空间和吞吐量来实现的，试想一下，一个更小的新生代虽然GC时间缩短了，但是却导致了更加频繁的GC；第二个参数：垃圾收集时间占总时间的比率，是一个0-100的整数，默认是99，计算公式为：

这个收集器和ParNew收集器还有一个显著的差异是，引入了自适应调节策略，这个参数的开关打开后，系统会自动收集性能监控信息，动态调整垃圾收集的合理停顿时间或吞吐量。我们只需要将基本的参数做好设置，比如堆内存大小，然后使用以上两个参数的其中一个进行设置，作为一个调优目标，具体细节交给jvm自动调节即可。

3.4、**Serial Old收集器**

以上介绍的是新生代的垃圾收集器，下面开始介绍老年代的垃圾收集器。第一个，Serial Old 收集器。