1.垃圾收集器判断对象死亡、存活的方法

2.java引用概念的分类

强引用：在程序代码中普遍存在，类似于Object obj=new Object()这类的引用，只要强引用还存在，垃圾收集器永远不会回收掉被引用的对象

软引用：用来描述一些还有用但并非必需的对象。对于软引用关联的对象，在系统将要发生内存溢出异常之前，将会把这些对象列进回收范围之中的进行第二次回收。提供SoftReference类来实现软引用

弱引用：也是用于描述非必需对象，强度比软引用还弱，当垃圾收集器工作时，无论当前内存是否足够，都会回收掉被弱引用关联的对象。提供WeakReference类来实现软引用

虚引用：最弱的引用关系。为一个对象设置虚引用关联的目的只有一个就是能在这个对象被垃圾收集器回收时收到一个系统通知

3.判断一个对象是否真正死亡需要经历两次标记过程，标记过程分别如下：

(1)第一次标记：如果对象在进行可达性分析后发现没有与GC Roots相连接的引用链，那它将会被第一次标记，并且执行一次筛选，筛选的条件是此对象是否有必要执行finalize()方法，如果这个对象被判定为有必要执行finalize()方法，那么这个对象将会被放置在一个叫做F-Queue的队列之中，并在稍后由一个由虚拟机自动建立的、地优先级的Finalizer线程去执行它。

(2)第二次标记：finalize()方法是对象逃脱死亡命运的最后一次机会，稍后GC将对F-Queue中的对象进行第二次小规模的标记，如果要在finalize()中成功拯救自己，只要重新与引用链上的任何一个对象建立关联即可，那么第二次标记时它将被移出即将被回收的集合

4.回收方法区

.

5.垃圾收集算法分类

现在的商业虚拟机都采用这种手机算法来回收新生代，但是并不是按照1:1比例来划分的，而是将内存分为一块较大的Eden空间和两块较小的Survivor空间，每次使用Eden和其中一块Survivor，回收时，将Eden和Survivor中还存活着的对象一次性复制到另外一个Survivor块上，最后清理掉使用过的Eden和Survivor空间

6.垃圾收集器

CMS收集器与G1收集器的介绍略

7.内存分配与回收策略

**Minor GC与Major GC的区别**

1. 新生代GC（Minor GC）：发生在新生代的垃圾收集动作，因为java对象大多数都具备朝生夕灭的特性，所以Minor GC非常频繁，一般回收速度也比较快。
2. 老年代GC（Major GC/Full GC）:只发生在老年代的GC，出现了Major GC，经常会伴随至少一次的Minor GC，Major GC的速度一般会比Minor GC慢十倍以上

**长期存活的对象将放在老年代**

为了做到这一点。虚拟机将给每个对象定义一个对象年龄(Age)计数器。如果对象在Eden出生并且经过第一次Minor GC后仍然存活，并且能被Survivor容纳的话，将被移动到Survivor空间中，并且对象年龄设为1,。对象在Survivor区中每熬过一次Minor GC，年龄就会增加1岁，当它的年龄增加到一定程度，就会被晋升到老年代中。

**规则4：**动态年龄判定。虚拟机并不是永远地要求对象的年龄必须达到MaxTenuringThreshold设定值才能晋升到老年代，如果在Survivor空间中相同年龄所有对象大小的总和大于Survivor空间的一半，年龄大于或等于该年龄的对象就可以直接进入老年代

**规则5：**空间分配担保。