垃圾回收机制的主要解决问题

1.哪些内存需要回收？

2.什么时候回收？

3.如何回收？

针对问题1——垃圾收集器会对堆进行回收前，确定对象中哪些是“存活”，哪些是“死亡”（不可能再被任何途径使用的对象）

判断方法：

1、引用计数算法

每当一个地方引用它时，计数器+1；引用失效时，计数器-1；计数值=0——不可能再被引用。

//举例：

Test test1 = new Test();

Test test2 = new Test();

test1.obj = test2;

test2.obj = test1;

//test1 ，test12能否被回收？

System.gc();

1

2

3

4

5

6

7

查看运行结果，会发现并没有因为两个对象互相引用就没有回收，因此引用计数算法很难解决对象之间相互矛盾循环引用的问题。

2、可达性分析算法：

向图，树图，把一系列“GC Roots”作为起始点，从节点向下搜索，路径称为引用链，当一个对象到GC Roots没有任何引用链相连，即不可达时，则证明此对象时不可用的。

举例：一颗树有很多丫枝，其中一个分支断了，跟树上没有任何联系，那就说明这个分支没有用了，就可以当垃圾回收去烧了。

注：在Java中可作为GCRoots的对象：

1）虚拟机栈（栈帧中的本地变量表）中引用的对象；

2）方法区中类静态属性引用的对象；

3）方法区中常量引用的对象；

4）本地方法栈中JNI引用的对象。

针对问题2——什么时候回收？

即使是被判断不可达的对象，也要再进行筛选，当对象没有覆盖finalize()方法，或者finalize方法已经被虚拟机调用过，则没有必要执行；

如果有必要执行——放置在F-Queue的队列中——Finalizer线程执行。

注意：对象可以在被GC时可以自我拯救（this），机会只有一次，因为任何一个对象的finalize() 方法都只会被系统自动调用一次。并不建议使用，应该避免。使用try\_finaly或者其他方式。

针对问题3——如何回收，这就牵扯到垃圾收集算法和垃圾收集器

垃圾收集算法：

1）标记—清除算法

两个阶段：标记，清除；

不足：效率问题；空间问题(会产生大量不连续的内存碎片)

2）复制算法

将可用内存按容量分为大小相等的两块，每次都只使用其中一块；

不足：将内存缩小为了原来的一半

新生代

3）标记—整理算法

标记，清除（让存活的对象都向一端移动）

老年代

最后来讲一下垃圾回收流程。

1、新建的对象，大部分存储在Eden中

2、当Eden内存不够，就进行Minor GC释放掉不活跃对象；然后将部分活跃对象复制到Survivor中（如Survivor1），同时清空Eden区

3、当Eden区再次满了，将Survivor1中不能清空的对象存放到另一个Survivor中（如Survivor2），同时将Eden区中的不能清空的对象，复制到Survivor1，同时清空Eden区

4、重复多次（默认15次）：Survivor中没有被清理的对象就会复制到老年区（Old）

5、当Old达到一定比例，则会触发Major GC释放老年代

6、当Old区满了，则触发一个一次完整的垃圾回收（Full GC）

7、如果内存还是不够，JVM会抛出内存不足，发生oom，内存泄漏。

补充1：分代垃圾回收

Minor GC：用于清理新生代（Eden）区域，Eden区满了就会触发一次Minor GC，清理无用对象，将有用对象复制到"Survivor1"，"Survivor2"区中（这两个区，大小空间相同，同一时刻Survivor1和Survivor2只有一个在用一个为空）。

Major GC：用于清理老年代区域。

Full GC：用于清理新生代，老年代区域，成本较高，会对系统性能产生影响。

补充2：

在对JVM调优的过程，很大一部分工作就是对于Full GC调节。根据上述GC过程，我们可以发现年老代被写满,System.gc()被显示调用，上一次GC之后Heap的各域分配策略动态变化都可能导致Full GC。

开发中容易造成内存泄漏的操作：

创建大量无用对象：比如需要大量连接字符串时，使用String而不是StringBulider/StringBuffer/线程池。

静态集合类的使用：HashMap，Vector，List

各种连接对象未及时释放关闭

监听器的使用