1. 垃圾收集器
2. 为什么需要垃圾收集

垃圾收集的历史比Java要久很多，虽然Java虚拟机实现了自动回收内存，我们还是要学习垃圾收集。因为当我们需要排查各种内存溢出，内存泄露问题时，当垃圾收集称为高并发系统的瓶颈时，我们需要监控垃圾收集的细节并调节。

程序计数器，虚拟机栈，本地方法栈三个区域随着线程而生，随线程而消失，这几个区域的内存分配和回收是确定的，因此不需要过多地考虑如何回收的问题。Java堆和方法区这两个区域是不确定的，内存分配和回收是动态的，因此垃圾收集器关注的是这两个区域的内存回收。

1. 判断对象是否存活
2. 引用计数算法

这是传统的判断算法，在对象中添加一个引用计数器，每当有一个地方引用它时，计数器值+1；当引用失效时，计数器值-1；若计数器值为0，说明没有地方引用这个对象，可以回收。

这种算法原理简单，判定效率也很高，但是对于很多情况无法处理，如对象之间相互循环引用的情况，两个对象的计数器永远不会为0，永远不会被回收。因此，Java虚拟机中采用的是可达性算法。

1. 可达性算法

通过一系列称为“GC Roots”的根对象作为起始节点集，从这些节点开始根据引用关系向下搜索，搜索所走过的路径称为“引用链”，如果这个对象到GC Roots 间没有任何引用链，则说明此对象是不会被使用的。

固定可作为GC Roots的对象有如下几种：

1. 虚拟机栈（栈帧中的本地变量表）中引用的对象，如方法中的参数，局部变量
2. 方法区中类静态属性引用的对象，如Java类的引用类型静态变量
3. 方法区中常量引用的对象，如字符串常量池里的引用
4. 本地方法栈中JNI（Native方法）引用的对象
5. Java虚拟机内部的引用，如基本数据类型对应的Class对象
6. 所有被同步锁持有的对象
7. 反应Java虚拟机内部情况的JMXBean，JVMTI中注册的回调，本地代码缓存

还有根据用户所选用的垃圾回收器以及当前回收内存区域的不同，还有其他对象临时加入GC Roots集合中，

1. 引用类型
2. 强引用

传统的引用，即该引用对象存储的是另外一块内存的起始地址，只要强引用关系在，则垃圾收集器就不会回收被引用的对象。

1. 软引用

软引用用来描述一些非必须的对象，只被软引用关联着的对象，在系统将要发生内存溢出异常前，会把这些对象列入回收范围中进行第二次回收，如果回收效果依然不行，则抛出内存溢出异常。JDK1.2后提供SoftReference类实现软引用。

1. 弱引用

描述非必须的对象，若引用关联的对象只能存活到下一次垃圾收集，JDK1.2后提供了WeakReference类实现弱引用。

1. 虚引用

也称为幽灵引用，不会对其生存时间产生影响，也无法通过虚引用获得一个对象实例。为一个对象设置虚引用关联的唯一目的是为了能在这个对象被回收时收到一个系统通知，PhantomReference类实现虚引用。

1. 对象死亡前的垂死挣扎

当对象被判定为不可达对象时，也不是立刻被回收的，真正宣告对象死亡，需要经过两次标记过程。对象经过可达性分析被判定为不可达是第一次标记，随后对象会被筛选，条件是此对象是否有必要执行finalize()方法。若对象没有覆盖finalize()方法或者该方法已经调用了，则该对象没有必要执行finalize()方法。若这个对象被判定为有必要执行finalize()方法，则该对象会被放置在一个F-Queue队列中，等待执行finalize()方法，只要该对象在finalize()方法中重新与引用链上的任何一个对象建立关联，则该对象将会被移出“即将回收”集合。

注意，每个对象的finalize()只能被调用一次，但是，这个方法官方不建议使用，因为运行代价高，不确定大，无法保证各个对象的调用顺序。

1. 回收方法区

在《Java虚拟机规范》中不要求虚拟机在方法区中实现垃圾回收，因为方法区垃圾收集的效率很低，它需要通过严苛的条件判断，才能够回收。方法区的垃圾回收主要回收两部分内容：废弃的常量和不再使用的类型，通过判断常量池中常量是否被引用来回收常量，而对于类型的收集，需要通过严苛的三个条件判断：

1. 该类以及任何派生子类的实例都已经被回收
2. 加载该类的类加载器已经被回收，这个条件很难达成
3. 该类对应的java.lang.Class对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法

再者，满足这三个条件后，仅仅是说Java虚拟机被允许对这个类型进行回收，而不是必然被回收，HotSpot提供了-Xnoclassgc参数控制是否对类型进行回收，还可以使用-XX:+TraceClassLoading参数查看类加载信息，通过-XX:+TraceClassUnLoading查看类卸载信息。

在大量使用反射，动态代理，CGLib等字节码框架的场景中，都需要Java虚拟机具有类型卸载的能力，保证不会对方法区造成内存压力。