一. 什么是垃圾回收

曾几何时，内存管理是程序员开发应用的一大难题。传统的系统级编程语言（主要指C/C++）中，程序员必须对内存小心的进行管理操作，控制内存的申请及释放。稍有不慎，就可能产生内存泄露问题，这种问题不易发现并且难以定位，一直成为困扰开发者的噩梦。如何解决这个头疼的问题呢?

过去一般采用两种办法：

- 内存泄露检测工具。这种工具的原理一般是静态代码扫描，通过扫描程序检测可能出现内存泄露的代码段。然而检测工具难免有疏漏和不足，只能起到辅助作用。(Valgrind)
- 智能指针。这是c++中引入的自动内存管理方法，通过拥有自动内存管理功能的指针对象来引用对象，是程序员不用太关注内存的释放，而达到内存自动释放的目的。这种方法是采用最广泛的做法，但是对程序员有一定的学习成本（并非语言层面的原生支持），而且一旦有忘记使用的场景依然无法避免内存泄露。

为了解决这个问题，后来开发出来的几乎所有新语言（java，python，php等等）都引入了语言层面的自动内存管理 – 也就是语言的使用者只用关注内存的申请而不必关心内存的释放，内存释放由虚拟机（virtual machine）或运行时（runtime）来自动进行管理。而这种对不再使用的内存资源进行自动回收的行为就被称为垃圾回收。

二. 常见的垃圾回收算法

1. **引用计数**

这是最简单的一种垃圾回收算法，和之前提到的智能指针异曲同工。对每个对象维护一个引用计数，当引用该对象的对象被销毁或更新时被引用对象的引用计数自动减一，当被引用对象被创建或被赋值给其他对象时引用计数自动加一。当引用计数为0时则立即回收对象。

这种方法的优点是实现简单，并且内存的回收很及时。这种算法在内存比较紧张和实时性比较高的系统中使用的比较广泛，如ios cocoa框架，php，python等。简单引用计数算法也有明显的缺点：

- 频繁更新引用计数降低了性能。一种简单的解决方法就是编译器将相邻的引用计数更新操作合并到一次更新；还有一种方法是针对频繁发生的临时变量引用不进行计数，而是在引用达到0时通过扫描堆栈确认是否还有临时对象引用而决定是否释放。等等还有很多其他方法，具体可以参考这里。
- 循环引用问题。当对象间发生循环引用时引用链中的对象都无法得到释放。最明显的解决办法是避免产生循环引用，如cocoa引入了strong指针和weak指针两种指针类型。或者系统检测循环引用并主动打破循环链。当然这也增加了垃圾回收的复杂度。

2. **标记清除**

该方法分为两步，标记从根变量开始迭代得遍历所有被引用的对象，对能够通过应用遍历访问到的对象都进行标记为“被引用”；标记完成后进行清除操作，对没有标记过的内存进行回收（回收同时可能伴有碎片整理操作）。

这种方法解决了引用计数的不足，但是也有比较明显的问题：每次启动垃圾回收都会暂停当前所有的正常代码执行，回收是系统响应能力大大降低！当然后续也出现了很多mark&sweep算法的变种（如三色标记法）优化了这个问题。

3. **分代收集**

经过大量实际观察得知，在面向对象编程语言中，绝大多数对象的生命周期都非常短。分代收集的基本思想是，将堆划分为两个或多个称为 代（generation）的空间。新创建的对象存放在称为 新生代（young generation）中（一般来说，新生代的大小会比 老年代小很多），随着垃圾回收的重复执行，生命周期较长的对象会被 提升（promotion）到老年代中。因此，新生代垃圾回收和老年代垃圾回收两种不同的垃圾回收方式应运而生，分别用于对各自空间中的对象执行垃圾回收。新生代垃圾回收的速度非常快，比老年代快几个数量级，即使新生代垃圾回收的频率更高，执行效率也仍然比老年代垃圾回收强，这是因为大多数对象的生命周期都很短，根本无需提升到老年代。

三. Go的GC工作机制

go语言垃圾回收总体采用的是经典的 **标记清除** 算法。

标记阶段:

- 在内存堆中（由于有的时候管理内存页的时候要用到堆的数据结构，所以称为堆内存）存储着有一系列的对象，这些对象可能会与其他对象有关联（references between these objects）
- a tracing garbage collector 会在某一个时间点上停止原本正在运行的程序，之后它会扫描runtime已经知道的的object集合（already known set of objects），通常它们是存在于stack中的全局变量以及各种对象。
- gc会对这些对象进行标记，将这些对象的状态标记为可达，从中找出所有的，从当前的这些对象可以达到其他地方的对象的reference，并且将这些对象也标记为可达的对象，这一步的主要目的是用于获取这些对象的状态信息。

清除阶段:

- 一旦将所有的这些对象都扫描完，gc就会获取到所有的无法reach的对象（状态为unreachable的对象），并且将它们回收
- gc仅仅搜集那些未被标记为可达（reachable）的对象。如果gc没有识别出一个reference，最后有可能会将一个仍然在使用的对象给回收掉，就引起了程序运行错误。

四. Go触发GC机制

1. 当前申请的内存是否大于上次GC后内存的2倍，如果是则触发GC
2. 监控线程发现上次GC的时间已经超过2分钟了，触发GC

五. Go GC流程

1. stop the world：等待所有的M休眠，此时所有的业务逻辑代码都停止
2. 标记：扫描所有对象集合，标记哪些对象不大达
3. 清理：启动一个goroutinne去清理那些不可达的对象
4. start the world：唤醒所有的M（M指的是os线程）