1. **关于垃圾回收**

      在.NET Framework中,内存中的资源(即所有二进制信息的集合)分为"托管资源"和"非托管资源".托管资源必须接受.NET Framework的CLR(通用语言运行时)的管理(诸如内存类型安全性检查),而非托管资源则不必接受.NET Framework的CLR管理. 需要手动清理垃圾（显式释放）。

      托管资源在.NET Framework中又分别存放在两种地方: "堆栈"和"托管堆"(以下简称"堆");规则是,所有的值类型(包括引用和对象实例)和引用类型的引用都存放在"堆栈"中,而所有引用所代表的对象实例都保存在堆中。在C#中,释放托管资源是可以自动通过"垃圾回收器"完成的(注意,"垃圾回收"机制是.NET Framework的特性,而不是C#的).

      在C++时代，我们需要自己来管理申请内存和释放内存. 于是有了new, delete关键字. 还有的一些内存申请和释放函数(malloc/free). C++程序必须很好地管理自己的内存, 不然就会造成内存泄漏(Memory leak). 在.net时代, 微软为开发人员提供了一个强有力的机制--垃圾回收. 垃圾回收机制是CLR的一部分, 我们不用操心内存何时释放, 我们可以花更多精力关注应用程序的业务逻辑. CLR里面的垃圾回收机制用一定的算法判断某些内存程序不再使用，回收这些内存并交给我们的程序再使用.

1. 2

**垃圾回收的功能**

     1、用来管理托管资源和非托管资源所占用的内存分配和释放。

     2、寻找不再使用的对象，释放其占用的内存, 以及释放非托管资源所占用的内存。

     3、垃圾回收器释放内存之后, 出现了内存碎片, 垃圾回收器移动一些对象, 以得到整块的内存，同时所有的对象引用都将被调整为指向对象新的存储位置。

1. 3

**回收内存的模式**

**在.net中提供三种模式来回收内存资源：dispose模式，finalize方法，close方法。**

     1、dispose提供了一种显示释放内存资源的方法。dispose调用方法是:要释放的资源对象.dispose().

     2、finalize方法是.net的内部的一个释放内存资源的方法。这个方法不对外公开，由垃圾回收器自己调用。

     3、close和dispose其实一样，只不过有的对象没有提供dispose的方法，只提供了close方法，而close其实在那个对象的类中，依然是调用了一个私有的dispose方法，而finalize其实也是调用一个不对外公开的dispose方法。

1. 4

**回收一般过程**

     1、垃圾回收时机：托管堆满了，内存分配即将不足时，0代内存分配满了，或其他情况，微软没有公开该部分算法。程序员可以手动调用GC.Collect()，但是会有警告，微软并不建议这么做。

     2、垃圾确认：通过根来寻找可达的对象（以后添加），并做标记，然后回收没有标记的对象。

     3、垃圾回收：内存回收，对于实现了Finalize方法的对象请参考最上面1的介绍。

     4、内存转移，合并。垃圾回收后使得内存不连续，零碎，.Net会将利用的内存合并为连续的块，然后更新对象的指针。

1. 5

**注意的地方**

     1、值类型(包括引用和对象实例)和引用类型的引用其实是不需要什么"垃圾回收器"来释放内存的,因为当它们出了作用域后会自动释放所占内存(因为它们都保存在"堆栈"中,学过数据结构可知这是一种先进后出的结构);

     2、只有引用类型的引用所指向的对象实例才保存在"堆"中,而堆因为是一个自由存储空间,所以它并没有像"堆栈"那样有生存期("堆栈"的元素弹出后就代 表生存期结束,也就代表释放了内存),并且非常要注意的是,"垃圾回收器"只对这块区域起作用;

     3、"垃圾回收器"也许并不像许多人想象的一样会立即执行(当堆中的资源需要释放时),而是在引用类型的引用被删除和它在"堆"中的对象实例被删除中间有 个间隔,为什么呢? 因为"垃圾回收器"的调用是比较消耗系统资源的,因此不可能经常被调用!(当然,用户代码可以用方法System.GC.Collect()来强制执行"垃圾回收器")

    4、有析构函数的对象需要垃圾收集器两次处理才能删除：第一次调用析构函数时，没有删除对象，第二次调用才真正删除对象。

    5、由于垃圾收集器的工作方式，无法确定C#对象的析构函数何时执行。

    6、可实现IDisposable接口的Dispose()来显示释放由对象使用的所有未托管资源。

    7、垃圾收集器在释放了它能释放的所有对象后，就会压缩其他对象，把他们都移动回heap的端部，再次形成一个连续的块。