**垃圾回收器GC**

**GC是.NET运行库的内置垃圾回收器（程序），它专门回收托管堆上的可回收对象，一般程序员无需干预这一自动过程，除非以下情况需要我们手动强制GC进行回收：**

* 应用程序将要进入一段代码，且该代码不想被GC回收中断；
* 应用程序分配非常多对象后，尽可能多删除已获得的内存。
* **GC有什么用？**

回收托管堆上的垃圾内存。

* **GC回收发生在什么时候？**

GC回收发生在任何时候。当可用内存不能满足新对象的创建存储，

GC将会启动一次回收，而此时刻是未知的，随时发生。

* **GC回收的影响是什么？**
* 释放可回收对象的内存，并压缩原有内存空间，腾出的未使用内存提供新对象的存储空间；
* 未被回收的旧对象的内存地址可能被改变，因此指向它们的引用将重新赋值新地址；
* 每次回收发生后，旧对象的代数递增加一，除非它的代数已最高。
* **GC为何有高效的回收机制？**
* GC每次回收都会试图检查最低代数的对象是否可以回收，因为最低代数的对象一般是最近创建的局部本地变量，因此它们的生命周期较短，而代数越大的对象更应该被保留，如程序窗体对象。除非某一代回收完后，仍需较多内存，则GC会继续尝试回收上一级的代对象。
* GC使用两个不同的堆，一个存储非常大的对象，另一个存储小对象。一般GC很少会回收前者，因为重新定位大对象的系统开销很大。
* .NET会使用两个后台垃圾回收线程负责回收不同代对象，一个用于暂时代对象（0、1代），一个用于非暂时代对象（2代）。
* **什么是对象的代？**

对象的代是用于GC高效回收机制的变量。.NET会给创建的对象

增加代数变量用来标志该对象的生命周期，一般从0代开始。每次GC回收后，保留下来的对象代会递增加一，除非代数已最大，一般为2代。

代数越高的对象，GC认为它们更应该保留，因此GC会尽可能更快的释放刚创建并已无用的低代数对象内存。

* **GC如何判断对象已可回收？**

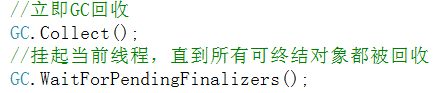
GC会检查对象是否仍有程序的根（即对象的引用，其存在于栈中），

若无，此时该对象已从对象图（可以的对象及对象间的依赖关系图）中删除，GC就认为该对象可被回收删除。

* **手动强制进行一次GC回收**

**通过使用GC类的静态成员，我们可以让GC立即进行一次垃圾回**

**收。**

****

* **托管堆和栈有什么区别？**
* 托管堆存放引用类型的变量，其垃圾回收由GC处理。回收是不可预期的，因为GC可能在一次回收处理中不释放其内存。
* 栈存放值类型变量和引用类型对象的引用，其垃圾回收由内存处理。回收是可预期的，因为当程序执行完该变量的作用域后，内存会自动释放。
* **什么是非托管对象？**

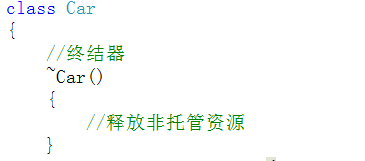
非托管对象一般是操作系统API和COM对象，如数据库、文件、

网络连接等。 GC无法回收非托管对象，因此，如果一个托管对象中有非托管对象成员，则需要手动释放非托管对象。否则，非托管对象可能永远存在内存中，导致内存泄露。

* **什么是终结器？**

终结器就是类的析构函数，其定义了对象被释放时需要的操作（主

要是释放非托管资源）。

****

终结器实际上是重写虚方法System.Object.Finalize()，该方

法默认空操作，而析构函数是重写该方法的唯一合法语法。

* **GC回收时，被回收对象发生什么？**

被回收对象会调用终结器，用来释放对象内部的成员。终结器默

认是空操作，因为一般的托管对象不包含非托管资源，否则，需要自定义终结器来释放非托管资源。终结器就是析构函数，编译器会将析构函数转化为Finalize方法的细节。

* **谁可以自定义终结器？**

只能是类，结构不行，因为结构不能有析构函数。

* **如何自定义终结器？**

重写类的析构函数。

* **自定义终结器有什么用？**
* 自定义终结器的类对象被CRL标记为可终结对象，并赋予指向对象的指针，该指针保存在由GC维护的终结队列表中。
* 终结器可以自定义释放非托管资源的方法。
* 每个可终结对象至少经历两次GC回收。

第一次GC回收时，GC确定可终结对象的回收时间后，将该对象复制到一个叫终结可达表的托管结构上。

下次垃圾回收，GC会开辟线程调用可终结对象非Finalize()方法，即析构函数。

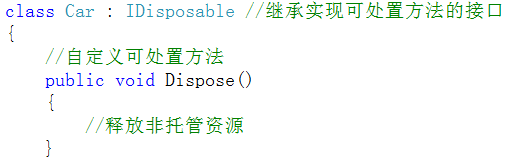
* **如何释放结构里面的非托管资源？**

由于结构无法定义终结器，故只能定义可处置方法Dispose()。

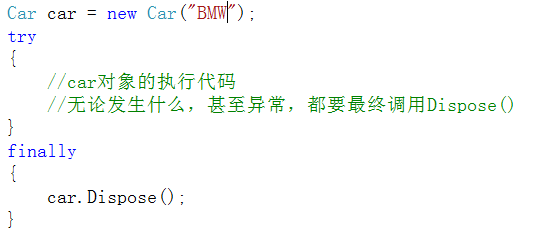
* **什么是可处置对象**？

可处置对象是继承接口IDisposable的对象，实现该接口的类必

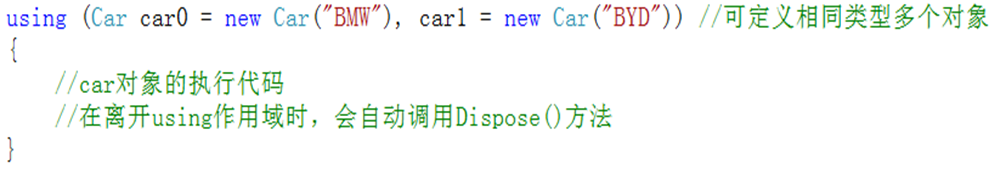
须自定义可处置方法Dispose()。



* **为什么要实现可处置方法？**
* 结构无终结器，故需要Dispose()方法释放非托管资源。
* 由于GC的回收不确定，故可终结对象的终结器何时被调用也是不确定的。因此，若要立即释放非托管资源，需要调用Dispose()方法立执行。
* 构建既可终结又可处置的类对象，对释放非托管资源起到双重保障。
* **可处置方法正式的调用模式**



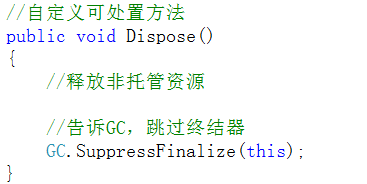
上面代码可以通过using关键词简化过程：



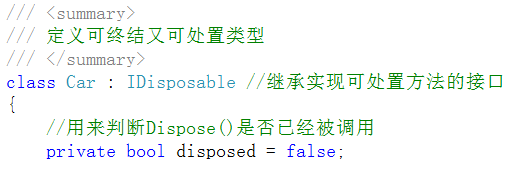
* **为什么要构建即可终结又可处置类型**

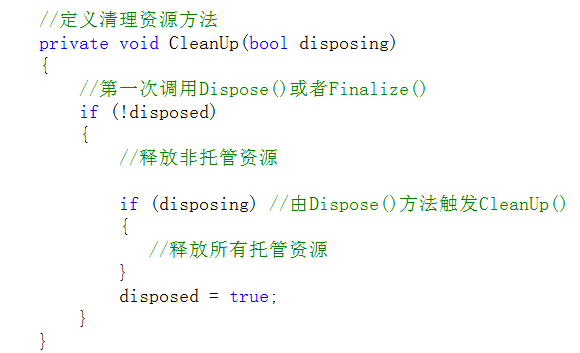
当类成员存在非托管资源时，我们最好同时定义终结器方法

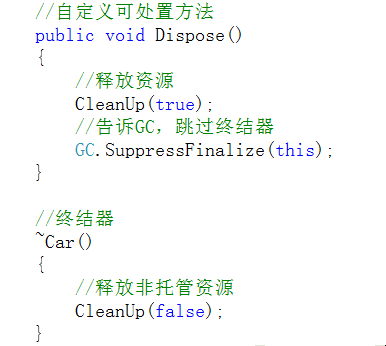
Finalize()和可处置方法Dispose()。这是因为，如果我们忘记了调用Dispose()方法，则GC会在回收该对象时，自动调用终结器释放非托管资源。但如果我们调用了Dispose()方法（该对象此刻一定未过期，否则编译错误。也就是说，GC回收该对象是调用Dispose()以后很久的事了），则可以立即释放非托管资源。因为已经释放了非托管资源，所以终结器已无再调用的意义（终结器就是为了释放非托管资源，因为其他托管资源和栈数据系统自动释放），可在Dispsoe()方法里面声明以下方法，告诉GC跳过对该对象调用终结器。



* **构建即可终结又可处置类型的微软模式**

****

****

****

注意，在由Dispose()触发的垃圾清理逻辑中，多了“释放所有托管资源”的操作，里面主要是调用托管对象成员里面的Dispose()方法来释放非托管资源，而由终结器触发的垃圾清理逻辑中，则不允许这么做，因为托管对象成员可能已被回收，调用其Dispsoe()会发生异常错误。