关于我们组之前出现的一个问题：**窗体1通过ShowDialog方式显示窗体2，同时窗体2内部开启一个System.Windows.Forms.Timer定时器执行定时任务，当窗体2通过Close方法关闭之后，定时器仍在工作。**

这个问题最后应该是通过在窗体2关闭的时候手动调用窗体的Dispose()方法解决的，因为问题出现时的第一感觉是只调用Close方法，窗体的资源并没有完全释放。之前跟超哥讨论，**如果问题真是因为窗体没有及时释放定时器引起的，那么至少应该在垃圾回收的时候得到释放——窗体的Dispose方法得到调用**（准确说应该是析构函数得到调用，从而调用Dispose），如果连这个都不能满足，那么就可能产生内存泄漏的问题，毕竟在窗体2关闭了之后，我们自己的代码里并没有任何引用指向它。为了验证这个想法，我在窗体2关闭之后，在窗体1里调用GC.Collect()强制进行垃圾回收，但得到的结果是窗体2对象并不会被回收！

再结合上午无意间发现的另一个区别，当使用Show()方法（而不是ShowDialog）显示窗体2的时候，并不会出现上面的问题。当时还差点真以为是因为ShowDialog引起的：调用ShowDialog导致窗体1（父窗体）无形中保持有对窗体2对象的引用，所以才无法回收窗体2。但是后来做了测试发现这个猜测并不正确：在不开启定时器的情况下，采用ShowDialog显示窗体2，窗体2在关闭后是可以被回收的！

上面的描述其实产生了两个问题：

1. 使用ShowDialog显示窗体，并且同时开启定时器的情况下，当窗体Close掉后为什么垃圾回收器无法回收它；
2. 窗体的ShowDialog和Show有什么区别；

第一个问题的答案最后在Timer的[Enabled](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://System.Windows.Forms:4.0.0.0:b77a5c561934e089/System.Windows.Forms.Timer/property:Enabled:Boolean)属性代码里找到（这个属性会由Start和Stop方法调用）：



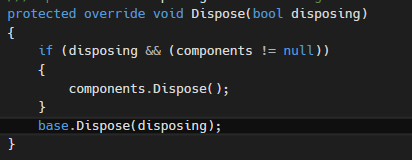
关键在于System.Runtime.InteropServices.GCHandle结构的静态方法：Alloc(object)。文档对这个方法的解释是：**为指定的对象分配一个GC句柄，保证它不会被垃圾回收器回收**。说到这儿我又要很不要脸地装下X啦~（^\_^）：CLR的垃圾回收器是一种标记清除算法，它首先从一组被称作“根”的对象开始，这些“根对象”是可以被代码直接访问到的，所以直接被标记。然后标记对象内部所有实例字段引用的对象（如果不为null），如此递归处理所有这些对象及其引用的对象，这一步称为标记。第二步是回收所有没有被标记的对象，因为这些对象是无法通过“根对象”直接或间接访问到的（对象不可达）。最常见的“根对象”有**引用类型的静态字段**和**方法栈上引用类型的局部变量。**可能有人会觉得对象内部引用类型的实例字段可以算作“根对象”，不好意思，这些对象实际上不能被直接访问到，而要通过this访问，而this属于方法栈上的局部变量。

言归正传，根据文档的解释，System.Runtime.InteropServices.GCHandle结构的Alloc静态方法相当于告诉垃圾回收器将参数指定的对象当作一个“根对象”来看待，被该对象直接或间接引用的对象将不能被垃圾回收器回收。Timer的Start方法内部会将Enabled属性设置为true，根据Enabled的实现代码可以知道即使将来我们自己的代码里不再有任何地方持有对Timer的引用了，Timer对象本身也不能被垃圾回收器回收，除非事先调用Stop方法或将Enabled设为false。现在第一个问题就能解释得通了：**不是窗体2无法释放定时器，而是定时器保持有对窗体2对象的引用从而导致窗体2无法回收**。

也许你会问为什么定时器会持有窗体2对象的引用，这是因为在出现问题的代码里，定时器的Tick事件注册的是窗体2的一个实例方法，此时事件内部的委托会持有窗体2的引用（具体可以查看所有委托的基类System.Delegate的[\_target](http://127.0.0.1/roeder/dotnet/Default.aspx?Target=code://mscorlib:4.0.0.0:b77a5c561934e089/System.Delegate/_target:Object)字段以及它的几个构造函数）。而如果定时器注册的是窗体2的一个静态方法，此时事件内部的委托不需要引用窗体2对象，所以窗体2对象不受定时器的影响，可以被回收（建议可以做一下测试）。

这其实揭示了一个更严重的问题：内存泄露。如果窗体2内部有很多控件，并且绑定了很多数据，那么这些对象和数据都不能被垃圾回收！

对于第二个问题我也做了相关的测试，但是并没有得出一个很确定的答案。我发现如果采用ShowDialog方式显示窗体2，在调用Close方法关闭后，窗体2的Dispose(bool)方法并不会得到调用，而如果使用Show方法显示的话，Dispose(bool)会被调用，并且传递的参数disposing为true：



从窗体的Dispose(bool)可以看到此时窗体内部所有组件类型的控件都会得到释放。这解释了为什么ShowDialog和Show会导致定时器的不同行为：前者在Close后并不会释放定时器，而后者会通过调用Dispose(true)释放定时器。至于为什么一个会调用而另一个不会调用就不得而知了（>\_<），这可能得理解模态显示和普通显示更深层次的区别。

反思：

最近我们的软件出现很多问题，虽然我没有参与其中修复这些问题，但是在平时也经常听大家讨论各种问题，随之产生的一个思考是：我们软件中出现的很多bug是不是反映了我们之前开发过程中出现了比较大的规范性问题，换句话说是不是因为很多不规范或不合理的做法导致软件出现了某些bug。我们往往在出现问题的时候会太过关注于问题本身和当下问题的解决方案，而不会退一步思考我们原先的做法是不是合理。比如上面的Timer使用，在显示Form2的时候开启定时器，却并没有在关闭窗体的时候事先调用Stop停止定时器，是否可以认为这是个规范或者合理性的问题？如果我们遵循定时器的这个使用规范，根本不会出现上面的问题，即使我们不了解上面描述的问题的原因。这话说得其实有点事后诸葛亮了，因为在查找问题原因的时候我也没有想到要Stop定时器，最多想到的是应当先注销定时器的Tick事件，不管怎样至少这个反思我觉得是有意义的。