C 编程人员容易犯的10个C#错误2 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/134/2021_2022_C___E7_BC_96_E7_A8_8B_E4_c97_134346.htm 错误2：Finalize和Dispose使用谁？ 从上面的论述中我们已经很清楚，显性地调用finalizer是不允许的，它只能被碎片收集程序调用。如果希望尽快地释放一些不再使用的数量有限的非可管理性资源（如文件句柄），则应该使用IDisposable界面，这一界面有个Dispose方法，它能够帮你完成这个任务。Dispose是无需等待Finalize被调用而能够释放非可管理性资源的方法。 如果已经使用了Dispose方法，则应当阻止碎片收集程序再对相应的对象执行Finalize方法。为此，需要调用静态方法GC.SuppressFinalize，并将相应对象的指针传递给它作为参数，Finalize方法就能调用Dispose方法了。据此，我们能够得到如下的代码： public void Dispose() { // 完成清理操作 // 通知GC不要再调用Finalize方法 GC.SuppressFinalize(this). } public override void Finalize() { Dispose(). base.Finalize(). } 对于有些对象，可能调用Close方法就更合适（例如，对于文件对象调用Close就比Dispose更合适），可以通过创建一个private属性的Dispose方法和public属性的Close方法，并让Close调用Dispose来实现对某些对象调用Close方法。 由于不能确定一定会调用Dispose，而且finalizer的执行也是不确定的（我们无法控制GC会在何时运行），C#提供了一个Using语句来保证Dispose方法会在尽可能早的时间被调用。一般的方法是定义使用哪个对象，然后用括号为这些对象指定一个活动的范围，当遇到最内层的括号时，Dispose方法就会被自动调用，对该对象进行处理。

using System.Drawing. class Tester { public static void Main() { using (Font theFont = new Font("Arial", 10.0f)) { //使用theFont对象 } // 编译器将调用Dispose处理theFont对象 Font anotherFont = new Font("Courier",12.0f). using (anotherFont) { // 使用anotherFont对象 } // 编译器将调用Dispose处理anotherFont对象 } } 在本例的第一部分中，Font对象是在Using语句中创建的。当Using语句结束时，系统就会调用Dispose，对Font对象进行处理。在本例的第二部分，Font对象是在Using语句外部创建的，在决定使用它时，再将它放在Using语句内，当Using语句结束时，系统就会调用Dispose。 Using语句还能防止其他意外的发生，保证系统一定会调用Dispose。 