<http://www.cnblogs.com/luminji/archive/2011/03/29/1997812.html>

[改善C#程序的建议4：C#中标准Dispose模式的实现](http://www.cnblogs.com/luminji/archive/2011/03/29/1997812.html)

2011-03-29 08:50 by Luminji, 8647 阅读, 32 评论, [收藏](http://www.cnblogs.com/luminji/archive/2011/03/29/1997812.html), [编辑](http://www.cnblogs.com/luminji/admin/EditPosts.aspx?postid=1997812)

需要明确一下C#程序（或者说.NET）中的资源。简单的说来，C#中的每一个类型都代表一种资源，而资源又分为两类：

托管资源：由CLR管理分配和释放的资源，即由CLR里new出来的对象；

非托管资源：不受CLR管理的对象，windows内核对象，如文件、数据库连接、套接字、COM对象等；

毫无例外地，如果我们的类型使用到了非托管资源，或者需要显式释放的托管资源，那么，就需要让类型继承接口IDisposable。这相当于是告诉调用者，该类型是需要显式释放资源的，你需要调用我的Dispose方法。

不过，这一切并不这么简单，一个标准的继承了IDisposable接口的类型应该像下面这样去实现。这种实现我们称之为Dispose模式：

[复制代码](javascript:void(0);)

public class SampleClass : IDisposable  
 {  
 //演示创建一个非托管资源  
 private IntPtr nativeResource = Marshal.AllocHGlobal(100);  
 //演示创建一个托管资源  
 private AnotherResource managedResource = new AnotherResource();  
 private bool disposed = false;  
  
 /// <summary>  
 /// 实现IDisposable中的Dispose方法  
 /// </summary>  
 public void Dispose()  
 {  
 //必须为true  
 Dispose(true);  
 //通知垃圾回收机制不再调用终结器（析构器）  
 GC.SuppressFinalize(this);  
 }  
  
 /// <summary>  
 /// 不是必要的，提供一个Close方法仅仅是为了更符合其他语言（如C++）的规范  
 /// </summary>  
 public void Close()  
 {  
 Dispose();  
 }  
  
 /// <summary>  
 /// 必须，以备程序员忘记了显式调用Dispose方法  
 /// </summary>  
 ~SampleClass()  
 {  
 //必须为false  
 Dispose(false);  
 }  
  
 /// <summary>  
 /// 非密封类修饰用protected virtual  
 /// 密封类修饰用private  
 /// </summary>  
 /// <param name="disposing"></param>  
 protected virtual void Dispose(bool disposing)  
 {  
 if (disposed)  
 {  
 return;  
 }  
 if (disposing)  
 {  
 // 清理托管资源  
 if (managedResource != null)  
 {  
 managedResource.Dispose();  
 managedResource = null;  
 }  
 }  
 // 清理非托管资源  
 if (nativeResource != IntPtr.Zero)  
 {  
 Marshal.FreeHGlobal(nativeResource);  
 nativeResource = IntPtr.Zero;  
 }  
 //让类型知道自己已经被释放  
 disposed = true;  
 }  
  
 public void SamplePublicMethod()  
 {  
 if (disposed)  
 {  
 throw new ObjectDisposedException("SampleClass", "SampleClass is disposed");  
 }  
 //省略  
 }  
 }

[复制代码](javascript:void(0);)

在Dispose模式中，几乎每一行都有特殊的含义。

在标准的Dispose模式中，我们注意到一个以~开头的方法：

[复制代码](javascript:void(0);)

/// <summary>  
 /// 必须，以备程序员忘记了显式调用Dispose方法  
 /// </summary>  
 ~SampleClass()  
 {  
 //必须为false  
 Dispose(false);  
 }

[复制代码](javascript:void(0);)

这个方法叫做类型的终结器。提供终结器的全部意义在于：我们不能奢望类型的调用者肯定会主动调用Dispose方法，基于终结器会被垃圾回收器调用这个特点，终结器被用做资源释放的补救措施。

一个类型的Dispose方法应该允许被多次调用而不抛异常。鉴于这个原因，类型内部维护了一个私有的布尔型变量disposed：

        private bool disposed = false;

在实际处理代码清理的方法中，加入了如下的判断语句：

if (disposed)  
 {  
 return;  
 }  
 //省略清理部分的代码，并在方法的最后为disposed赋值为true  
 disposed = true;

这意味着类型如果被清理过一次，则清理工作将不再进行。

应该注意到：在标准的Dispose模式中，真正实现IDisposable接口的Dispose方法，并没有实际的清理工作，它实际调用的是下面这个带布尔参数的受保护的虚方法：

[复制代码](javascript:void(0);)

/// <summary>  
 /// 非密封类修饰用protected virtual  
 /// 密封类修饰用private  
 /// </summary>  
 /// <param name="disposing"></param>  
 protected virtual void Dispose(bool disposing)  
 {  
 //省略代码  
 }

[复制代码](javascript:void(0);)

之所以提供这样一个受保护的虚方法，是为了考虑到这个类型会被其他类继承的情况。如果类型存在一个子类，子类也许会实现自己的Dispose模式。受保护的虚方法用来提醒子类必须在实现自己的清理方法的时候注意到父类的清理工作，即子类需要在自己的释放方法中调用base.Dispose方法。

还有，我们应该已经注意到了真正撰写资源释放代码的那个虚方法是带有一个布尔参数的。之所以提供这个参数，是因为我们在资源释放时要区别对待托管资源和非托管资源。

在供调用者调用的显式释放资源的无参Dispose方法中，调用参数是true：

public void Dispose()  
 {  
 //必须为true  
 Dispose(true);  
 //其他省略  
 }

这表明，这个时候代码要同时处理托管资源和非托管资源。

在供垃圾回收器调用的隐式清理资源的终结器中，调用参数是false：

~SampleClass()  
 {  
 //必须为false  
 Dispose(false);  
 }

这表明，隐式清理时，只要处理非托管资源就可以了。

那么，为什么要区别对待托管资源和非托管资源。在认真阐述这个问题之前，我们需要首先弄明白：托管资源需要手动清理吗？不妨先将C#中的类型分为两类，一类继承了IDisposable接口，一类则没有继承。前者，我们暂时称之为非普通类型，后者我们称之为普通类型。非普通类型因为包含非托管资源，所以它需要继承IDisposable接口，但是，这个包含非托管资源的类型本身，它是一个托管资源。所以说，托管资源需要手动清理吗？这个问题的答案是：托管资源中的普通类型，不需要手动清理，而非普通类型，是需要手动清理的（即调用Dispose方法）。

Dispose模式设计的思路基于：如果调用者显式调用了Dispose方法，那么类型就该按部就班为自己的所以资源全部释放掉。如果调用者忘记调用Dispose方法，那么类型就假定自己的所有托管资源（哪怕是那些上段中阐述的非普通类型）全部交给垃圾回收器去回收，而不进行手工清理。理解了这一点，我们就理解了为什么Dispose方法中，虚方法传入的参数是true，而终结器中，虚方法传入的参数是false。

注意：我们提到了需要及时释放资源，却并没有进一步细说是否需要及时让引用等于null这一点。有一些人认为等于null可以帮助垃圾回收机制早点发现并标识对象是垃圾。其他人则认为这没有任何帮助。下一篇“[**引用类型赋值为null与加速垃圾回收**](http://www.cnblogs.com/luminji/archive/2011/04/07/2007205.html)”我们再细说这一点。