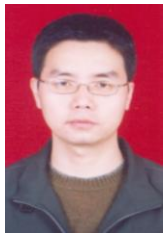
VS.NET 中内存泄露问题的研究

亢红波[1, 2] 魏 燕[3]
(1. 长安大学电子与控制工程学院，西安 710064; 2. 西安邮电大学自动化学院，西安 710121;
3. 中国电建集团西北勘测设计研究院有限公司城建与交通设计院，西安 710065）



摘要：详细分析了.net 中出现内存泄露的表现形式及可能途径，介绍了内存泄露的具体检测方法，以案例的形式重点描述了出现内存泄露问题时的处理方法。最后，针对不同的应用，提出了服务器程序性能优化方法，从不同角度提高了程序的效率与可靠性。

关键词：.net; 内存泄露; 内存管理; 溢出; CLR

Research of Memory Leak in VS.NET

Kang Hongbo[1,2] Wei Yan[3]
(1. Institute of Electrical and Control Engineering, Chang'an University, Xi'an 710064; 2. College of Automation, Xi'an University of Posts Communications, Xi'an 710121;3. Urban Construction and Transport engineering div, Powerchina Xibei Engineering Corporation Limited, Xi'an 710065)

Abstract: The paper analyses the forms and ways of memory leak in the development based on .net, and several detection methods are introduced in the memory leak debugging. In addition, handing methods of memory leak are focused on in the study combining cases. Finally, according to the different application, the methods of program optimization are discussed to improve the efficiency and reliability of server program from different ways.

Key words: .net; memory leak; memory management; overflow; CLR

1 引言

微软的.net 框架是一个可以快速开发、部署 web 程序及 windows 程序的开发平台。它实现了多语言基类库的调用，让各语言能方便调用；Web Services 方便远程不同平台的共享；支持移动开发技术；.net 构建于公共语言运行时 CLR 的基础上，其垃圾回收机制(Garbage Collector，GC)使得程序员从分配和释放内存的负担中解脱出来，可以开发更安全、健壮、可移植性强的程序，是服务器端程序开发的一大利器[1]。

内存管理工作是很多软件错误发生的根源。C++、Java 以及嵌入式环境都会发生内存泄露，这会直接影响程序的稳定性与效率，由于工作机制不同，其表现也有较大区别[2, 3]。对于服务器程序来说，内存泄露问题是非常隐蔽的问题，其表现形式是，刚开始程序运行良好，随着程序连续运行时间的增长，内存支出越来越多，且不能正常释放，最后会导致程序崩溃，服务器卡死。服务器程序通常都是比较重要的程序，而程序崩溃具有偶然性特征，故障难以排查且损失往往难以估量。因为.net 具有垃圾回收机制，所以很多开发人员在开发程序时，默认为.net 能够自动管理内存，因此对内存泄露问题存在主观性的忽视，从而造成服务器程序的大量内存泄露。.net 中内存泄露问题有两种原因：一是由于 GC 机制设计的不完善性，二是由于处理非托管资源与托管资源时存在着一定的差异[4]。事实上，因为开发人员较为关注程序的功能实现，而内存泄露在相当长时间内表现为“隐性”，当问题发生时，解决的成本及难度将大大增加，很难

作者简介：亢红波（1974–），讲师，智能交通与信息系统工程专业；研究方向：自动控制及信息集成。
收稿日期：2015-02-11

补救；同时，此类问题的研究需要较深的研发背景，且问题所涉及的方面较多，没有较为清晰的理论脉络，所以内存泄露的书面研究相对较少。

详细描述了.net 中出现内存泄露的可能途径；描述了内存泄露的表现形式；介绍了几种在调试内存泄露问题时需要的工具；并以一个案例描述了出现内存泄露问题时的处理方法。

2 .net 中出现内存泄露途径分析

内存泄露并非物理内存的丢失，指没有对已不使用的内存进行针对性释放，即放弃了对此段内存的控制。内存泄露发生有一定的主观性因素，如程序设计时的疏忽与不周，内存泄露会使程序性能下降，累积到一定程度则直接导致内存溢出错误[5]。导致内存泄露的常见原因有以下几种。

2.1 未手动退订的事件

如果涉及事件是非静态事件，绑定到事件上的对象会被释放；如果是系统静态事件，绑定到事件上的对象不会被自动释放；必须手动写上 Dispose 才能释放事件句柄。Static 的生命周期很长，永远不会被 GC 回收；内存一旦被静态事件所引用，则不可能释放。如下段程序所示[6]：

```
class CButton{
public:
void OnButtonClick(object sender, EventArgs e){
….
}
};

class Program{
static event EventHandler ButtonClick;
};

void Main(string[] args)
{
CButton button=new CButton();

Program::ButtonClick+=button.OnButtonClick;
}
```

所示代码中，静态事件 ButtonClick 的生命周期与 AppDomain 相一致，如果不对事件取消注册，所引用对象将永远伴随着进程。

2.2 非托管资源

托管资源在.net 中占大多数，常由 GC 自动释放，无需做显式资源释放，虽然 GC 执行的时机不可预知，一般不会导致明显的资源泄露问题；而一些较常用的非托管资源则需做显示释放，这些资源包括了串行口连接、数据库连接、文件操作等，否则程序运行多次则容易造成内存泄露。再者，当使用 API 时，调用了非托管资源，资源需手动释放。

2.3 静态变量

静态变量中的成员所占内存如果不手动处理是不会自动释放的。单态模式的对象也是静态的，需要特别注意。因为静态对象中的成员所占的内存不会释放，如果此成员是一个对象，同时此对象中的成员所占的内存也不会释放，以此类推，如果对象很复杂而且是静态的，就很容易造成内存泄露。

此外需注意的是 Singleton 单态模式实现的类，它们也是 static 的，生命周期很长。所以要注意引用链，如果类被它引用了，在其引用链上，就会发生内存泄露。

还有，如果对象既不是被 static 对象引用，也不是 singleton 引用，而是被一个永远不释放的对象引用着，即使这个对象不是 static，也会发生泄露问题。如界面上的 MainForm，因为 MainForm 是永远不会关闭和释放的，被它引用也就不会释放了。

2.4 Dispose 方法没被调用

Dispose 方法没被调用或 Dispose 方法没有处理对象的释放，也会造成内存泄露。

2.5 内存溢出

在进行数据查询时，如果查询结果数据量很大，达到几百万条，数据存放到 datatable 或 dataset 中也会造成内存溢出，可以采用分页查询等方法解决。

3 .net 中内存泄露的检测

如何判断程序是否发生内存泄露？“Out of memory”为较直观的常见错误，这种问题带有定位性质，找到问题点改正即可。但实际应用中，这种明显的泄露情况较少，大多是隐性的渐进式的。通常，是否发生内存泄露，不以程序实时占用内存为依据，而是给程序足够长的运行评判时间，然后判断内存的上涨趋势，未进入稳态前的内存上涨则没有评判价值。实际操作中，通常情况是关注任务管理器中进程的以下指标得出结论：句柄数、线程数、虚拟字节数、私

有字节、堆大小等。

当监测时，如果所关注指标在一个区域内来回波动，说明状态良好，如发生断续或持续的上升，则存在内存泄露情况可能性。通常这种观察的周期与程序具体应用有关，可能是几天，也可能是一个星期甚至一个月。

针对以上指标的观察，可以采用 windows 任务管理器、Process Tracker、微软与.net 开发环境集成的 Memory Profiler 等。

4 .net 中调试内存泄露的实例分析

工作中遇到的一个问题是，用 C#写的一个通信服务器程序，接收来自于设备的数据并将收到的数据通过 restful API 的形式传递给接收端。接收端的工作是将收到的数据解析后写入数据库。

当设备数量不多时，这种方式没有问题。随着设备数量的增多，会发现程序的句柄数越来越多，运行一段较长的时间后会发现程序卡死。通过 ProcessHacker 可以看到程序的句柄数，句柄数为 3665，点击 Details 按钮，在随后弹出的 Handle Statistics 框中，可以看到 File 占大多数，以上过程如图 1 所示。Socket 连接大多数以 File 的形式出现。通过 ProcessHacker 的观察，发现句柄数大多数是文件。而在 ProcessHacker 的 Network 处，则会发现 tcp 连接越来越多。可以通过 netstat 命令将所有的网络连接写入一个文本文件，从而可以具体查看各种连接的个数。在 windows 的 command 命令行中，如通过以下命令：netstat –n >c:\test.txt 可以将当前服务器上的所有网络连接状态输入到 C:\test.txt 文件中。通过文本文件可以很方便地知道目前的 tcp 连接，udp 连接个数，并容易统计分析。

通过以上分析方法，很快发现程序卡死的原因是因为 tcp 连接不断增加。可是为什么 tcp 连接不能正常释放？通过分析发现，接收端程序要操作数据库，而且是同步操作。当数据库返回时，接收端才给发送端返回 socket，而发送端只有在收到接收端的回复后，才会断开 tcp 连接。如接收端没有回复时，发送的 socket 连接就会一直存在。随着接收端堆积的 socket 越来越多，接收端处理得越来越慢，发送端堆积的 socket 就会越来越多，与 socket 相关的资源不能快速释放，造成程序的内存消耗越来越大，最终导致程序卡死。

一种常用方法为：当发送端发送数据时，同时启一个定时器，如果收到接收端的回复，就取消定时器。若在一定时间内收不到接收端的回复，导致定时器超时，在定时器的超时响应事件中主动关闭发送端 socket。这种方法可规避 socket 堆积现象的发生，从而解决由此引起的内存泄露问题。

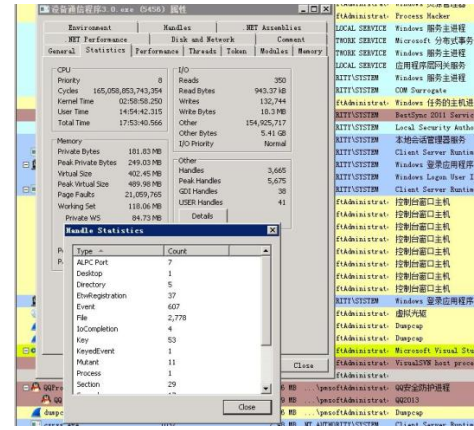


图 1 Process Hacker 中查找句柄数及句柄数的统计信息

5 解决内存泄露及服务器程序性能优化方法

5.1 事件未退订——弱引用

忘记取消注册事件在.net 的内存泄露中占有很大比例。通过弱引用，当事件所属对象被弃用时，就可以自动地取消其注册。

委托也属于类的范畴，而.net 中的委托则为强引用，如需要利用弱引用，则可以通过重新的特征重塑，创建专属 WeakDelegate 类。

事件和委托的关系与字段和属性相似，通过 add 与 remove 方法对其进行更改，则可以把外部委托转换成相己定义的委托。如下段程序所示。

```
class CButton
{
    Private:
      struct WeakDelegate
{
      Public:
        WeakReference Target;
        MethodInfo Method;
}
List<WeakDelegate>  clickSubscribers=new
List<WeakDelegate>();
```

```
Public:
 event EventHandler Click{
    add
{
    clickSubscribers.Add(new WeakDelegate{
       Target=new WeakReference(value.Target),
       Method=value.Method
});
}
remove
{
   …
}
}
void FireClick()
{
List<WeakDelegate> toRemove=new List<WeakDelegate>();
foreach(WeakDelegate subscriber in clickSubscribers)
{
    object target=subscriber.Target.Target;
    if(target==null)
    {
        toRemove.Add(subscriber);
}
    else
    {
    subscriber.Method.Invoke(target, newobject[]{this,EventArgs.Em});
}
}
clickSubscribers.RemoveAll(toRemove);
}
};
```

对象池可对对象进行重复使用，减少不必要的操作消耗，弱引用也可以创建对象池，通过对象池，可以管理若干对象，以减小对内存的开销。

对于未引用而注册了的事件，不使用时需及时手动注销，以便于资源的回收。

5.2 针对非托管资源

对于 API 形式的非托管资源，不使用时需手动释放资源，在处理大对象时需特别注意，具体描述如下：

a. 确保连接关闭

针对非托管资源，特别是 socket 连接，串口连接，数据库连接，文件操作等，在使用完后一定要保证连接能被正常关闭。对于数据库操作，尽可能用数据库连接池，以减少连接。

b. Using 的使用

Using 不但可以引用功能 D11，而且可以对对象应用范围加以限制，如超出这个范围，对象将自动被释放。比如常用的操作数据库的代码：

```
using (SqlConnection connection = new SqlConnection(connectionString))
    {
     CDataSet dataSet = new CDataSet();
     connection.Open();
     SqlDataAdapter sqlDA = new CSqlDataAdapter();
     sqlDA.SelectCommand = BuildQueryCommand(connection, storedProcName, parameters);
    sqlDA.Fill(dataSet, tableName);
    connection.Close();
    return dataSet;
    }
```

所示代码段中， connection 连接对象只在 using 后面的｛｝范围内有意义。执行完后，connection 会自动释放。在代码执行完成之前，一定要调用 connection.Close()以关闭数据库连接。

c. Dispose 的使用

Dispose()可对对象的生存期进行限定，可访问非托管对象。如某个对象提供自己的 Dispose()方法，则结束对象使用时必须加以调用；另有些对象提供对 Dispose()的扩展，可以在大对象处理时选择合适的时机提前释放；对于不提供 Dispose()的对象，如有的 COM 组件，可自行定义并调用。

d. 析构函数

析构函数在所指对象脱离其作用域后自动执行，以清理内存，一个类只有一个析构函数。对于非托管资源的释放，可自定义析构函数；对于托管资源对象则推荐使用自身所提供的 Dispose 方法，使用析构函数以免引起其它冲突错误[6]。

5.3 其它

提高服务器的性能有几个技巧：因为当程序异常时，程序的性能会大大降低，所以要尽量用程序来捕

获异常，而不是用 try catch 来捕获异常，只有程序无法预料的异常才用 try catch 来捕获，即不能滥用 try catch 机制；日志输出过于频繁也会增加程序的 I/O，所以在程序正常运行时，尽可能减少不必要的日志输出；当软件定型后，对于程序缺陷，可利用操作系统中页面替换的思想，设计一个垃圾收集自愈系统，但已是弥补手段，对软件的使用方来说存在着难度，需从根本上改变软件设计思想。

6 结束语

介绍了用.net 开发服务器程序时，出现内存泄露问题的几种可能，检测内存泄露问题的工具，针对不同内存泄露的解决方法，并给出了一个实际工作中碰到的内存泄露的例子及解决方案。另外，还描述了用.net 开发程序时提高服务器性能的方法，有助于开发更为健壮高效的服务器程序。

参考文献

1 Trey Nash 著，C#4.0 捷径教程[M]. 刘新军译. 北京：人民邮电出版社，2011

2 程振林，方金云，唐志敏，等. C++编码中减少内存缺陷的方法和工具[J]. 计算机工程，2007，33(4)：40～42

3 王喆. C/C++代码内存泄漏缺陷检测方法研究[D]. 大连：大连理工大学，2012：11～20

4 维恩霍尔特. .NET 性能优化[M]. 北京：清华大学出版社，2008

5 叶俊民，魏鹏，金聪，等. 内存泄漏故障静态分析研究[J]. 计算机科学，2010(6)：171～175

6 王翔. 设计模式－基于 C#的工程化实现及扩展[M]. 北京：电子工业出版社，2009

★★★

（上接第 43 页）

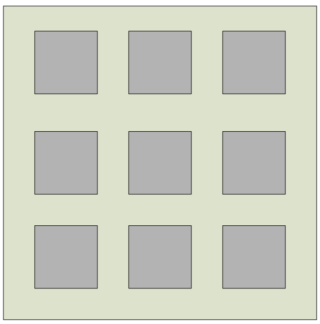


图 6 散热焊盘网板开孔位置示意图

4 结束语

在焊接此类高密度 QFN 封装芯片时，对于印制电路板焊盘不规则、位置不准确、金属化过孔打在焊盘上的情况，采用传统的网板设计方法与刮膏焊接工艺，锡珠、连焊等质量问题几乎无法避免。唯有从制板工艺开始，选择先进技术，同时在网板设计环节注重细节控制，方能装联出可靠的产品。对于已经出现的问题，工艺师应在分析、实验的基础上及时归纳总结，既能够解决问题又可以尽量避免类似问题反复发生，消除隐患。

参考文献

1 王文利，闫焉服. 电子组装工艺可靠性[M]. 北京：电子工业出版社，2011

2 秦佩，陈长生. PCB 微盲孔电镀铜填平影响因素研究[J]. 电子工艺技术，2012，33(5)：277～280

3 郑冠群. QFN 返修工艺技术[J]. 电子工艺技术，2009，30(3)：158～161