

.NET应用程序调试：原理、工具、方法 - 文章 - 伯乐在线



阅读目录：

- 1. 背景介绍
- 2. 1. Windows调试工具箱
 - 2. 2. .NET调试扩展包，SOS.DLL、SOSEX.DLL
 - 2. 3. 调试系统的基本流程及架构（.NETDAC概念、mscordacwks.dll）
 - 2. 4. VisualStudio中集成扩展调试（更加细粒度的调试程序）
- 3. 调试程序类型（客户端程序、服务端程序）
- 4. 调试方式及场景
 - 4. 1. 本机调试（Attach Process，调试器启动）
 - 4. 2. 不中断调试或者称事后调试（对Dump文件进行调试）
- 5. 一般调试步骤
 - 5. 1. 设置符号文件（公有符号、私有符号）
 - 5. 2. 加载.NET程序扩展调试包（SOS.DLL、SOSEX.DLL）
 - 5. 3. 调试的三种命令类型（标准命令、元命令、扩展命令）
- 6. 调试扩展的几个比较常用的命令（SOS.DLL、SOSEX.DLL）
- 7. 简单示例，常见的线上两类问题
 - 7. 1. 内存问题（内存偏高，内存溢出）
 - 7. 2. 线程问题（CPU过高，线程死锁）
- 8. 获取Dump文件时的重要注意事项
- 9. 总结

1. 背景介绍

随着应用程序的复杂度不断上升，要想将好的设计思想稳定的落实到线上，我们需要具备解决问题的能力。需要具备对运行时的错误进行定位且快速的解决它的能力。本篇文章我将分享一下我对.NET应用程序调试方面的学习和使用总结。

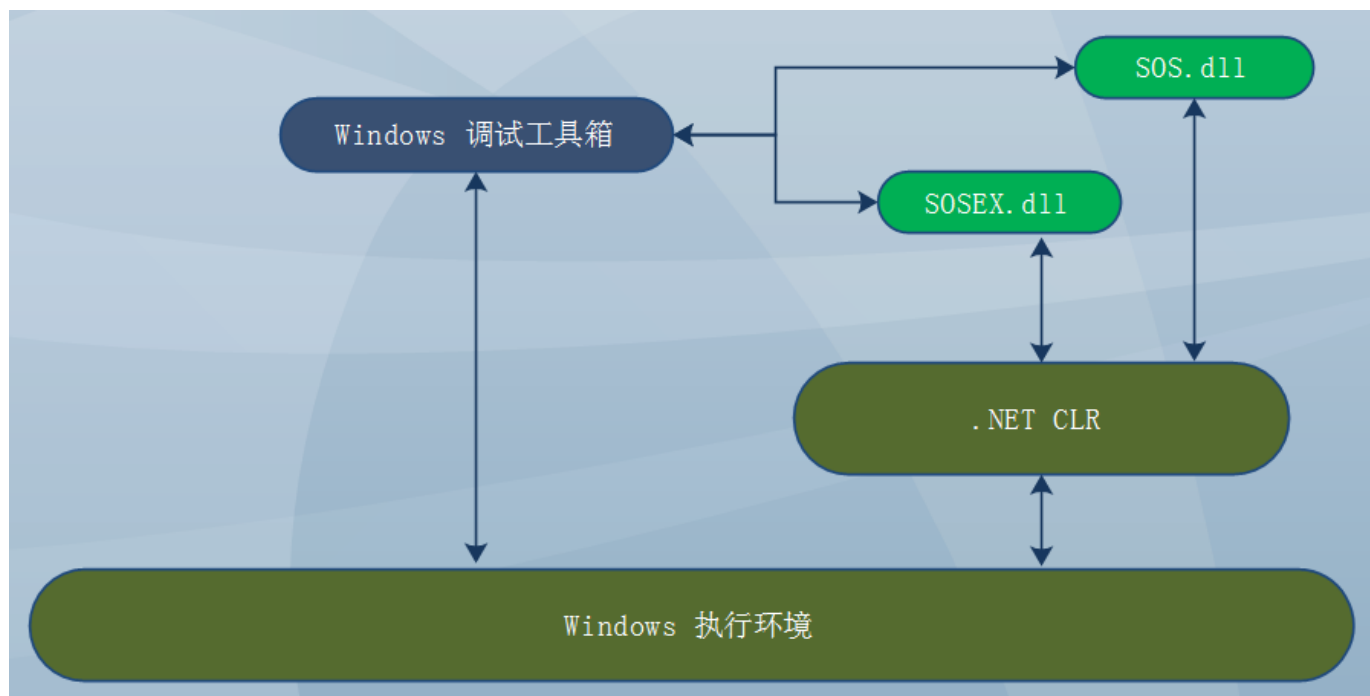
其实对调试程序的使用是不难的，关键是知道它的调试原理才行，因为调试一个程序或者dump文件，都需要了解一定的.NET调试的原理才行，比如你在附加到进程调试时在执行某个SOS扩展命令是需要切换到指定线程上的，而调试dump文件就不需要，但是对Dump文件的分析有些SOS扩展命令是不能用的，类似这样的问题，一旦出现你就一头雾水，所以花点时间学习一下原理是有必要的。

2. 基本原理（Windows调试工具箱、.NET调试扩展SOS.DLL、SOSEX.DLL）

在Windows平台上调试应用程序首选Windows调试工具箱，该工具箱包含了一套专门用来针对Windows进行很多复杂场景调试所需要的工具和组件。需要注意的是此工具箱是针对于非托管.NET平台用的，意思就是说此工具箱的所有工具和组件默认是不能够进行.NET应用程序调试的，只能用来对原生Windows程序进行调试。

那么.NET平台也并不是有自己一套专用的调试工具箱，毕竟.NET还是属于Windows平台的，所以很大部分的运行时原理还是基于Windows的，要想在原生的调试器中对.NET这个具有虚拟运行时程序进行调试就需要专门的翻译器才能够执行。SOS.DLL、SOSEX.DLL这两个就是用来对.NET程序在Windows调试工具中起到翻译作用的调试器扩展。简单讲就是，这两个组件是.NET项目组专门开发出来用来对.NET应用程序进行方便调试用的，当然不用这两个扩展也能调试.NET程序，只不过就会很困难，会被很多细节束缚住。有了这个调试扩展之后，我们就可以让原生Windows调试器正确的翻译出.NET相关概念。

图1：（Windows调试工具执行流程）



所有对.NET程序发起的调试会话都要经过.NET调试扩展组件进行翻译才行，也就是要使用.NET调试扩展的调试命令来调试.NET程序。上图中，我们如果要想调试.NET程序就需要将.NET调试扩展组件加载到Windows调试工具中去，然后才能方便在Windows调试工具中使用。

2.1. Windows调试工具箱

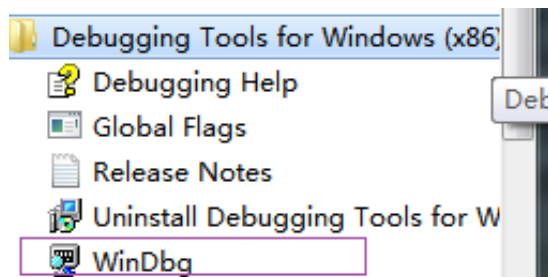
Windows调试工具箱中包含了很多调试工具，都是用来辅助于我们进行方便调试用的。Windows调试工具箱分为两个执行版本，X86、X64这两个版本是专门用来分析不同的运行时环境的，如果你的分析环境是32位的你就需要使用X86的版本，同理，如果是用64位的环境就需要使用X64的版本。

下载地址为：<http://www.microsoft.com/whdc/devtools/debugging/default.aspx>

记住选择你需要的版本，建议你两个版本都下载，因为你随时需要针对Dump文件进行分析，而Dump文件是随时都有可能是两个版本。

Windows工具箱中的默认使用WinDbg.exe作为调试首选，它是一个GUI程序。

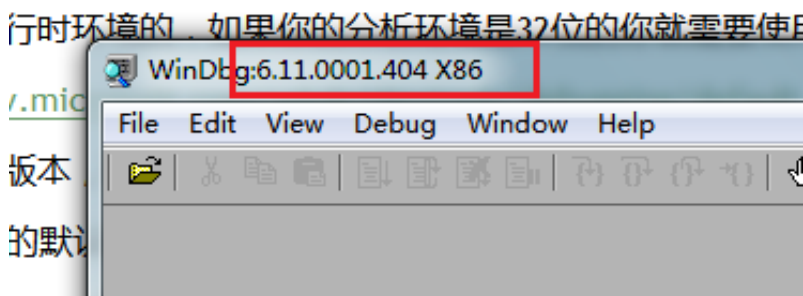
图2：（默认的Windows调试工具，WinDbg）



安装过后的菜单中就只有WinDbg作为调试选择。

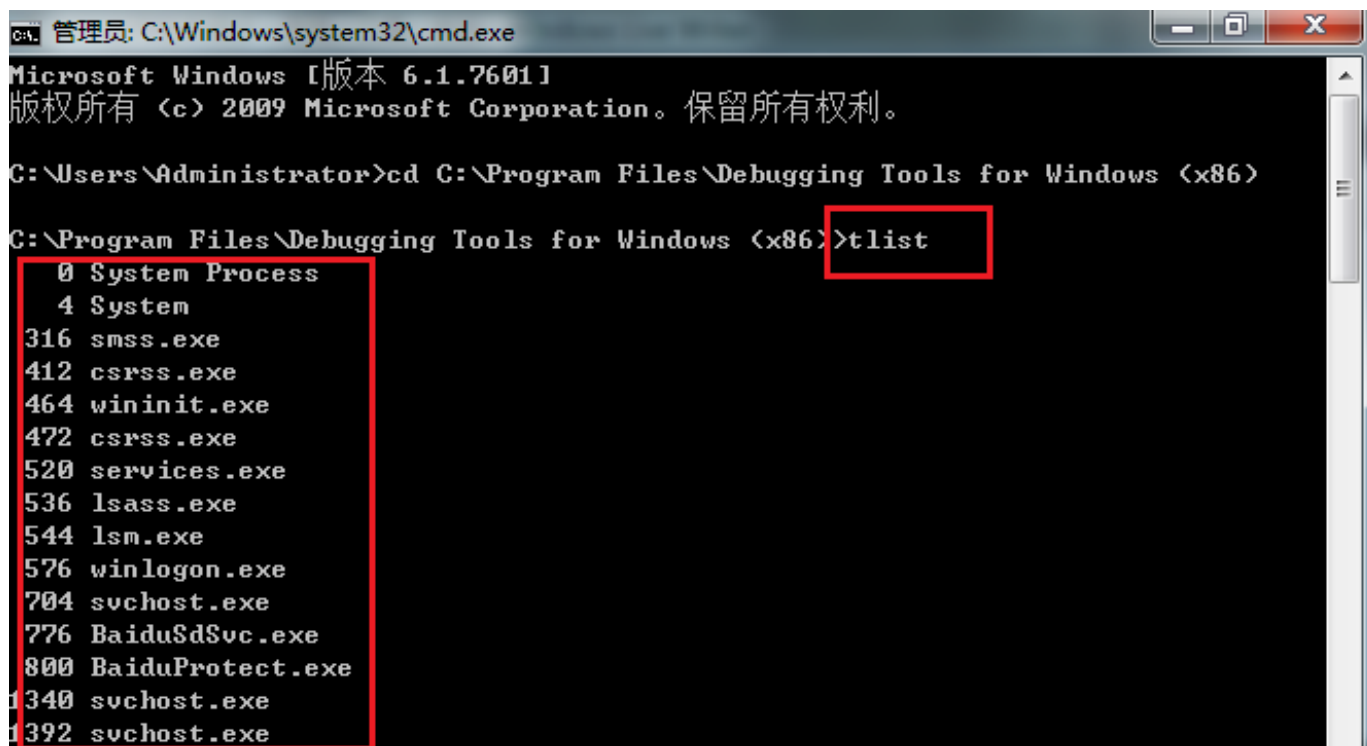
这里需要注意的是，当你启动了WinDbg之后要留意程序的名字和标题，因为当你存在两个版本的WinDbg时会容易搞错，在调试时会有各种奇怪的问题出现，当你找了半天之后结果发现是因为用错了版本，那就正的无语了。

图3：（注意运行WinDbg的环境版本）



WinDbg是默认的调试工具，但是在工具箱中还有几个控制台调试工具，他们行必之下比较轻量简单，有些任务比较好执行，在配合cmd使用会很方便，比如工具箱中的tlist.exe用来查看进程信息的小工具就非常方便。

图4：（方便查看进程ID）



```
管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd C:\Program Files\Debugging Tools for Windows (x86)

C:\Program Files\Debugging Tools for Windows (x86)>tlist
 0 System Process
 4 System
316 smss.exe
412 csrss.exe
464 wininit.exe
472 csrss.exe
520 services.exe
536 lsass.exe
544 lsm.exe
576 winlogon.exe
704 svchost.exe
776 BaiduSdSvc.exe
800 BaiduProtect.exe
1340 svchost.exe
1392 svchost.exe
```

这样我们就可以很方便的attach到一个指定的进程进行调试。

Windows调试工具箱中有很多其他的工具，需要用的话可以使用cmd切换到当前安装的目录下:C:\Program Files\Debugging Tools for Windows (x86)，或者你直接到工具的安装目录运行也行，这就看此工具是不是支持手动无参数启动了。

2.2..NET调试扩展包，SOS.DLL、SOSEX.DLL

.NET调试扩展包分为两个，一个是SOS.DLL，该扩展包是.NET平台的一部分，属于官方版本。而SOSEX.DLL是微软的一名叫“Steve Johnson”软件工程师开发，属于个人维护的，用来增强SOS.DLL功能的，在SOSEX.DLL有很多功能比较强大的扩展命令。

下载地址为：

32位：http://www.stevetechspot.com/downloads/sosex_32.zip

64位：http://www.stevetechspot.com/downloads/sosex_64.zip

具体的帮助文档可以查看该工程师的博客了解详情。这两个版本用来调试不同环境的程序的，如果你的程序是运行在32位环境下，就用32位的SOSEX，同理，用在64位下就用64位SOSEX。

而SOS.DLL扩展包是跟着.NETFramework一起安装的，地址位于：

C:\Windows\Microsoft.NET\Framework\v4.0.30319。如果你是64位系统的话地址就是：

C:\Windows\Microsoft.NET\Framework64\v4.0.30319。在这两个地址下面都可以找到SOS.dll文件，不同的目录下对应于调试不同机器类型的.NET程序。

有了这两个扩展包之后就可以在WinDbg中对.NET程序进行分析了，具体使用我们后面会介绍。

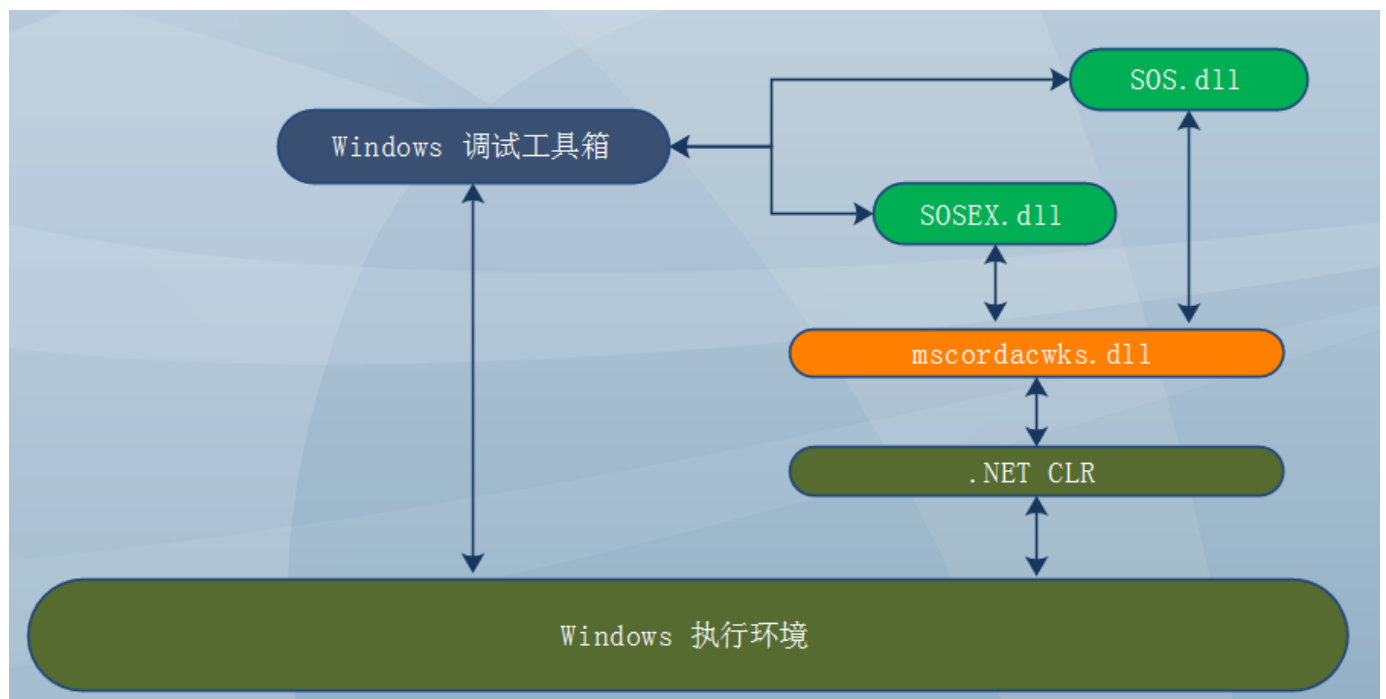
2.3. 调试系统的基本流程及架构（.NETDAC概念、mscordacwks.dll）

有一个很重要的原理我觉得很有必要讲一下，就是.NETDAC概念。

其实.NETDAC也就是.NET Data Access .NET数据访问层，这个是专门用来提供给SOS.DLLS0SEXDLL或者其他调试扩展包使用的，所有的调试扩展组件必须通过这个DAC才能访问到.NET运行时的数据，所以在初次使用SOS的时候会经常碰见加载错误的mscordacwks.dll文件，此文件就是DAC的物理文件。

这个文件和SOS扩展文件一样，都有这不同的版本，当加载不同类型的.NET程序时会使用到不同版本的mscordacwks.dll文件，当然大部分情况下此文件时自动加载的，只有出现你分析的文件与生成调试文件的环境不一致时才会出现头疼的问题。

图5：（mscordacwks.dll位置）



当你知道这个组件是工作于此位置时，当出现跟它相关的错误提示时你就不需要担心了，无非就是文件加载的位置或者版本不匹配而已。

调试器会话、调试器注入线程

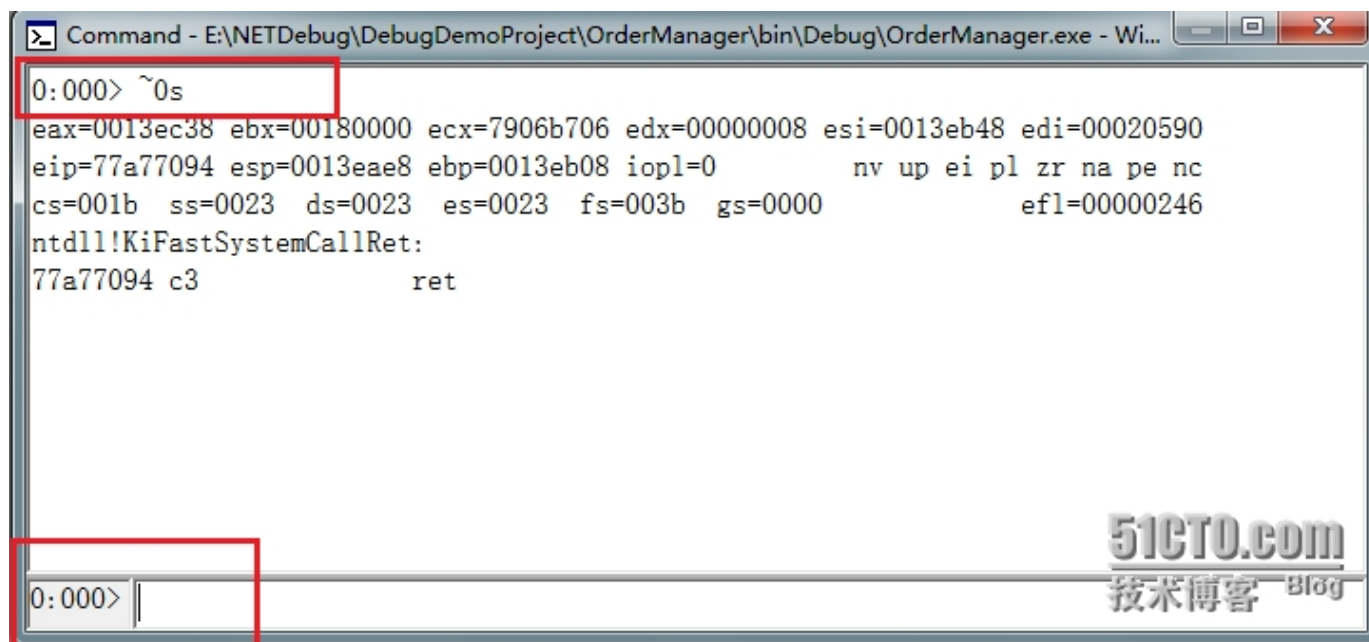
还有一点我觉得也很有必要介绍的就是有关调试器如何调试.NET程序的，当我们在使用调试器启动被调试程序或者将调试器附加到被调试进程时，其实调试器会注入一些线程到.NET程序中，让调试线程与.NET程序原本的线程在一个.NET执行环境中，这样的目的是能够起到最.NET程序在执行时的控制，比如中断执行，设置断点。当我们需要执行某些跟线程上下文相关的扩展命令时就需要切换到正确的线程上去。

图6：（调试器注入线程）

```
ModLoad: 75b40000 75b4c000 C:\Windows\system32\CRYPTBASE.dll
ModLoad: 78d80000 78ddb000 C:\Windows\Microsoft.NET\Framework
(cb7ec.cb520): Control-C exception - code 40010005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00e3fa58 ebx=00000000 ecx=00000000 edx=77a77094 esi=00000000
eip=7750e37d esp=00e3fa48 ebp=00e3fac8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
KERNEL32!CtrlRoutine+0xcbb:
7750e37d c745fcfeffff mov     dword ptr [ebp-4],0FFFFFFFFh
0:003>
```

此时，调试器使用一个注入线程将.NET程序在执行时中断，原理就是通过发送线程中断命令来达到控制目标线程，那么首先要能够与原线程通讯才行，所以需要注入托管线程。（注意：注入的线程不一定是托管.NET线程，严重它最好的方法就是查看所有所有的进程内线程和所有托管线程，对比一下就知道了。），其实这个ID为3的线程是调试器会话线程。

图7：（切换到原托管线程）



我们通过~0s命令切换到我们需要调试的原托管线程中，比如，在执行!ClrStack命令时，就需要切换到当前线程上执行。

我们需要验证它是否是注入了托管线程还是非托管线程。

图8：（托管线程列表）


```

0:007> !threads
ThreadCount:      2
UnstartedThread:  0
BackgroundThread: 1
PendingThread:    0
DeadThread:       0
Hosted Runtime:   no

           Lock
ID OSID ThreadOBJ  State GC Mode  GC Alloc Context Domain Count Apt Exception
0   1 ccf40 003edea8  2a020 Preemptive 01BA7DDC:00000000 003b4f08 1    MTA
2   2 c93a4 003bd7e0  2b220 Preemptive 00000000:00000000 003b4f08 0    MTA (Finalizer)
  
```

使用!Threads命令可以查看进程内所有的托管线程，仅仅是托管线程，此命令是无法查看非托管线程的，接下来我们使用另外一个命令来查看所有的线程。

图9：（所有的执行时线程）

```

0:007> !runaway
User Mode Time
Thread      Time
0:ccf40     0 days 0:00:00.031
7:cc9cc     0 days 0:00:00.000
6:cc8d8     0 days 0:00:00.000
5:ccb34     0 days 0:00:00.000
4:cc9e4     0 days 0:00:00.000
3:cc0b8     0 days 0:00:00.000
2:c93a4     0 days 0:00:00.000
1:cb5c8     0 days 0:00:00.000
  
```

目前调试器会话使用的非托管线程

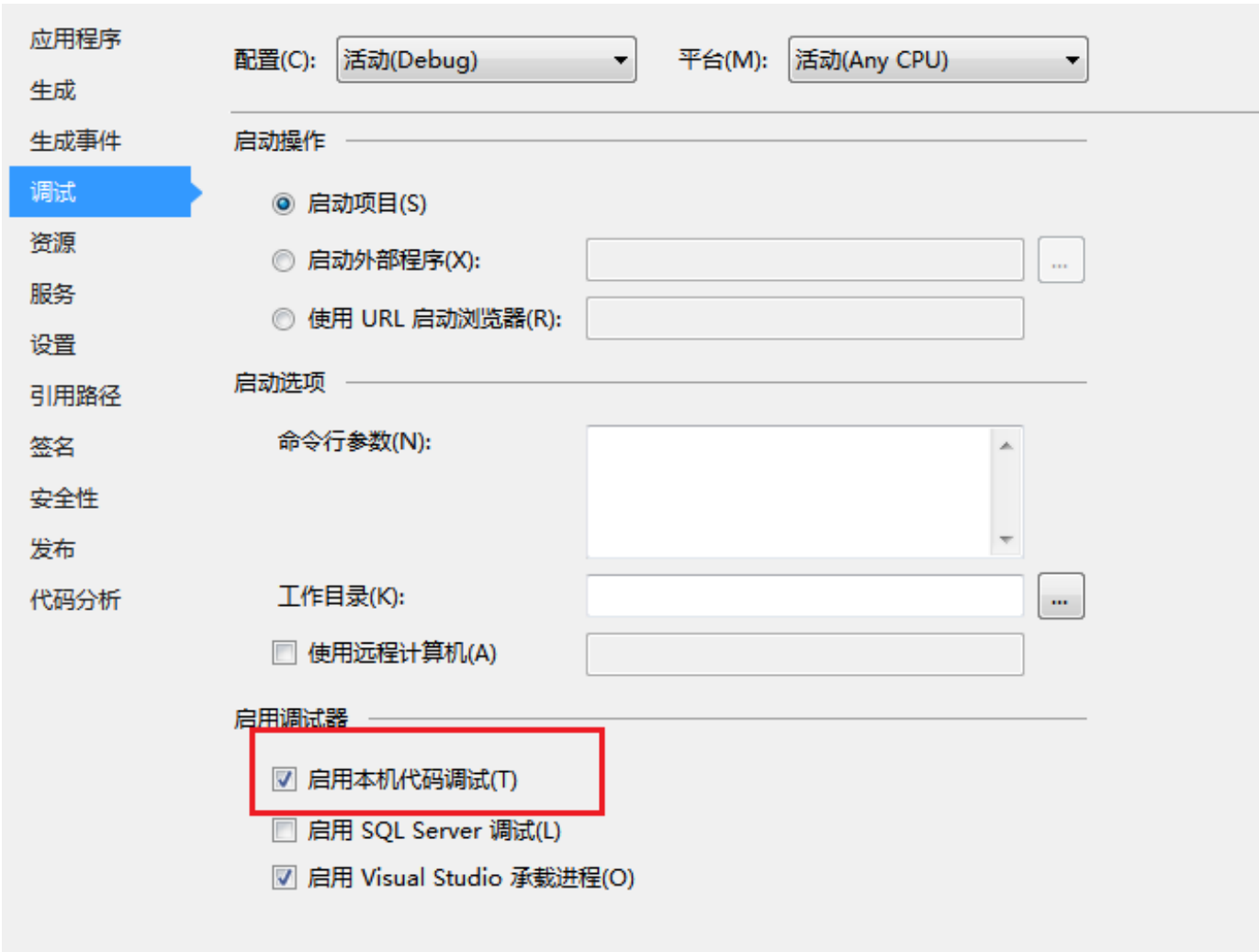
两个托管线程

这样我们就可以判断出，调试器使用了ID位7的作为目前的调试会话线程。知道这些背后的原理很重要，当你在执行某个调试命令时你就会发现此命令是否需要在.NET线程中执行，还是说可以在调试器会话线程中执行，一般dump类的命令都是可以远程执行的，也就是说在调试器会话中执行，当需要跟踪.NET线程内部过程时就需要切换到.NET线程上去执行。

2.4. VisualStudio中集成扩展调试（更加细粒度的调试程序）

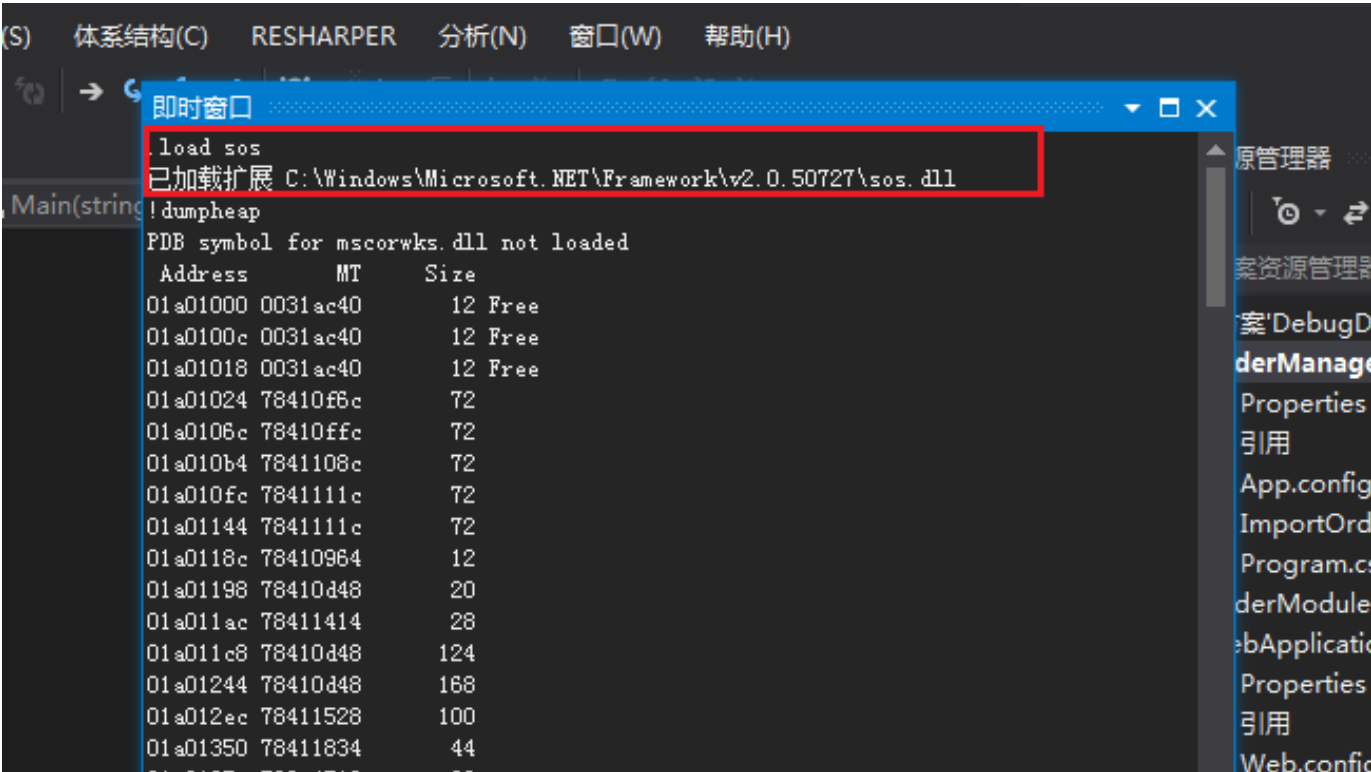
SOS扩展也是可以和VisualStudio进行集成的，这样真的方便了我们调试一些性能要求比较高的程序，当程序运行一段时间后我们用VS附加到进程，然后查看一些重要的对象数据，但是此时我们看不到.NET运行时的一些数据，比如：对象的代龄，托管堆的大小，线程池的任务等。通过集成SOS扩展会让我们对程序的运行时有了一个更加方便的跟踪。

图10：（打开本地代码调试）



设置断点，然后在”即时窗口 “（调试->窗口->即时）中加载扩展SOS.DLL。

图11：（在VisualStudio2012中加载SOS.dll扩展）



这样的便利性大大提高我们在调试程序内存方面、线程方面的好处，我们可以适当的做压力测试，然后 Attach process，执行SOS扩展命名来查看内存问题，当需要调试程序逻辑时在单步调试C#代码，一举两

得。

3. 调试程序类型（客户端程序、服务端程序）

.NET程序主要分为两类，一类是客户端程序，另一类是服务端程序。对于这两类程序来说前者调试时基本上可以通过附加进程的方式进行调试，而对于服务端程序则不行，因为服务程序通常是运行在一个复杂的线上环境中，我们没有任何权限或机会去接触，此时是通过获取进程的dump文件来进行分析。

客户端程序也大概分为控制台、Winform两种，服务端程序都是基于ASP.NET框架，宿主与IIS进程中。

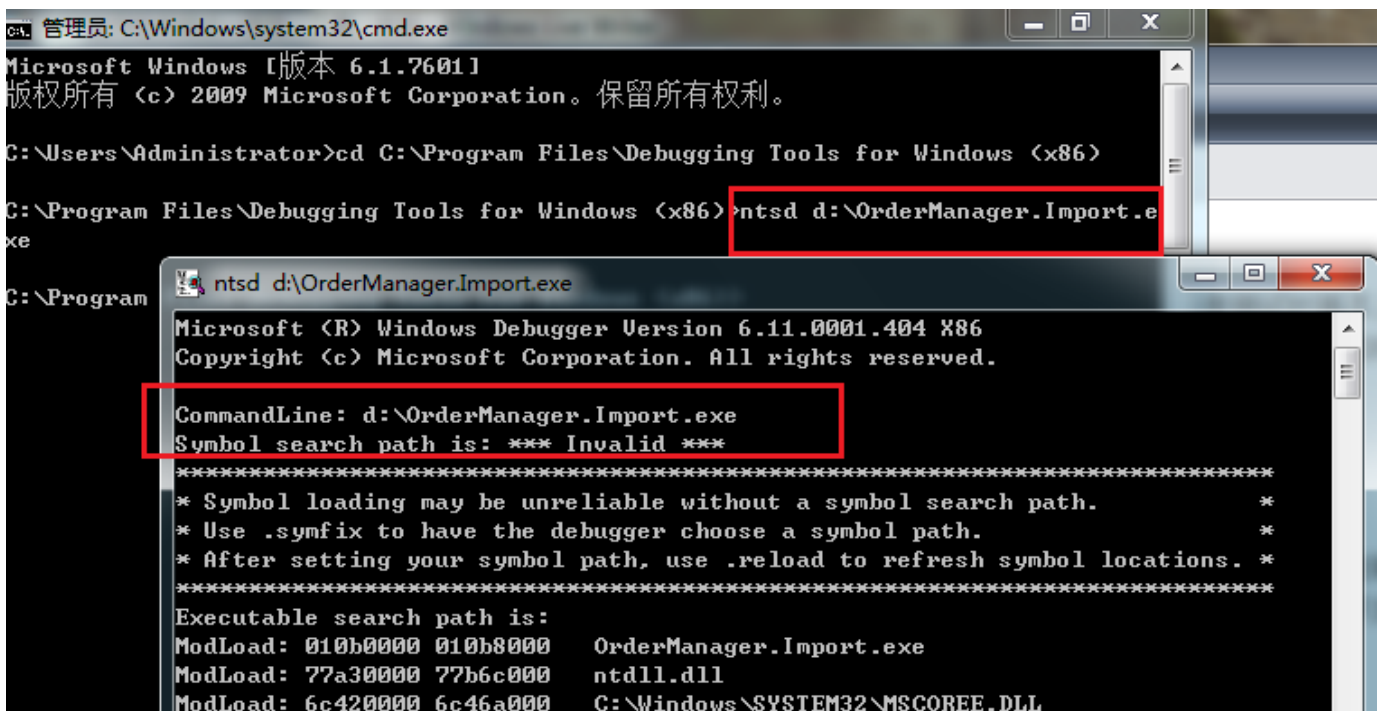
4. 调试方式及场景

针对不同类型的程序及场景需要使用不同的方式进行调试，客户端程序中的控制台程序基本上可以通过在调试器中启动的方式进行调试。如果是GUI程序则需要附加进程方式。服务端程序如果在条件允许下也是可以使用附加进程的方式进行调试的，但是这一般不太可能，因为一旦附加进程将block住所有的线程活动。

4.1. 本机调试（Attach Process，调试器启动）

本机调试可以直接在调试器中启动程序，WinDbg打开后，在文件中有一个Open Executable，可以打开一个可执行文件。如果是使用NTSD控制台调试器，则需要在NTSD后面跟上程序的执行路径。

图12：（ntsd.exe打开调试程序）



同样，在WinDbg中也有一个附加进程的选项，NTSD也是一样，操作起来都比较简单，需要注意的是当你对进程进行附加时要清楚此进程是多少位的，然后你需要选择正确的调试器进行调试。

4.2. 不中断调试或者称事后调试（对Dump文件进行调试）

在不能够对被调试程序直接调试时我们就需要此程序的进程镜像文件，此镜像文件就是进程在某一个时刻的快照，通过分析这个快照，我们也是可以定位出问题的。首先我们需要使用适当的工具来获取进程的dump文件，操作系统本身的任务管理器就有这个功能，dump文件的存放位置默认在用户信息临时文件下面，比如：XXXUsersAdministratorAppDataLocalTemp，获取完dump文件后任务管理器会有提示路径的。

图13：（使用任务管理器获取dump文件）

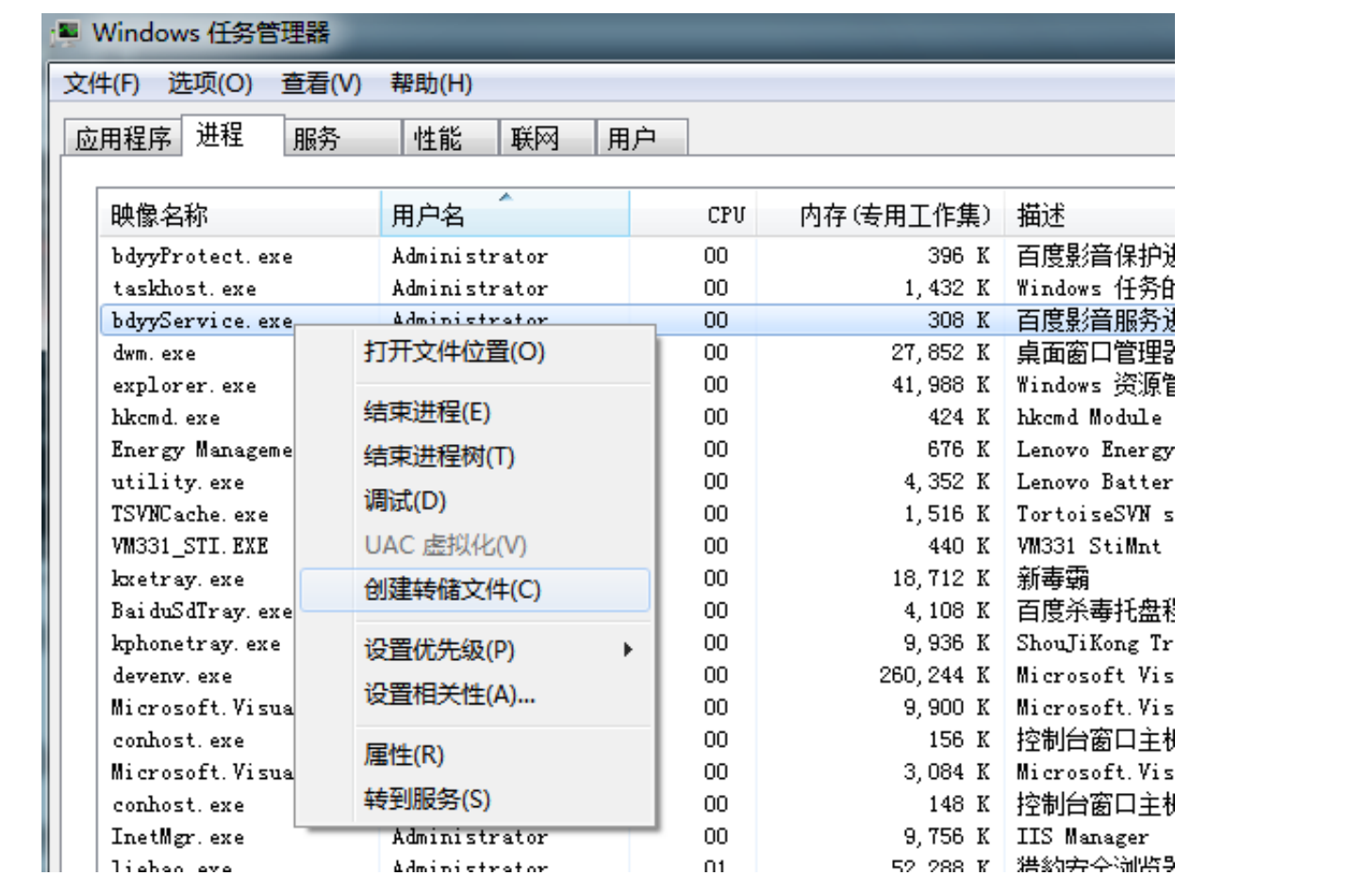
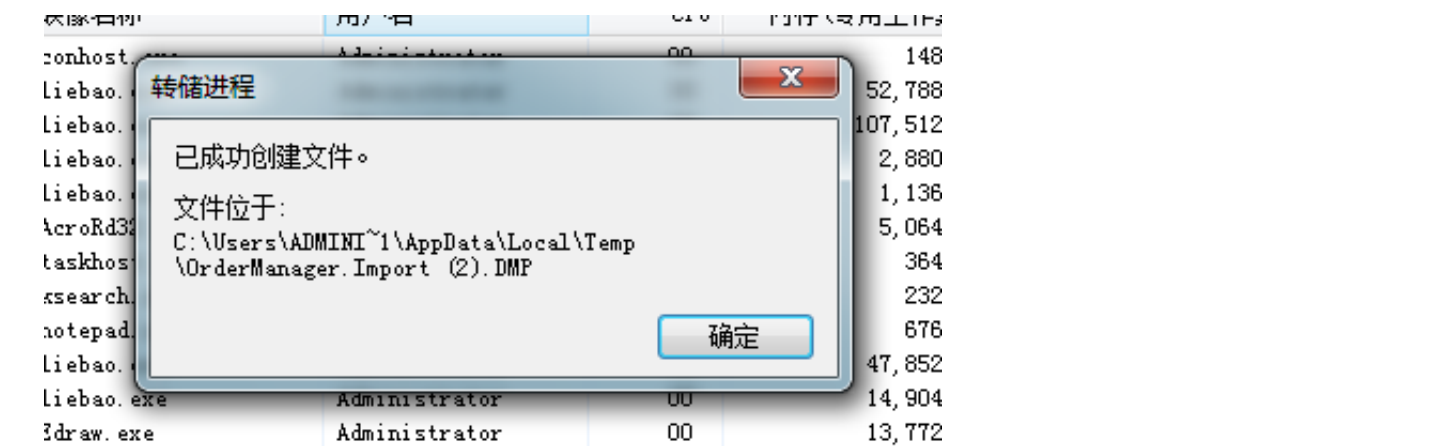


图14：

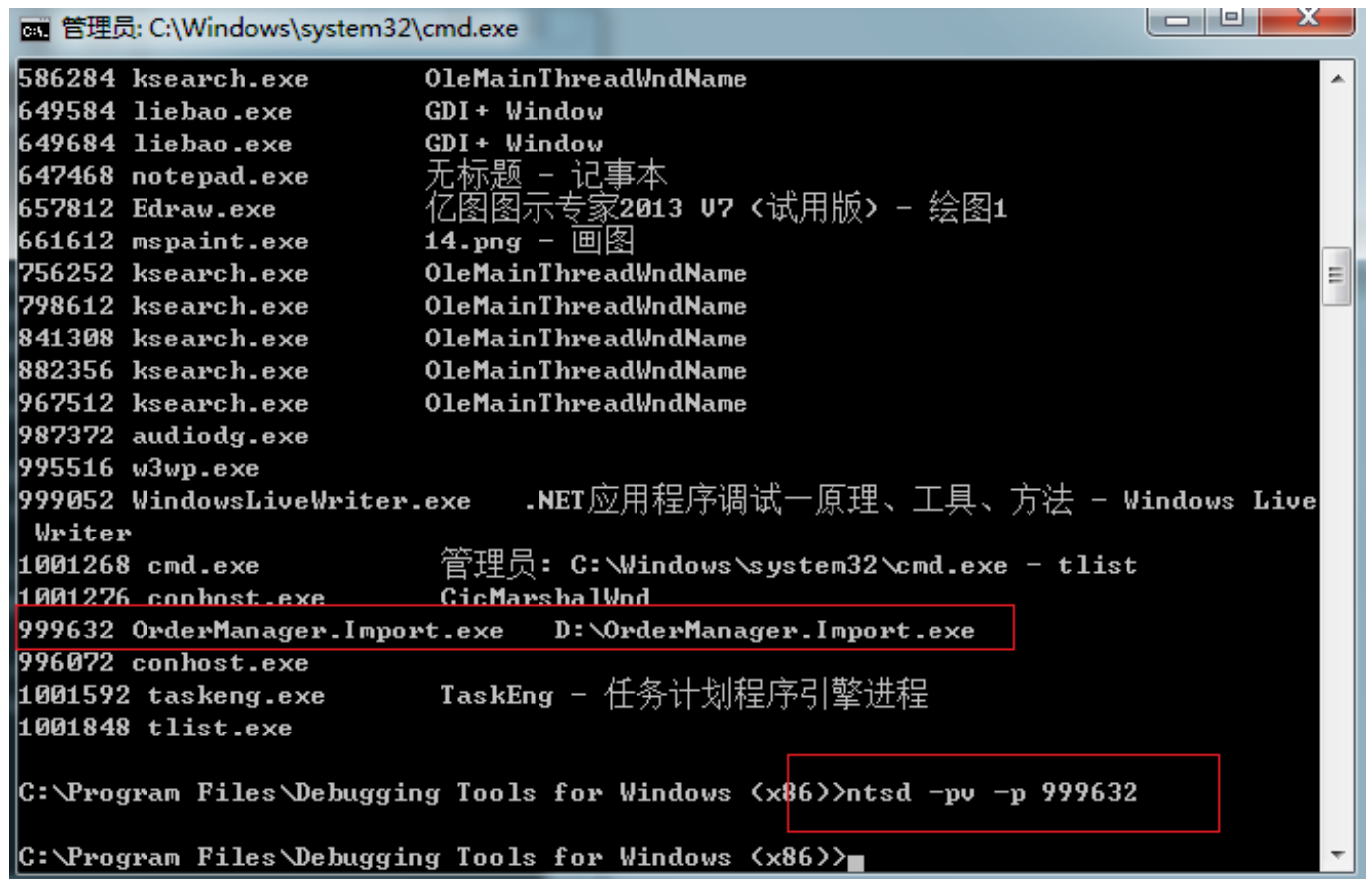


使用任务管理器获取dump文件固然很方便，但是有一个问题就是如果当前机器是64位的，并且你的进程是以32位方式运行的，那么此时你获取出来的dump文件是64位的，当你通过32位的调试器无法进行分析，甚至会有各种其他的问题，这些问题就是因为获取dump文件的机器环境和你预想的不一致。这个时候我们希望能够通过很明了的方式来获取dump文件，就是通过调试器来获取dump文件。

通过调试器来获取dump文件有很多好处，可以设置很多选项，包括只获取进程的哪部分镜像数据等。

先通过tlist.exe查看所有进程列表，会有一个进程ID号，有了ID号才能进行获取。

图15：（tlist、ntsd 进入到指定进程中）



```
管理员: C:\Windows\system32\cmd.exe
586284 ksearch.exe      OleMainThreadWndName
649584 liebao.exe       GDI+ Window
649684 liebao.exe       GDI+ Window
647468 notepad.exe      无标题 - 记事本
657812 Edraw.exe         亿图图示专家2013 U7 <试用版> - 绘图1
661612 mspaint.exe         14.png - 画图
756252 ksearch.exe      OleMainThreadWndName
798612 ksearch.exe      OleMainThreadWndName
841308 ksearch.exe      OleMainThreadWndName
882356 ksearch.exe      OleMainThreadWndName
967512 ksearch.exe      OleMainThreadWndName
987372 audiodg.exe
995516 w3wp.exe
999052 WindowsLiveWriter.exe .NET应用程序调试一原理、工具、方法 - Windows Live
Writer
1001268 cmd.exe         管理员: C:\Windows\system32\cmd.exe - tlist
1001276 conhost.exe     CicMarshalWnd
999632 OrderManager.Import.exe D:\OrderManager.Import.exe
996072 conhost.exe
1001592 taskeng.exe       TaskEng - 任务计划程序引擎进程
1001848 tlist.exe

C:\Program Files\Debugging Tools for Windows (x86)>ntsd -pv -p 999632
C:\Program Files\Debugging Tools for Windows (x86)>
```

进入到ntsd调试器中，然后使用.dump/mf d:\order.dmp 命令获取dump文件到D盘。

图16：（使用NTSD.exe获取dump文件）



```
ntsd -pv -p 999632
0:000> .dump/mf d:\order.dmp
Creating d:\order.dmp - mini user dump
Dump successfully written
0:000>
```

此时我们就成功的获取到了dump文件。

通过调试器获取dump文件比较稳定可靠，因为机器运行环境的不同，通过任务管理器获取的dump文件会存在一些无法预知的问题，你并不清楚，当前任务管理器是使用哪个版本的环境输出调试信息的。

有了dump文件之后就是通过调试工具打开就行了，WinDbg就有一个菜单专门打开dump文件的，Open

Crash Dump。使用ntsd需要使用命令ntsd -z d:order.dmp。

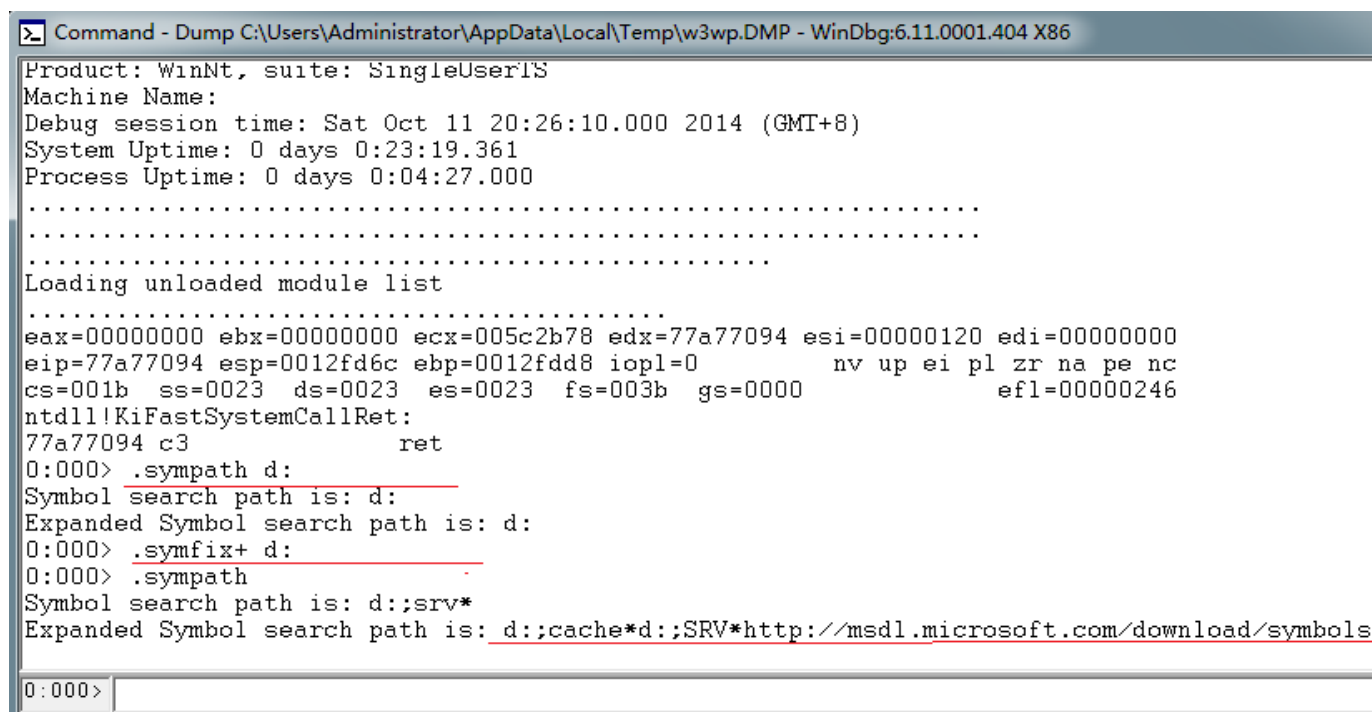
5. 一般调试步骤

知道了调试的一些原理和工具之后我们来看一下调试的基本步骤，这些步骤都具体是指的什么意思，有哪些好处。

5.1. 设置符号文件（公有符号、私有符号）

设置符号文件的目的是为了能够在调试器中正确的对应到源代码的位置和一些元数据信息。符号文件都是*.pdb文件名。符号文件分为公有和私有两种，公有的都是公司公开出去用于帮助调试用的，而私有的是公司内部使用的，为什么要区分公有和私有，是为了防止逆向工程。

图17：（设置符号文件路径）



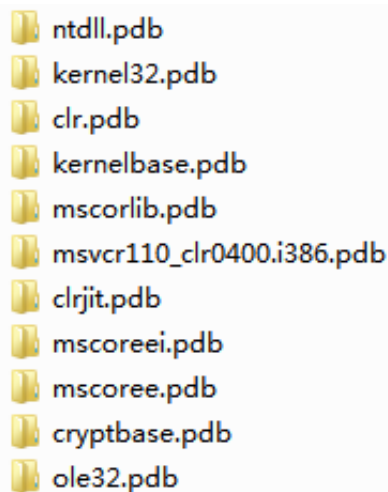
```
Command - Dump C:\Users\Administrator\AppData\Local\Temp\w3wp.DMP - WinDbg:6.11.0001.404 X86
Product: WinNt, suite: SingleUserIS
Machine Name:
Debug session time: Sat Oct 11 20:26:10.000 2014 (GMT+8)
System Uptime: 0 days 0:23:19.361
Process Uptime: 0 days 0:04:27.000
.....
Loading unloaded module list
.....
eax=00000000 ebx=00000000 ecx=005c2b78 edx=77a77094 esi=00000120 edi=00000000
eip=77a77094 esp=0012fd6c ebp=0012fdd8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!KiFastSystemCallRet:
77a77094 c3          ret
0:000> .sympath d:
Symbol search path is: d:
Expanded Symbol search path is: d:
0:000> .symfix+ d:
0:000> .sympath
Symbol search path is: d;;srv*
Expanded Symbol search path is: d;;cache*d;;SRV*http://msdl.microsoft.com/download/symbols
0:000>
```

首先通过.sympath d:，设置了符号路径为D盘，然后又使用.symfix+ d:，是设置私有符号路径，并且使用d盘为缓存路径。在最后一个红线中我们能看出来。

为什么使用.symfix 时要带上一个+号，其实是告诉调试器我们是多加一个符号位置，而不是覆盖原有符号位置。

设置好了两个符号位置后需要使用.reload命令来重新加载模块，这样调试器才会去符号位置去加载这些符号。

图18：（加载的符号文件）



调试器会自动的将公有符号下载到刚才设置的缓存目录中。

5.2. 加载.NET程序扩展调试包（SOS.DLL、SOSEX.DLL）

对.NET程序分析当然是需要加载SOS扩展了。加载SOS扩展有两个命令可以使用，第一个是`.load C:\Windows\Microsoft.NET\Framework\v4.0.30319\SOS.dll`，`.load`命令是要给出`sos.dll`绝对路径的。第二个是`.loadby sos modulename`，`.loadby`命令是可以根据已经加载的模块名称来加载SOS.dll扩展。使用第一个命令有一个问题就是，我们需要人工的判断当前环境到底是需要什么版本的SOS扩展，而使用`.loadby`是可以根据已经加载的模块来自动的查找对应的SOS扩展。

```
0:000> .load C:\Windows\Microsoft.NET\Framework\v4.0.30319\SOS.dll
```

```
0:000> .loadby sos.dll clrjit
```

使用`.loadby`命令很容易的就可以加载SOS扩展，而不需要自己去判断当前程序是.NET什么版本的。

5.3. 调试的三种命令类型（标准命令、元命令、扩展命令）

在使用调试器调试程序时，所要使用的命令主要分为三类。

第一类是标准命令，就是不带任何符号开始的命令，比如：`pb`、`lmvm`。这一类命令是所有Windows调试工具箱中的调试工具通用的，不管你是使用`ntsd`还是`winDbg`都可以。

第二类命令是元命令，就是使用“`.`”号开始的命令，这一类命令并不是在所有调试工具中通用的。第三类是扩展命令，扩展命令就是各个调试器扩展出来的命令，也就是以“`!`”开始的命令，如：`!dumpheap -stat`，`!dumpstatcobjects`。

6. 调试扩展的几个比较常用的命令（SOS.DLL、SOSEX.DLL）

当然这个纯粹是我的个人感觉，排名不分先后。

`!dumpheap -stat` （查看托管堆统计信息）

```
0:000> !dumpheap -stat
```

Statistics:			
MT	Count	TotalSize	Class Name
65366e78	1	12	
System.Collections.Generic.EnumEqualityComparer<System.Web.Compilation.FolderLevelBuildProviderAppliciesto, System.Web>			
653667cc	1	12	
System.Collections.Generic.ObjectEqualityComparer<System.Web.WebSockets.IAsyncAbortableWebsocket, System.Web>			
65365f08	1	12	
System.Lazy<Boxed<System.Web.Security.Cryptography.AspNetCryptoServiceProvider, System.Web>>			
65365a34	1	12	
System.Web.Security.Cryptography.HomogenizingCryptoServiceWrapper			
65361e20			
1	12	System.Web.Configuration.CustomErrorsMode	

```
!dumpheap -type      （查看某个类型在堆中的信息）
0:000> !dumpheap -type System.String

Address             MT             Size
10731228 624aacc0      14
107312c4 624aacc0  22
107312dc 624aacc0      78
10731370
624aacc0             28
```

可以一眼看出哪些对象过大，这里我是为了演示而用，一般在项目开发中，我们都大概知道哪些对象可能会有内存问题，比如：同步数据时的缓存对象。

```
!dumpobj 10731228 （查看对象详情）
0:000> !dumpobj 10731228
Name:
System.String
MethodTable: 624aacc0
EEClass:      620b486c
Size:
14(0xe) bytes
File:
C:\Windows\Microsoft.Net\assembly\GAC_32mscorlibv4.0.4.0.0__b77a5c561934e089mscorlib.dll
String:

Fields:
```



```
MT      Field      Offset      Type VT      Attr
Value Name
624ac480  40000aa      4      System.Int32  1 instance
0 m_stringLength
624ab6b8  40000ab      8      System.Char   1 instance
0 m_firstChar
624aacc0  40000ac      c
System.String  0      shared      static Empty
>> Domain:Value
00dbe558:NotInit  00e11c90:NotInit  00e5f040:NotInit

!threads (查看托管线程)
0:000> !threads
ThreadCount:      17
UnstartedThread:  0
BackgroundThread: 12
PendingThread:    0
DeadThread:       5
Hosted Runtime: no

Lock
ID OSID ThreadOBJ      State GC Mode      GC Alloc Context  Domain      Count Apt
Exception
7      1 43a8 00dc2620      28220 Preemptive  1484CA40:00000000 00dbe558 0      Ukn
15     2 4414 00dd38d0      2b220 Preemptive  00000000:00000000 00dbe558 0
MTA (Finalizer)
17     3 441c 00e09e88      102a220 Preemptive  00000000:00000000 00dbe558 0      MTA
(Threadpool
Worker)
18     4 4420 00e0ce80      21220 Preemptive  00000000:00000000 00dbe558 0
Ukn
```

当然还有很多其他很不错的命令，这里我个人觉得这几个比较常用，要想了解所有的命令可是在调试器中使用扩展命令!help来查看所有的命令帮助。

```
0:000> !help

-----

SOS is a debugger extension DLL designed to aid in the debugging of managed programs.
Functions are listed by category, then roughly in order of importance. Shortcut names for
popular functions are listed in parenthesis. Type "!help " for detailed info on that
function.

Object Inspection      Examining code and stacks
```

DumpObj (do)	Threads
DumpArray (da)	ThreadState
DumpStackObjects (dso)	IP2MD
DumpHeap	U
DumpVC	DumpStack
GCRoot	EESStack
ObjSize	CLRStack
FinalizeQueue	GCInfo
PrintException (pe)	EHInfo
TraverseHeap	BPMD
COMState	
Examining CLR data structures	Diagnostic Utilities
DumpDomain	VerifyHeap
EEHeap	VerifyObj
Name2EE	FindRoots
SyncBlk	HeapStat
DumpMT	GCWhere
DumpClass	ListNearObj (lno)
DumpMD	GCHandles
Token2EE	GCHandleLeaks
EEVersion	FinalizeQueue (fq)
DumpModule	FindAppDomain
ThreadPool	SaveModule
DumpAssembly	ProcInfo
DumpSigElem	StopOnException (soe)
DumpRuntimeTypes	DumpLog
DumpSig	VMMap
RCWCleanupList	VMStat
DumpIL	MinidumpMode
DumpRCW	AnalyzeOOM (ao) DumpCCW

Examining the GC history

Other

HistInit

FAQ HistRoot HistObj

HistObjFind HistClear

7. 简单示例，常见的线上两类问题

这里我们使用两个小示例直观的感受一下接触.NET运行时状态的感受，尽管真实的问题可能比这个复杂很多，但是解决问题的思路是一样的。

7.1. 内存问题（内存偏高，内存溢出）

服务程序最怕的性能问题之一就是内存，当内存很高的情况下我们能够通过对dump文件进行查看，看哪些对象导致内存一直高。当内存一直高的情况下就会容易导致内存溢出异常，甚至是GC频繁的执行，当GC一执行就会导致服务并发下降，因为它要挂起所有的线程（这里指的是服务器模式的.NETCLR，相对应的还有工作站模式的.NETCLR）。

```
1
2
3
4
5
6
7
8
9
10
11
12
namespace OrderManager
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("app begin...");
            Console.ReadLine();
            List l = new List();
            for (int i = 0; i
```

这一段代码会一直分配内存直到最后内存溢出异常终止程序，我们在内存比较的情况下来获取一个dump文件，然后通过适当的命令来定位哪个对象占用内存过高。

在不知道对象类型的情况下比较简单的方式就是使用：0:000> !dumpheap -stat，命令，该命令的意思是统计当前堆的信息，在这里就可以一眼找到哪个对象占用多少内存。

```
0:000> !dumpheap -stat
```

```
Statistics:
```

MT	Count	TotalSize	Class
----	-------	-----------	-------

Name

624ad6a8	1	12	
----------	---	----	--

System.Collections.Generic.GenericEqualityComparer<System.String, mscorlib>			
---	--	--	--

624ac480	1	12	System.Int32
----------	---	----	--------------

624aa58c	1	12	
----------	---	----	--

System.Collections.Generic.ObjectEqualityComparer<System.Type, mscorlib>			
--	--	--	--

624adec0	1	16	
----------	---	----	--

System.Security.Policy.AssemblyEvidenceFactory			
--	--	--	--

624ace34	1	16	System.Text.DecoderReplacementFallback
----------	---	----	--

624acde4	1	16	System.Text.EncoderReplacementFallback
----------	---	----	--

6247a840	1	16	System.IO.TextReader+SyncTextReader
----------	---	----	-------------------------------------

624ade0c	1	20	
----------	---	----	--

Microsoft.Win32.SafeHandles.SafePEFileHandle			
--	--	--	--

6245fe58	1	20	
----------	---	----	--

Microsoft.Win32.SafeHandles.SafeFileMappingHandle			
---	--	--	--

6245fe08	1	20	
----------	---	----	--

Microsoft.Win32.SafeHandles.SafeViewOfFileHandle			
--	--	--	--

6245fd74	1	20	System.Text.InternalEncoderBestFitFallback
----------	---	----	--

6245f714	1	20	System.IO.Stream+NullStream
----------	---	----	-----------------------------

624ad3d4	1	24	System.Version
----------	---	----	----------------

6245fdc4	1	24	System.Text.InternalDecoderBestFitFallback
----------	---	----	--

6245fa8c	1	24	System.IO.TextWriter+SyncTextWriter
----------	---	----	-------------------------------------

00163170	1	24	
----------	---	----	--

System.Collections.Generic.List<System.Byte[], mscorlib>			
--	--	--	--

624ad4b4	1	28	System.Text.StringBuilder
----------	---	----	---------------------------

624ab0b4	1	28	System.SharedStatics
----------	---	----	----------------------

6247c1b8	1	28	
----------	---	----	--

System.Text.DBCSCodePageEncoding+DBCSDecoder			
--	--	--	--

6245f94c	1	28	Microsoft.Win32.Win32Native+InputRecord
----------	---	----	---

6245f664	1	28	System.Text.EncoderNLS
----------	---	----	------------------------

624ade68	1	32	
----------	---	----	--

System.Security.Policy.PEFileEvidenceFactory			
--	--	--	--

624acc10	1	32	System.Text.UnicodeEncoding
----------	---	----	-----------------------------

624ab938	1	36	System.Security.PermissionSet
----------	---	----	-------------------------------

624aced8	2	40 Microsoft.Win32.SafeHandles.SafeFileHandle
624ab7b0	1	40 System.Security.Policy.Evidence
624aaa64	1	44 System.Threading.ReaderWriterLock
6247cd1c	1	44
System.Text.InternalEncoderBestFitFallbackBuffer		
624aab90	1	48 System.Collections.Hashtable+bucket[]
620c2348	1	48
System.Collections.Generic.Dictionary<System.String, mscorlib>, [System.Globalization.CultureData, mscorlib]]		
620c2268		
1	48	System.Collections.Generic.Dictionary<System.Type, mscorlib>, [System.Security.Policy.EvidenceTypeDescriptor, mscorlib]]
624acf98	1	52 System.Collections.Hashtable
624ab8d8	1	52 System.Threading.Thread
624acb20	2	56 System.Reflection.RuntimeAssembly
6245f994	2	56 System.IO.__ConsoleStream
624adaa8	1	60 System.IO.StreamWriter
624ad7b4	1	60
System.Collections.Generic.Dictionary<Entry<System.String, mscorlib>, [System.Globalization.CultureData, mscorlib]]>		
6249fbec		
1	64	System.IO.StreamReader
624ab4e4	1	68 System.AppDomainSetup
6247c624	1	76 System.Text.DBCSCodePageEncoding
624ad474	1	84 System.Globalization.CalendarData
624ab060	7	84 System.Object
624aafe4	1	84 System.ExecutionEngineException
624aafa0	1	84 System.StackOverflowException
624aaf5c	1	84 System.OutOfMemoryException
624aae08	1	84 System.Exception
624ab130	1	112 System.AppDomain
624ad164	2	144 System.Globalization.CultureInfo
624ab028	2	168 System.Threading.ThreadAbortException
624ad82c	2	264 System.Globalization.NumberFormatInfo
624aa9f8	1	284
System.Collections.Generic.Dictionary<Entry<System.Type, mscorlib>, [System.Security.Policy.EvidenceTypeDescriptor, mscorlib]]>		
624ac448	8	484 System.Int32[]
624ad3a0	2	616 System.Globalization.CultureData
624abe78	26	728 System.RuntimeType

624ab680	7	2910	System.Char[]
6245ab98	25	18064	System.Object[]
624aacc0	3283	85972	System.String
00363a78	7	2031754	Free
624696f8	2	2097184	System.Byte[][]
624acf54	301232	304844554	System.Byte[]

最后一个显然内存占用比较高，占了304844554 byte，如果你想在此情况下知道对象的内存地址你就直接使用!dumpheap，不带任何参数。由于此命令会导致很多输出，我这里就写出输出内容了。通过!dumpheap 会得到内存很高的对象地址，02d55368，这个地址就是System.Byte[]对象，为了找到对象在哪里分配的，我们需要使用!gcroot 02d55368，命令，查看对象的根在哪里。

```
0:000> !gcroot 02d55368
Thread 143310:      0028f364
004f0100 OrderManager.Program.Main(System.String[])
[e:NETDebugDebugDemoProjectOrderManagerProgram.cs @ 22]
ebp+18: 0028f380
-> 01b746c0 System.Collections.Generic.List1[[System.Byte[], mscorlib]]
-> 02d55368 System.Byte[][]
```

知道了根就好办多了，直接看源代码就能发现问题。如果你还不死心的话可以使用!dumpobj 查看List对象。

```
0:000> !dumpobj 01b746c0
Name:
System.Collections.Generic.List1[[System.Byte[], mscorlib]]
MethodTable:
00163170
EEClass:      6211c8b0
Size:         24(0x18) bytes
File:
C:\Windows\Microsoft.Net\assembly\GAC_32mscorlibv4.0_4.0.0.0__b77a5c561934e089mscorlib.dll
Fields:
MT      Field      Offset                                     Type VT      Attr      Value
Name
6245ab98 4000c75          4          System.Object[]  0 instance 02d55368 _items
624ac480 4000c76          c          System.Int32    1 instance 301229
_size
624ac480 4000c77         10          System.Int32    1 instance 301229
_version
624ab060 4000c78 8 System.Object 0 instance 00000000 _syncRoot
```



```
6245ab98 4000c79 0 System.Object[] 0 shared static
_emptyArray
```

```
>> Domain:Value dynamic statics NYI 00359520:NotInit
```

这里需要注意的是，如果你是想执行!Clrstack -a 命令的话，当你使用调试器启动或者是附加进程的方式的话，要记住切换到适当的线程上才能看行。

7.2. 线程问题（CPU过高，线程死锁）

CPU过高也是线上比较棘手的问题之一，查看CPU过高的步骤一般分为两步，查看线程的执行时间，然后切换到线程上下文，执行!ClrStack -a，看当前线程在哪里工作，到底做什么操作呢。

```
0:004> !runaway
```

```
User Mode Time
```

Thread	Time
0:143310	0 days 0:00:01.934
4:142ac0	0 days 0:00:00.046
7:143874	0 days 0:00:00.000
6:143870	0 days 0:00:00.000
5:14386c	0 days 0:00:00.000
3:1432ec	0 days 0:00:00.000
2:143384	0 days 0:00:00.000
1:143254	0 days 0:00:00.000

测试线程ID为0的执行时间比较大，我们需要切换到线程0上去执行查看调用堆栈信息，~0s。

```
0:000> !ClrStack -a
```

```
0028f348 62b897f9 System.IO.TextWriter+SyncTextWriter.WriteLine(Int32)
```

```
PARAMETERS:
```

```
this () = 0x01b74258 value =
```

```
0028f358 62a66313 System.Console.WriteLine(Int32)
```

```
PARAMETERS:
```

```
value =
```

```
0028f364 004f0100 OrderManager.Program.Main(System.String[])
```

```
[e:NETDebugDebugDemoProjectOrderManagerProgram.cs @ 22] PARAMETERS:
```

```
args (0x0028f38c) = 0x01b71fe4
```

LOCALS:

0x0028f380 = 0x01b746c0

0x0028f388 = 0x000498ac

0x0028f37c = 0x16a2e338

0x0028f384 = 0x00000001

0028f51c 63162952 [GCFrame: 0028f51c]

我们会发现在Main方法中有一个本地变量0x0028f380，保存的值是0x01b746c0，它就是指向上次分配很多内存的List对象。

线程死锁比较复杂，这里只给我认为比较简单的命令，通过此命令可以一眼看出哪个线程持有了哪个锁，目前在等待哪个锁。

0:000> !syncblk

Index	SyncBlock	MonitorHeld	Recursion	Owning Thread	Info
4	0021fb20	3	1	00221f98 14974c	3 01ae2394
OrderManager.ImportOrder					
5	0021fb54	3	1	002234a8 149754	4 01ae23a0
OrderManager.ImportOrder					

Total					
5					
CCW 0					
RCW 0					
ComClassFactory					
0					
Free 0					

这是两个锁，也就是两个对象同步块。进一步使用SOSEX.dll中的!dlk查看死锁的自动化检查信息。

0:000> !dlk

Examining SyncBlocks... Scanning for ReaderWriterLock instances... Scanning for holders of ReaderWriterLock locks... Scanning for ReaderWriterLockSlim instances... Scanning for holders of ReaderWriterLockSlim locks... Examining CriticalSections... Could not find symbol ntdll!RtlCriticalSectionList. Scanning for threads waiting on SyncBlocks... Scanning for threads waiting on ReaderWriterLock locks... Scanning for threads waiting on ReaderWriterLockSlim locks... Scanning for threads waiting on CriticalSections... *DEADLOCK DETECTED* CLR thread 0x3 holds the lock on SyncBlock 0021fb20

```
OBJ:01ae2394[OrderManager.ImportOrder] ...and is waiting for the lock on SyncBlock
0021fb54 OBJ:01ae23a0[OrderManager.ImportOrder] CLR thread 0x4 holds the lock on
SyncBlock 0021fb54 OBJ:01ae23a0[OrderManager.ImportOrder] ...and is waiting for the lock
on SyncBlock 0021fb20 OBJ:01ae2394[OrderManager.ImportOrder] CLR Thread 0x3 is waiting at
System.Threading.Monitor.Enter(System.Object, Boolean ByRef) (+0x17 Native) CLR Thread 0x4
is waiting at System.Threading.Monitor.Enter(System.Object, Boolean ByRef) (+0x17 Native)
```

1 deadlock detected.

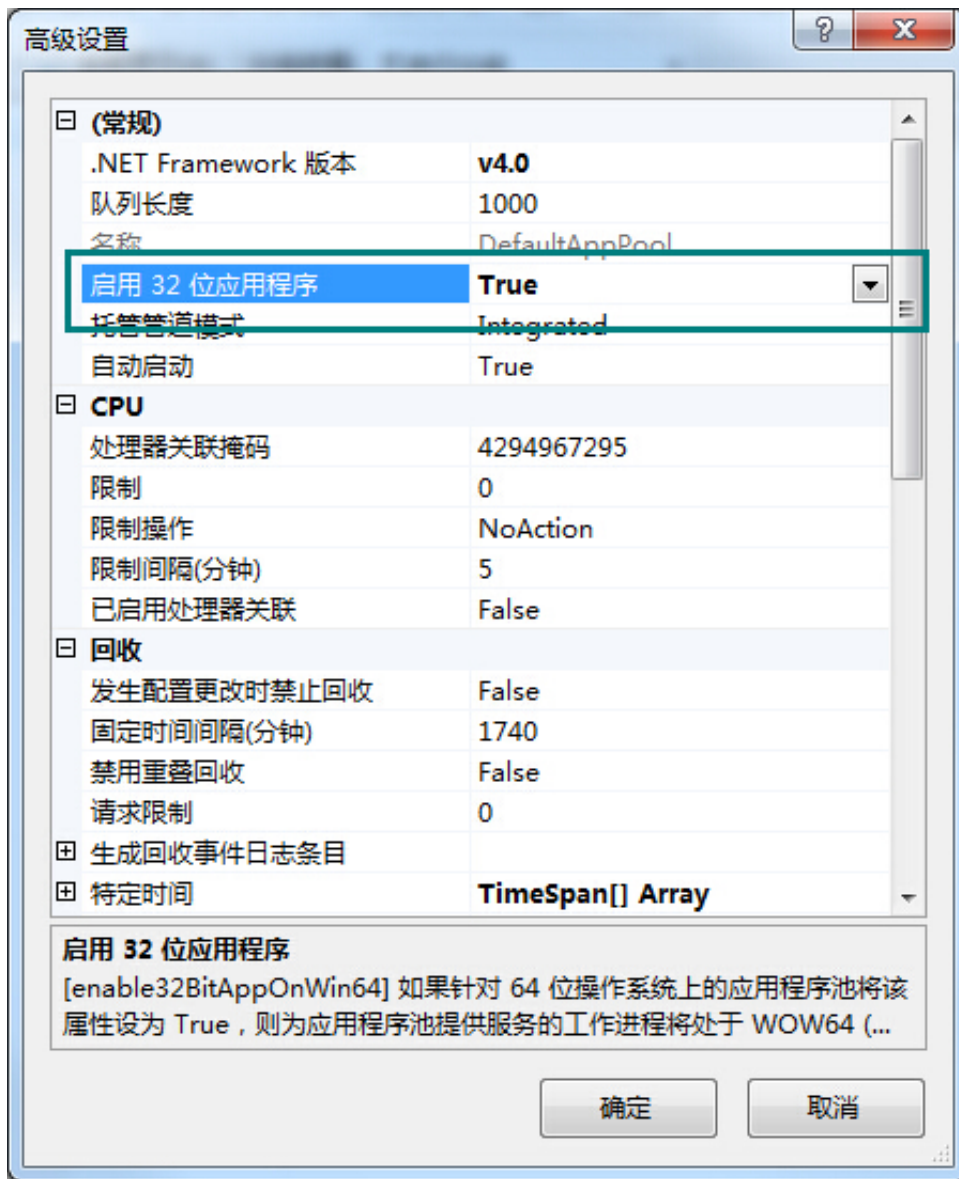
注意我加粗的那段话，检测到死锁。

8. 获取Dump文件时的重要注意事项

在获取dump文件方面我也要分享一下重要的注意事项。如果获取dump文件不正确的话是无法进行分析的，会出现任何奇怪的问题。

第一个就是使用64位机器上的任务管理获取32位进程dump文件，这通常是发生在服务器上，由于服务器IIS默认的启动进程方式是64位的，但是也有些情况下会变成32位的。

图19:



如果进程是以32位方式运行的，那么这个时候获取出来的dump文件是不好分析的，此时应该使用调试器工具进行dump的获取。获取出来的dump文件和分析机器上的调试器环境不一致的情况下会出现如下几个错误。

图20:

```
HistInit
HistRoot
HistObj
HistObjFind
HistClear
0:000> !eeversion
Failed to load data access DLL, 0x80004005
Verify that 1) you have a recent build of the debugger (6.2.14 or newer)
2) the file mscordacwks.dll that matches your version of clr.dll is
in the version directory or on the symbol path
3) or, if you are debugging a dump file, verify that the file
mscordacwks_<arch>_<arch>_<version>.dll is on your symbol path.
4) you are debugging on supported cross platform architecture as
the dump file. For example, an ARM dump file must be debugged
on an X86 or an ARM machine; an AMD64 dump file must be
debugged on an AMD64 machine.

You can also run the debugger command .cordll to control the debugger's
load of mscordacwks.dll. .cordll -ve -u -l will do a verbose reload.
If that succeeds, the SOS command should work on retry.

If you are debugging a minidump, you need to make sure that your executable
path is pointing to clr.dll as well.
```

这个问题是未能加载正确版本的mscordacwks.dll .NETDAC调式组件。

图21:

```
0:000> !eeversion
SOS does not support the current target architecture.
```

这个问题是当前SOS.dll和.NET程序所使用的.NET版本不一致，这个问题的出现一般都是我们通过.load xxxxSOS.dll，手动方式加载的。

图22:

```
00000000`76a80000 00000000`76b70000  rpcrt4      (deferred)
00000000`76ba0000 00000000`76c6c000  msctf       (deferred)
00000000`76cd0000 00000000`76cf7000  cfgmgr32    (deferred)
00000000`76d00000 00000000`76d57000  shlwapi     (deferred)
00000000`76d60000 00000000`76da5000  Wldap32     (deferred)
00000000`774b0000 00000000`77659000  ntdll       (pdb symbols)      d:\ntdll.pdb\400F215C54DA404788F84F5C504914952\ntdll.pdb
00000000`77660000 00000000`77666000  nsi         (deferred)
00000000`77690000 00000000`77810000  ntdll_77690000 (deferred)
00000000`79be0000 00000000`7a55c000  System.ni   (deferred)
0:000> loadby sos.dll clrjit
The call to LoadLibrary(C:\Windows\Microsoft.NET\Framework\v4.0.30319\sos.dll) failed, Win32 error 193
"%1 不是有效的 Win32 应用程序。"
Please check your debugger configuration and/or network access.
```

这个问题出现有好几种可能性，对常见的问题就是未能使用正确的方法或者工具获取dump文件，导致dump文件获取的机器和本地调试的机器整个环境不一致。

9. 总结

本篇文章分享我对.NET应用程序调试方面学习和实践的一些经验，供广大博友参考。如果想系统的学习一下这方面的知识可以参考《.NET高级调试》一书，此书非常底层，对.NET运行时原理讲的很透彻，可以作为深入学习.NET的一门参考书。

作者: [王清培](#)

出处: <http://www.cnblogs.com/wangiqngpei557/>

本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《[招募专栏作者](#)》

1 赞 收藏 [评论](#)



合作联系

Email: bd@jobbole.com

QQ: 2302462408 (加好友请注明来意)

更多频道

- [小组](#) - 好的话题、有启发的回复、值得信赖的圈子
- [头条](#) - 分享和发现有价值的内容与观点
- [相亲](#) - 为IT单身男女服务的征婚传播平台

- [资源](#) - 优秀的工具资源导航
- [翻译](#) - 翻译传播优秀的外文文章
- [文章](#) - 国内外的精选文章
- [设计](#) - UI, 网页, 交互和用户体验
- [iOS](#) - 专注iOS技术分享
- [安卓](#) - 专注Android技术分享
- [前端](#) - JavaScript, HTML5, CSS
- [Java](#) - 专注Java技术分享
- [Python](#) - 专注Python技术分享