開発者必読！

“もしも”のときのデバッグ技法

.NETアプリケーション障害解析

株式会社NTTデータ
飯山 教史
IIYAMA, Takashi

第４回

COMコンポーネントで例外が発生したときの解析

Level

1 2 3 **4** 5

Technology Tools

- ❑ Visual Basic
- ☑ Visual C#
- ❑ Visual C++
- ❑ SQL Server
- ❑ Oracle
- ❑ Access
- ❑ ASP.NET
- ☑ Other:
  - WinDbg（Debugging Tools for Windows）
  - SOS.DLL
  - Visual Studio 6.0

Samples

レガシー環境との共存

前回はWinDbg（Microsoft Debugging Tools for Windows）とSOS.DLLを利用して.NETアプリケーションを解析する方法を説明した。.NETは順調に市場への導入が進められており、「新規システムは.NETで」というプロジェクトも多いと聞く。そのため.NETのみで開発されるシステムが今後増加することが予想され、その意味では先月取り上げた.NETアプリケーションのSOS.DLLを利用した解析は今後その意義を増していくと思う。

しかし、Windowsシステムの多くではレガシーな開発言語（VC++6.0、VB 6.0など）で開発されたCOMコンポーネントやDLLを.NETから呼び出して利用しているのが現状である。そこで今回は.NETから、これらのレガシーなモジュールを呼び出したときの解析手法について見てみることにする。

具体的には、COMコンポーネントで例外を起こす簡単なプログラムを開発し、それを.NETのプログラムから呼び出すアプリケーションを利用した際に、SOS.DLLで表示されるダンプ情報とその意味について説明する

.NETとCOMコンポーネントの連携

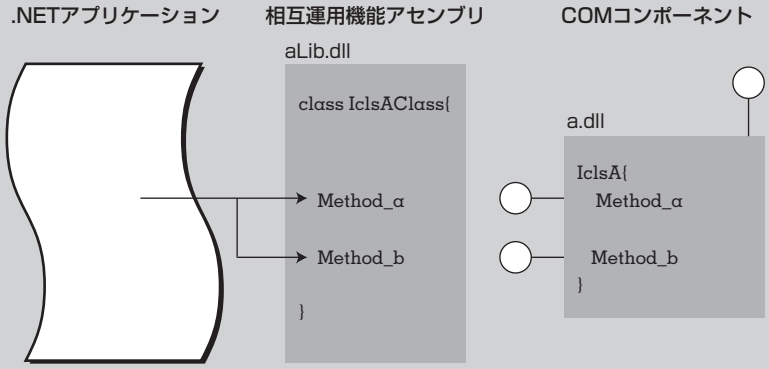
.NETからCOMコンポーネントを呼び出すため、Microsoftはランタイム呼び出し可能ラッパー（Runtime Callable Wrapper：RCW）を用意した。RCWはCOMコンポーネントへのプロキシの役割を果たす[注1]。今回はVisual Studio .NETの「参照の追加」の「COM」タブでCOMコンポーネントを追加する方法で、.NETアプリケーションからCOMコンポーネントを呼び出す構成にした。この方法を使えば簡単に「<COMモジュール名>Lib.dll」と命名される相互運用機能アセンブリが作成され、.NETの

注1）.NETの相互運用性については以下のURLを参照。
http://www.microsoft.com/japan/msdn/net/bda/cominterop.asp

コードからCOMコンポーネントのメソッドを呼び出せるようになる。この構成をまとめると図1のようになる。

今回利用したコードもいつもどおり非常にシンプルである。COMコンポーネントでは単にRaiseExceptionを呼び出し例外を起こしている。.NET側ではそのCOMをラップしたクラスをインスタンス化し、例外を起こすメソッドを呼び出している。ADPlusを利用してuser.dmpを作るときには前回と同様に以下の設定が必要になる。

図1：.NETアプリケーションからのCOMコンポーネント呼び出し



.NETアプリケーションからは“IclsAClass”をインスタンス化し、メソッドを呼び出すイメージになる

設定1：machine.config

<SystemRoot>¥Microsoft.NET¥Framework¥v1.1.4322¥CONFIGディレクトリ配下にある「machine.config」を開き、system.windows.formsセクションのjitDebuggingの値を「true」にする（デフォルトはfalse）。

```
<configration>
  ・
  ・
  <system.windows.forms
   jitDebugging="true" />
```

設定2：レジストリ

レジストリエディタを起動し、HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥.NETFrameworkキーのDbgJITDebugLaunchSettingの値を「0」にする。

▶ user.dmp解析の際に注意すべきこと

Visual Studio 6.0で開発されたモジュールを.NETから呼び出しているアプリケーションのuser.dmpの解析を進めるときには、この他にも2点ほど注意点があるので先にそれらの説明をしておく。

ひとつ目の注意点は、リリースビルドされた.NETアプリケーションから呼び出されたアンマネージドコード（リリースビルド）のDLLで例外が発生すると、アンマネージドコードのスタック情報が得られないことである。このため先月号同様、「!threads」コマンドを実行してもどのスレッドで例外が発生したのかは表示されない。また、「!clrstack」コマンドを実行しても、どのメソッドで例外が発生しているのかを特定できないのである。このような制限があるため、.NETアプリケーションでレガシーなCOMコンポーネントが例外を起こしている事例を解析するときには.NETアプリケーションをDebugバージョンでビルドし直して事例を再現しなくてはならない。

2つ目の注意点はADPlusの利用方法である。前回はcrashオプションを利用したが、今回はhangオプションを利用してuser.dmpを取得する。具体的には、例外が発生したことを示す画面が表示され、アプリケーションが停止している状態でADPlusをhangオプションで実行する。hangオプションで得られたuser.dmpは、crashオプションで作成されるuser.dmpと同一のフォーマットで作成されるため、WinDbg＋SOS.DLLの組み合わせで解析できる。

SOS.DLLでスタック情報をダンプ

今回も利用するコマンドは「!threads」と「!dumpstack」である。まずリスト1とリスト2のコードを実行して得られた

リスト1：例外を起こすCOMコンポーネントのコード

```
STDMETHODIMP CNetExcpt::RaiseExcpt()
{
   RaiseException(EXCEPTION_ACCESS_VIOLATION, 0, 0, NULL);
   return S_OK;
}
```