WinDBG基础----了解Symbols

转载

枫★曦 于 2015-07-21 15:57:05 发布 2084 收藏
分类专栏： Windows


Windows 专栏收录该内容
45 篇文章 0 订阅
订阅专栏

对于NET下的开发人员，可能对Symbol不了解的人还不少，因为MS给了我们太多方便的工具，让我们只需要去关注代码，对于其他的东西基本上不需要去关注，所以就养成了一个习惯（过多的依赖MS，依赖VS）。这里只是抛开VS，讲一下调试要用到的东西-Symbol。

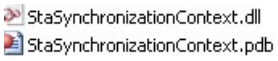
认识Symbol

用于程序调试的数据，它包含了调试中需要用到的各种数据，例如：全局变量、本地变量、函数名、函数类型、源代码行、程序入口地址.....，这些所有的东西都叫做Symbol。

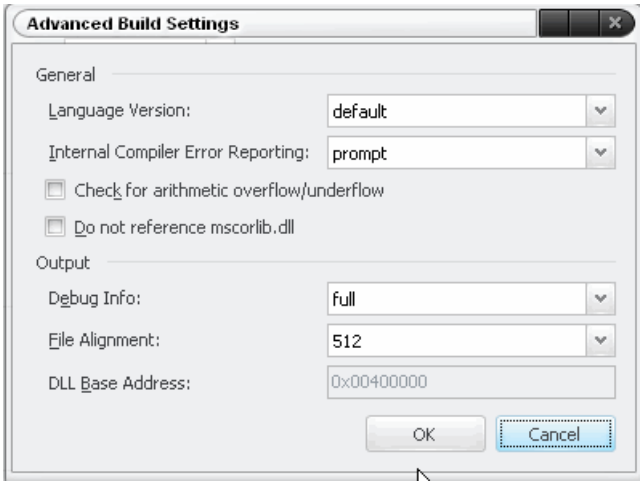
在windows系统中，windows2000将这些信息保存在.pdb和.dbg这些的文件中，而windowsXP和以后的版本都将这些信息保存在.pdb文件中。

Symbol一般分为两种：Public和Private，其实我们应该很容易理解他们，就是允许公开的数据和私有的数据。我们在发布产品的时候，可以同时发布Symbol数据，对于Symbol数据发布的粒度，我们也是可以去查看和控制的（下面会介绍PDBCopy和Symchk）。

在NET中，我们也可以看到VS生成的Symbols，在VS生成的DLL目录下面，我们除了DLL外，还可以看到另外的一类文件.pdb（如下图），VS能提供给我们这么大的调试能力和它是分不开的。



同时，我们也可以在VS中去设定是否输出Symbols或输出Public或Private的Symbols。如下图，在"Debug Info"中设置



通过Windbg查看Symbols

在Windbg查看Symbols前，我们可以设置一些Symbol Option：

例如：

SYMOPT_NO_PUBLICS：屏蔽Public的Symbol [.symbol +0x8000]

SYMOPT_AUTO_PUBLICS：Public和Private的Symbol [.symbol +0x4000]

SYMOPT_PUBLICS_ONLY：屏蔽Private的Symbol [.symbol +0x10000]

SYMOPT_DEBUG：当Windbg加载Symbol文件的时候，显示Symbol的路径，默认情况下是不显示的。

打开命令[!sym noisy]

```
Reloading current modules
....
DBGHELP: e:\ccg code\code repository\ccg_cn003_contract\productcode\zte\ccg\contract\ui\win\
DBGHELP: e:\ccg code\code repository\ccg_cn003_contract\productcode\zte\ccg\contract\ui\win\
DBGHELP: e:\ccg code\code repository\ccg_cn003_contract\productcode\zte\ccg\contract\ui\win\
DBGHELP: C:\WINDOWS\system32\ntdll.pdb - file not found
DBGHELP: ntdll.pdb - file not found
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
DBGHELP: ntdll - export symbols
```

关闭命令[!sym quiet]

```
Reloading current modules
....
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for ntdll.dll -
```

通过上面两个截图应该很容易看到区别了。

下面让我们用具体的命令来查看Symbol数据：

1、lm(命令):显示出程序运行加载的模块信息

```
0:000> lm
start    end        module name
00400000 00424000   Main       (deferred)
79000000 79046000   mscoree    (deferred)
7c800000 7c91d000   KERNEL32   (deferred)
7c920000 7c9b4000   ntdll      (export symbols)       C:\WINDOWS\system32\ntdll.dl
```

2、!lmi [模块名]：显示模块的详细信息，并且还有命令(!db [模块名])，它显示的信息要比!lmi多些

```
0:000> !lmi main
Loaded Module Info: [main]
         Module: Main
   Base Address: 00400000
     Image Name: Main.exe
   Machine Type: 332 (I386)
     Time Stamp: 4999178c Mon Feb 16 15:36:44 2009
           Size: 24000
       CheckSum: 0
Characteristics: 10e
Debug Data Dirs: Type  Size     VA  Pointer
             CODEVIEW    87, 1c8d0,   1b8d0 RSDS - GUID: {65F1F678-1663-4A22-97C4-952206F56024}
               Age: 8fc, Pdb: E:\CCG Code\Code Repository\CCG_CN003_Contract\ProductCode\ZTE\CCG\Cont
    Symbol Type: DEFERRED - No error - symbol load deferred
    Load Report: no symbols loaded
```

3、X modulename!symbols(命令):显示出所有的Symbols数据或者是指定模块的Symbols数据

```
0:000> x main!*
*** WARNING: Unable to verify checksum for Main.exe
<MSIL:00401d49> Main!txtPwd_KeyPress (void)
<MSIL:00401ecd> Main!cmbCulture_SelectedIndexChanged (void)
<MSIL:004003ff> Main!UpdateLoader (void)
<MSIL:00400000> Main!Main (void)
<MSIL:00401cb2> Main!lblCHNCode_Click (void)
<MSIL:00400578> Main!.ctor (void)
<MSIL:0040036d> Main!Application_ThreadException (void)
<MSIL:00401f50> Main!LoginFrm_HelpRequested (void)
```

设置Windbg中的Sympath

在Windbg中，运行上面的命令去查看Symbols数据，那么Windbg在什么地方去搜索这些数据呢？有些编译器会将这些pdb文件和dll文件或exe文件放在同一个目录下面，例如Visual Studio，所以通过VS调试的时候，VS会在dll的目录中搜索，不同用Windgb来调试程序的话，就需要我们来自己设置下搜索路径。

1、设置Windbg的环境变量： *_NT_SYMBOL_PATH* 和 *_NT_ALT_SYMBOL_PATH*

2、在命令行启动Windbg的时候，通过-y(command line)来设置，例如：windbg -y path

3、.sympath和.symfix

.sympath[+] [Path [; ...]]：指定一个新的路径

.sympath：查看设置的路径

```
0:000> .sympath
Symbol search path is: E:\CCG Code\Code Repository\CCG_CN003_Contract\ProductCode\ZTE\CC
Expanded Symbol search path is: e:\ccg code\code repository\ccg_cn003_contract\productcc
```

.Symfix[+] [path]：设置指向Microsoft symbol store的Symbols文件路径

等于sympatp[+] srv*DownstreamStore*http://msdl.microsoft.com/download/symbols(MS中所有Symbol文件的存放地址)

4、通过*Windbg*的图形界面*"File | Symbol file path"*也可以设置

重新加载Symbol以及Symbol的状态

Symbol的状态，我们可以先看下上面的图(lm命令)，在每个模块名称后面都有deferred、export。下面简单说下它的几个状态：

deferred：模块已经加载，但是模块的symbol文件并没有加载，这是属于延迟加载的，当需要的时候才加载，或者我们通过ld［模块名称］(命令)来加载指定模块的symbol。

export：没有对应的symbol文件，目前只能把dll或exe文件当做symbol来加载

private：表示加载的是私有的Symbol

public：表示加载的是共有的Symbol

```
module name
Main     C (private pdb symbols)  e:\ccg code\code repository\ccg_c
mscoree    (export symbols)       C:\WINDOWS\system32\mscoree.dll
KERNEL32   (deferred)
ntdll      (export symbols)       C:\WINDOWS\system32\ntdll.dll
```

.reload（命令）：重新加载Symbol

当有些时候我们已经加载了symbol，但是该文件不存在，这时候我们把symbol文件从其他地方拷贝过来后，我们可以用该命令使windbg重新加载。

Symchk和PDBCopy

Symchk：用来检测Symbol文件和执行文件是否匹配

symchk [/r] FileNames /s SymbolPath

看下下面的执行结果：

```
e:\debuggers> symchk /r c:\windows\system32 /s srv*\\manysymbols\windows

SYMCHK: msisam11.dll         FAILED  - MSISAM11.pdb is missing
SYMCHK: msuni11.dll          FAILED  - msuni11link.pdb is missing
SYMCHK: msdxm.ocx            FAILED  - Image is split correctly, but msdxm.dbg i
s missing
SYMCHK: expsrv.dll           FAILED  - Checksum doesn't match with expsrv.DBG
SYMCHK: imeshare.dll         FAILED  - imeshare.opt.pdb is missing
SYMCHK: ir32_32.dll          FAILED  - Built with no debugging information
SYMCHK: author.dll           FAILED  - rpctest.pdb is missing
SYMCHK: msvcrt40.dll         FAILED  - Built with no debugging information
------
SYMCHK: FAILED files = 211
SYMCHK: PASSED + IGNORED files = 4809
```

PDBCopy：用来分离Symbol，它可以将一个完整的Symbol文件分离成Public和Private的Symbol文件

1、把Private的Symbol删除，创建只有Public的Symbol

pdbcopy mysymbols.pdb publicsymbols.pdb -p

2、不仅删除Private的symbol，而且还可以删除Public中的一些指定数据

pdbcopy mysymbols.pdb publicsymbols.pdb -p -f:@c:\delete.txt

delete.txt就是指定需要删除的数据，例如希望去删除_myGlobal1和_myGlobal2这两个数据，那么你在该文件中只需要输入两行：

_myGlobal1

_myGlobal2

由于Windbg在国内的资料很少，所以自己边学边写点东西，虽然这些都是些简单的基础知识，但是希望和大家一起慢慢玩通它，呵呵！