

# WinDbg (用户模式) 入门

WinDbg 是 Windows 调试工具中包含的内核模式和用户模式调试器。 以下动手练习可帮助你开始使用 WinDbg 作为用户模式调试器。

有关如何获取 Windows 调试工具的信息，请参阅 [下载并安装 WinDbg Windows 调试器](#)。

安装调试工具后，找到 64 位 (x64) 和 32 位 (x86) 版本的安装目录。 例如：

- C:\Program Files (x86)\Windows Kits\10\Debuggers\x64
- **C: \Program Files (x86) \Windows Kits\10\Debuggers\x86**

## 打开记事本并附加 WinDbg

1. 转到安装目录，打开 **WinDbg.exe**。
2. 在“文件”菜单上，选择“启动可执行文件”。 在“启动可执行文件”对话框中，转到包含 notepad.exe 的文件夹。（notepad.exe 文件通常位于 C: \Windows\System32。）对于“文件名”，请输入 **notepad.exe**。 选择“打开”。

3. 在 WinDbg 窗口底部附近的命令行中，输入以下命令：

[.sympath srv\\*](#)

输出类似于以下示例：

dbgcmd 复制

```
Symbol search path is: srv*
```

```
Expanded Symbol search path is: cache*;SRV
```

符号搜索路径指示 WinDbg 查找符号 (PDB) 文件的位置。调试器需要符号文件来获取有关代码模块的信息，例如函数名称和变量名称。

然后，输入以下命令：

[.reload](#)

命令 `.reload` 指示 WinDbg 执行初始搜索以查找和加载符号文件。

- 若要查看 `notepad.exe` 模块的符号，请输入以下命令：

[x notepad!\\*](#)

### 备注

如果未显示任何输出，请输入 `.reload /f` 以尝试强制加载符号。

使用 `!sym noisy` 显示其他符号加载信息。

若要查看包含 `main` 的 `notepad.exe` 模块中的符号，请使用 [检查符号](#) 命令列出与掩码匹配的模块：

```
x notepad!wWin*
```

输出类似于以下示例：

dbgcmd 复制

```
00007ff6`6e76b0a0 notepad!wWinMain (wWinMain)
00007ff6`6e783db0 notepad!wWinMainCRTStartup
(wWinMainCRTStartup)
```

5. 若要在 处 notepad!wWinMain 放置断点，请输入以下命令：

[bu notepad!wWinMain](#)

若要验证是否已设置断点，请输入以下命令：

[bl](#)

输出类似于以下示例：

dbgcmd 复制

```
0 e Disable Clear 00007ff6`6e76b0a0 0001 (0001)
0:**** notepad!wWinMain
```

6. 若要启动记事本进程，请输入以下命令：

[g](#)

记事本将一直运行，直到它进入 WinMain 函数，然后它进入调试器。

dbgcmd 复制

```
Breakpoint 0 hit
notepad!wWinMain:
00007ff6`6e76b0a0 488bc4 mov rax, rsp
```

若要查看记事本进程中当前加载的代码模块的列表，请输入以下命令：

[lm](#)

输出类似于以下示例：

dbgcmd 复制

```
0:000> lm
start                end                module name
00007ff6`6e760000 00007ff6`6e798000  notepad      (pdb
symbols)
C:\ProgramData\Dbg\sym\notepad.pdb\BC04D9A431EDE299D4
625AD6201C8A4A1\notepad.pdb
00007ff8`066a0000 00007ff8`067ab000  gdi32full
(deferred)
00007ff8`067b0000 00007ff8`068b0000  ucrtbase
(deferred)
00007ff8`06a10000 00007ff8`06aad000  msvc_p_win
(deferred)
00007ff8`06ab0000 00007ff8`06ad2000  win32u
(deferred)
00007ff8`06b40000 00007ff8`06e08000  KERNELBASE
(deferred)
00007ff8`07220000 00007ff8`072dd000  KERNEL32
(deferred)
00007ff8`07420000 00007ff8`07775000  combase
(deferred)
00007ff8`07820000 00007ff8`079c0000  USER32
(deferred)
00007ff8`079c0000 00007ff8`079f0000  IMM32
(deferred)
00007ff8`07c00000 00007ff8`07c2a000  GDI32
(deferred)
00007ff8`08480000 00007ff8`085ab000  RPCRT4
(deferred)
00007ff8`085b0000 00007ff8`0864e000  msvcrt
(deferred)
00007ff8`08c40000 00007ff8`08cee000  shcore
(deferred)
00007ff8`08db0000 00007ff8`08fa5000  ntdll        (pdb
symbols)
C:\ProgramData\Dbg\sym\ntdll.pdb\53F12BFE149A2F50205C
8D5D66290B481\ntdll.pdb
00007fff`f8580000 00007fff`f881a000  COMCTL32
(deferred)
```

若要查看堆栈跟踪，请输入以下命令：

[k](#)

输出类似于以下示例：

dbgcmd 复制

```
0:000> k
00 000000c8`2647f708 00007ff6`6e783d36
notepad!wWinMain
01 000000c8`2647f710 00007ff8`07237034
notepad!__scrt_common_main_seh+0x106
02 000000c8`2647f750 00007ff8`08e02651
KERNEL32!BaseThreadInitThunk+0x14
03 000000c8`2647f780 00000000`00000000
ntdll!RtlUserThreadStart+0x21
```

7. 若要再次启动记事本运行，请输入以下命令：

[g](#)

8. 若要进入记事本，请在“文件”菜单上，选择“中断”。
9. 若要在 `ZwWriteFile` 设置和验证断点，请输入以下命令：

[bu ntdll!ZwWriteFile](#)

[bl](#)

10. 若要再次启动记事本运行，请输入 [g](#)。在记事本窗口中，输入一些文本。在“文件”菜单中，选择“保存”。当涉及到 `ZwCreateFile` 时，正在运行的代码会中断。输入 [k](#) 命令以查看堆栈跟踪。

在命令行左侧的 WinDbg 窗口中，显示处理器和线程编号。 在此示例中，当前处理器编号为 0，当前线程编号为 11 (0:011>)。

窗口显示处理器 0 上运行的线程 11 的堆栈跟踪。

11. 若要查看记事本进程中所有线程的列表，请输入以下命令（波形符）：

~

输出类似于以下示例：

dbgcmd 复制

```
0:011> ~
  0 Id: 5500.34d8 Suspend: 1 Teb: 000000c8`262c4000
Unfrozen
  1 Id: 5500.3960 Suspend: 1 Teb: 000000c8`262c6000
Unfrozen
  2 Id: 5500.5d68 Suspend: 1 Teb:
000000c8`262c8000 Unfrozen
  3 Id: 5500.4c90 Suspend: 1 Teb:
000000c8`262ca000 Unfrozen
  4 Id: 5500.4ac4 Suspend: 1 Teb:
000000c8`262cc000 Unfrozen
  5 Id: 5500.293c Suspend: 1 Teb:
000000c8`262ce000 Unfrozen
  6 Id: 5500.53a0 Suspend: 1 Teb:
000000c8`262d0000 Unfrozen
  7 Id: 5500.3ca4 Suspend: 1 Teb:
000000c8`262d4000 Unfrozen
  8 Id: 5500.808 Suspend: 1 Teb: 000000c8`262da000
Unfrozen
 10 Id: 5500.3940 Suspend: 1 Teb:
000000c8`262dc000 Unfrozen
. 11 Id: 5500.28b0 Suspend: 1 Teb:
000000c8`262de000 Unfrozen
 12 Id: 5500.12bc Suspend: 1 Teb:
000000c8`262e0000 Unfrozen
 13 Id: 5500.4c34 Suspend: 1 Teb:
000000c8`262e2000 Unfrozen
```

在此示例中，14 个线程的索引为 0 到 13。

12. 若要查看线程 0 的堆栈跟踪，请输入以下命令：

[~0s](#)

[k](#)

输出类似于以下示例：

dbgcmd 复制

```
0:011> ~0s
```

```
0:011> ~0s
```

```
win32u!NtUserGetProp+0x14:
```

```
00007ff8`06ab1204 c3 ret
```

```
0:000> k
```

| #  | Child-SP          | RetAddr           | Call Site                                |
|----|-------------------|-------------------|------------------------------------------|
| 00 | 000000c8`2647bd08 | 00007ff8`07829fe1 | win32u!NtUserGetProp+0x14                |
| 01 | 000000c8`2647bd10 | 00007fff`f86099be | USER32!GetPropW+0xd1                     |
| 02 | 000000c8`2647bd40 | 00007ff8`07d12f4d | COMCTL32!DefSubclassProc+0x4e            |
| 03 | 000000c8`2647bd90 | 00007fff`f8609aba | SHELL32!CAutoComplete::_EditWndProc+0xb1 |
| 04 | 000000c8`2647bde0 | 00007fff`f86098b7 | COMCTL32!CallNextSubclassProc+0x9a       |
| 05 | 000000c8`2647be60 | 00007ff8`0782e858 | COMCTL32!MasterSubclassProc+0xa7         |
| 06 | 000000c8`2647bf00 | 00007ff8`0782de1b | USER32!UserCallWinProcCheckWow+0x2f8     |
| 07 | 000000c8`2647c090 | 00007ff8`0782d68a | USER32!SendMessageWorker+0x70b           |
| 08 | 000000c8`2647c130 | 00007ff8`07afa4db | USER32!SendMessageW+0xda                 |

13. 若要退出调试并从记事本进程分离，请输入以下命令：

[qd](#)

## 打开自己的应用程序并附加 WinDbg

例如，假设你已编写并生成了此小型控制台应用程序：

C++复制

```
...  
void MyFunction(long p1, long p2, long p3)  
{  
    long x = p1 + p2 + p3;  
    long y = 0;  
    y = x / p2;  
}  
  
void main ()  
{  
    long a = 2;  
    long b = 0;  
    MyFunction(a, b, 5);  
}
```

在本练习中，假定生成的应用程序 (MyApp.exe) 和符号文件 (MyApp.pdb) 位于 C: \MyApp\x64\Debug 中。此外，假设应用程序源代码位于 C: \MyApp\MyApp 中，并且目标计算机编译 MyApp.exe。

1. 打开 WinDbg。
2. 在“文件”菜单上，选择“启动可执行文件”。在“启动可执行文件”对话框中，转到 C: \MyApp\x64\Debug。对于“文件名”，输入 **MyApp.exe**。选择“打开”。
3. 输入以下命令：

[.symfix](#)



[.sympath](#)+ C: \MyApp\x64\Debug

命令告知 WinDbg 在何处查找应用程序的符号和源代码。 在这种情况下，无需使用 [.srcpath](#) 设置源代码位置，因为符号具有源文件的完全限定路径。

4. 输入以下命令：

[.reload](#)

[bu MyApp!main](#)

[g](#)

应用程序在涉及到其 `main` 函数时会中断到调试器中。

WinDbg 显示源代码和命令窗口。

5. 在“调试”菜单上，选择“单步执行 (或选择 F11)”。继续单步执行，直到单步执行 `MyFunction`。单步执行行 `y = x / p2` 时，应用程序崩溃并中断到调试器中。

输出类似于以下示例：

dbgcmd 复制

```
(1450.1424): Integer divide-by-zero - code c0000094
(first chance)
First chance exceptions are reported before any
exception handling.
This exception may be expected and handled.
```

```
MyApp!MyFunction+0x44:  
00007ff6`3be11064 f77c2428 idiv eax,dword ptr  
[rsp+28h] ss:00000063`2036f808=00000000
```

6. 输入此命令:

[!analyze -v](#)

在本例中, WinDbg 显示 (问题分析, 除以 0) 。

dbgcmd 复制

```
FAULTING_IP:  
MyApp!MyFunction+44 [c:\myapp\myapp\myapp.cpp @ 7]  
00007ff6`3be11064 f77c2428 idiv eax,dword  
ptr [rsp+28h]  
  
EXCEPTION_RECORD: ffffffffffffffff -- (.exr  
0xffffffffffffffff)  
ExceptionAddress: 00007ff63be11064  
(MyApp!MyFunction+0x0000000000000044)  
ExceptionCode: c0000094 (Integer divide-by-zero)  
ExceptionFlags: 00000000  
NumberParameters: 0  
...  
STACK_TEXT:  
00000063`2036f7e0 00007ff6`3be110b8 : ... :  
MyApp!MyFunction+0x44  
00000063`2036f800 00007ff6`3be1141d : ... :  
MyApp!main+0x38  
00000063`2036f840 00007ff6`3be1154e : ... :  
MyApp!__tmainCRTStartup+0x19d  
00000063`2036f8b0 00007ffc`b1cf16ad : ... :  
MyApp!mainCRTStartup+0xe  
00000063`2036f8e0 00007ffc`b1fc4629 : ... :  
KERNEL32!BaseThreadInitThunk+0xd  
00000063`2036f910 00000000`00000000 : ... :  
ntdll!RtlUserThreadStart+0x1d  
  
STACK_COMMAND: dt ntdll!LdrpLastDllInitializer  
BaseDllName ;dt ntdll!LdrpFailureData ;.cxr 0x0 ;kb
```

FOLLOWUP\_IP:  
MyApp!MyFunction+44 [c:\myapp\myapp\myapp.cpp @ 7]  
00007ff6`3be11064 f77c2428 idiv eax,dword  
ptr [rsp+28h]

FAULTING\_SOURCE\_LINE: c:\myapp\myapp\myapp.cpp

FAULTING\_SOURCE\_FILE: c:\myapp\myapp\myapp.cpp

FAULTING\_SOURCE\_LINE\_NUMBER: 7

FAULTING\_SOURCE\_CODE:  
3: void MyFunction(long p1, long p2, long p3)  
4: {  
5: long x = p1 + p2 + p3;  
6: long y = 0;  
> 7: y = x / p2;  
8: }  
9:  
10: void main ()  
11: {  
12: long a = 2;  
...