Windbg是Microsoft公司免费调试器调试集合中的GUI的调试器，支持Source和Assembly两种模式的调试。Windbg不仅可以调试应用程序，还可以进行Kernel Debug。结合Microsoft的Symbol Server，可以获取系统符号文件，便于应用程序和内核的调试。Windbg支持的平台包括X86、IA64、AMD64。

加载符号

```
.sympath   // 查看当前符号查找路径
.sympath c:\symbols   // 将符号查找路径设为：c:\symbols
.sympath+ c:\symbols  // 将c:\symbols添加到符号查找路径集合中

.reload // 为所有已加载模块载入符号信息
.reload /f /v // f:强制立即模式（不允许延迟载入）  v:详细模式
.reload /f @"c:\windows\System32\verifier.dll" // 为指定模块加载符号信息
```

用户进程加载

```
1: kd> !process 0 0 x64.exe  // 根据进程名得到进程详细信息
1: kd> !process 0 0          // 列出系统所有进程
1: kd> !process -1 1         // 查看当前进程信息
1: kd> !process 0 7          // 查看详细进程信息
```

系统模块与PE文件检索

```
0:000> lm            // 列出所有模块对应的符号信息
0:000> lmv           // 列出所有模块对应的符号信息
0:000> lmt           // 列出所有模块的基地址和偏移
0:000> lmf           // 列出所有DLL的具体路径
0:000> lmvm ntdll    // 查看ntdll.dll的详细信息
0:000> !lmi ntdll    // 查看ntdll.dll的详细信息

0:000> !dlls -a               // 列出镜像文件PE结构的文件头
0:000> !dlls -l               // 按照顺序列出所有加载的模块
0:000> !dlls -c ntCreateFile  // 查询指定函数所在的模块
0:000> !dlls -c ntdll.dll     // 列出特定模块头信息
0:000> !dlls -s -c ntdll.dll  // 列出ntdll.dll的节区
0:000> !dlls -v -c ntdll      // 查看ntdll.dll的详细信息

0:000> ld *          // 为所有模块加载符号
0:000> ld kernel32 // 加载kernel32.dll的符号
0:000> x*!                    // 列出加载的所有符号信息
0:000> x ntdll!*              // 列出ntdll.dll中的所有符号
0:000> x ntdll!nt*            // 列出ntdll.dll模块中所有nt开头的符号
0:000> x /t /v ntdll!*        // 带数据类型、符号类型和大小信息列出符号
0:000> x kernel32!*Load*      // 列出kernel32模块中所有含Load字样的符号
```

进程与线程操作

```
| // 列出调试进程
!dml_proc             // 显示当前进程信息
.tlist -v             // 列出所有运行中的进程
~   // 列出线程
```

```
~.  // 查看当前线程
~*  // 所有线程
~0s // 查看主线程
~* k // 所有线程堆栈信息
~* r // 所有线程寄存器信息
~# // 查看导致当前事件或异常的线程
~N  // 查看序数为N的线程
~~[n]  // 查看线程ID为n的线程  n为16进制
~Ns   // 切换序数为N的线程为当前调试线程
~~[n]s  // 切换线程ID为n的线程为当前调试线程  n为16进制
~3f      把三号线程冻住
~2u       把二号线程解冻

~N n  // Suspend序数为N的线程
~N m // Resume序数为N的线程
!runaway  //显示当前进程的所有线程用户态时间信息
!runaway f  //显示当前进程的所有线程用户态、内核态、存活时间信息
!locks // 显示死锁
!cs  // 列出CriticalSection（临界段）的详细信息

0:000> .formats 1d78  // 格式化输出PID
!handle  // 查看所有句柄的ID
```

反汇编指令与内存断点

```
u   // 反汇编当前eip寄存器地址的后8条指令
ub  // 反汇编当前eip寄存器地址的前8条指令
u main.exe+0x10 L20 // 反汇编main.exe+0x10地址后20条指令
uf lyshark::add  // 反汇编lyshark类的add函数
uf /c main  // 反汇编main函数
ub 000c135d L20  // 查看地址为000c135d指令前的20条指令内容

r // 显示所有寄存器信息及发生core所在的指令
r eax, edx // 显示eax，edx寄存器信息
r eax=5, edx=6  // 对寄存器eax赋值为5，edx赋值为6

g  // Go 让程序跑起来
p    // 单步执行(F10)
p 2 // 2为步进数目
pc   // 执行到下一个函数调用处停下
pa 0x7c801b0b // 执行到7c801b0b地址处停下
t     // 停止执行

!address -summary  // 显示进程的内存统计信息
!address -f:stack  // 查看栈的内存信息
!address 0x77c000   // 查看该地址处的内存属性

bl   // 列出所有断点
bc * // 清除所有断点
be *  // 启用所有断点
bd *  // 禁用所有断点

bc 1 2 5  // 清除1号、2号、5号断点
be 1 2 5 // 启用1号、2号、5号断点
bd 1 2 5 // 禁用1号、2号、5号断点
```

```
bp main    // 在main函数开头设置一个断点
bp 0x7c801b00  // 在7c801b00地址处放置一个断点
bp main.exe+0x1032  // 在模块MyDll.dll偏移0x1032处放置一个断点
bp @$exentry  // 在进程的入口放置一个断点
bm message_*  // 匹配message_开头的函数，并在这些函数起始处都打上断点
```

堆栈操作

```
k  // 显示当前调用堆栈
kn // 带栈编号显示当前调用堆栈
kb  // 打印出前3个函数参数的当前调用堆栈
kb 5 // 只显示最上的5层调用堆栈

kv   // 在kb的基础上增加了函数调用约定、FPO等信息
kp  // 显示每一层函数调用的完整参数，包括参数类型、名字、取值
kd  // 打印堆栈的地址
kD  // 从当前esp地址处，向高地址方向搜索符号（注：函数是符号的一种）
dds 02a9ffec  // 从02a9ffec地址处，向高地址方向搜索符号（注：函数是符号的一种）
dds  // 执行完dds 02a9ffec后，可通过dds命令继续进行搜索

.frame // 显示当前栈帧
.frame n  // 显示编号为n的栈帧（n为16进制数）
.frame /r n // 显示编号n的栈帧（n为16进制数） 并显示寄存器变量
.frame /c n // 设置编号n的栈帧为当前栈帧（n为16进制数）
!uniqstack // 显示所有线程的调用堆栈
!findstack kernel32 2 // 显示包含kernel32模块（用星号标出）的所有栈的信息
!heap -s  // 显示进程堆的个数
dt _HEAP 00140000  // 选取一个堆的地址，打印该堆的内存结构
!heap -a 00140000 // 选取一个堆的地址，打印该堆的信息，比上面打印内存命令更详细直观
```

其他命令

```
dt ntdll!* // 显示ntdll里的所有类型信息
dt -rv _TEB
dt -rv _PEB
dt -v _PEB @$PEB
dt _PEB_LDR_DATA
dt _TEB ny LastErrorValue // 只查看TEB（thread's environment block）结构成员
LastErrorValue

dt _eprocess
dt _eprocess 0x510

!dh 773a0000                    显示文件PE头

*是通配符；显示所有peb打头的结构体名称；
dt ntdll!_peb*

0:000> dt -rv ntkrnlmp!*Object*   枚举ntkrnlmp中带"Object"的结构体名称；

.attach  PID  附加进程
.detach         结束会话
```

```
.dump   文件名   转存文件
.opendump       打开文件

dt -v ntdll!*            # 列出ntdll中的全部结构体，导出的函数名也会列出

dt ntdll!*file*            # 下面命令将列出ntdll导出的文件操作相关的函数名
dt _FILE_INFORMATION_CLASS        查看一个结构定义
dt ntdll!_*    列出ntdll中结构体
```

参考文献

https://www.cnblogs.com/luluping/p/15488354.html