# 堆

(安于此生译)

当进程启动时,堆管理器将自动创建一个新堆,叫做默认的进程堆.但是大多数 C/C++应用程序仍然是通过 CRT 堆(使用 new/delete 运算符以及 malloc/free 等 API)来满足它们的内存需求.还有些程序还会创建一些额外的堆(通过 HeapCreate)以将进程中不同的组件独立开来.Windows 堆管理器可以划分为两个部分:前端分配器和后端分配器.

## 前端分配器

前端分配器(Front End Allocator)是是后端分配器(Back End Allocator)的一个抽象优化层.在程序里可以选择不同类型的前端分配器来满足不同的内存需求.在 Windows 中有两种前端分配器:

① 旁视列表(Look Aside List,LAL)前端分配器

② 低碎片(Low Fragmentation,LF)前端分配器

旁视列表是一张表,其中包含了 128 项,每一项对应于一个单向链表.每个单项链表都包含了一组固定大小的空闲堆块,从 16 个字节的大小开始依次递增.每个堆块包含了 8 个字节的堆块元数据用于管理这个堆块.旁视列表索引的计算公式为:请求的块大小加上 8 个字节(元数据的大小),然后除以 8 再减去 1(旁视列表的索引从零开始).注意旁视列表的索引总是一个正数.

Windows Vista 以后的版本不再使用旁视列表前端分配器了.默认使用的是低碎片前端分配器.低碎片前端分配器非常复杂,其主要思想就是通过分配足够容纳待请求大小的最小内存块来减少堆碎片.

## 后端分配器

如果前端分配器无法满足分配请求,那么这个请求将被转发到后端分配器.

在 Windows XP 中,与前端分配器类似的是,后端分配器也包含了一张空闲列表,表中的每一项都是一个空闲链表.空闲列表的作用就是记录在指定堆中的所有空闲堆块.在空闲列表索引为 0 的项中的堆块大小将大于 1016 字节而小于虚拟内存分配的限值(0x777F0 字节).在空闲列表[0]中的堆块是按照堆块的大小来排序(升序)以获得最大效率的.通常索引为 1 的列表项不使用.索引为 x 的项中包含的空闲堆块大小为 8x.如果堆管理器无法找到某个空闲堆块的大小等于所请求的大小,那么它将使用一种块分割技术.块分割是指堆管理器首先找到一个比请求大小更大的空闲堆块,然后将其对半分割以满足分配请求.反过来,堆合并也是有可能的:当一个堆块被释放,堆管理器会检查两个相邻的堆块,如果其中一个或者两个都是空闲的话,则这些空闲的堆块将被合并为一个单独的堆块,这样可以减少堆碎片.前面已经提到了,在索引为 0 的空闲链表中包含的空闲堆块是从 1016 一直到 0x7FFF0（524272）字节的堆块.为了将查询空闲堆块的效率最大化,堆管理器将按照一定的顺序(升序)来存储空闲堆块.所有大于 0x7FFF0

的内存分配请求将被转发到虚拟分配列表(Virtual Allocation List)中.当请求一个很大的内存分配时,堆管理器将向虚拟内存管理器发出请求,并且将相应的分配信息保留在虚拟分配链表中.

在 Windows 7 中,再也没有专门的特定尺寸的空闲堆块链表了.Windows 7 使用单个空闲链表.该链表中包含了所有尺寸的空闲堆块,大小按照升序排列.与此同时,还有另外一种链表(该链表结点的类型为 ListHint),该链表中的结点指向了空闲链表中的结点,用来找到合适大小的结点来满足内存分配的请求.

## 堆段(Heap Segment)

堆管理器从何处获得内存?首先,堆管理器将通过 Windows 虚拟内存管理器来分配一大块内存.堆管理器从虚拟内存管理器请求分配的内存块也称为堆段(Heap Segment).当堆段最初被创建时,堆段中的大部分虚拟内存都是被保留的(Reserve),只有一小部分被提交(Commit).当堆管理器耗尽了已提交的空间时,堆段将进一步提交更多的内存,而堆管理器接着将对新提交的空间进行分割.当一个堆段耗尽了所有的空间后,堆管理器将创建另一个新的堆段.并且新的堆段的大小将是之前堆段大小的两倍.如果由于内存不足而无法创建这个新堆段,那么堆管理器将把这个大小减半.如果再次失败,那么将继续减半,知道堆段大小的最小阈值-这种情况下,堆管理器将返回一个错误给用户.

## 分析堆

堆链表位于 PEB(进程环境块)的偏移 0x90 处:

```
0:000> dt _PEB @$peb
ntdll!_PEB
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged    : 0x1 ''
   +0x003 BitField         : 0x8 ''
   +0x003 ImageUsesLargePages : 0y0
   +0x003 IsProtectedProcess : 0y0
   +0x003 IsLegacyProcess  : 0y0
   +0x003 IsImageDynamicallyRelocated : 0y1
   +0x003 SkipPatchingUser32Forwarders : 0y0
   +0x003 SpareBits        : 0y000
   +0x004 Mutant           : 0xffffffff Void
   +0x008 ImageBaseAddress : 0x00480000 Void
   +0x00c Ldr              : 0x77537880 _PEB_LDR_DATA
   +0x010 ProcessParameters : 0x006810a8 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : (null)
   +0x018 ProcessHeap      : 0x00680000 Void
   +0x01c FastPebLock      : 0x77537380 _RTL_CRITICAL_SECTION
   +0x020 AtlThunkSListPtr : (null)
   +0x024 IFEOKey          : (null)
   +0x028 CrossProcessFlags : 0
   +0x028 ProcessInJob     : 0y0
   +0x028 ProcessInitializing : 0y0
   +0x028 ProcessUsingVEH  : 0y0
   +0x028 ProcessUsingVCH  : 0y0
   +0x028 ProcessUsingFTH  : 0y0
   +0x028 ReservedBits0    : 0y00000000000000000000000000 (0)
   +0x02c KernelCallbackTable : 0x7726f620 Void
   +0x02c UserSharedInfoPtr : 0x7726f620 Void
   +0x030 SystemReserved   : [1] 0
   +0x034 AtlThunkSListPtr32 : 0
   +0x038 ApiSetMap        : 0x776a0000 Void
   +0x03c TlsExpansionCounter : 0
   +0x040 TlsBitmap        : 0x77537260 Void
   +0x044 TlsBitmapBits    : [2] 0x3fffff
   +0x04c ReadOnlySharedMemoryBase : 0x7f6f0000 Void
   +0x050 HotpatchInformation : (null)
   +0x054 ReadOnlyStaticServerData : 0x7f6f0590  -> (null)
   +0x058 AnsiCodePageData : 0x7ffa0000 Void
   +0x05c OemCodePageData  : 0x7ffa0000 Void
   +0x060 UnicodeCaseTableData : 0x7ffd0024 Void
   +0x064 NumberOfProcessors : 1
   +0x068 NtGlobalFlag     : 0x70
   +0x070 CriticalSectionTimeout : _LARGE_INTEGER 0xffffe86d`079b8000
   +0x078 HeapSegmentReserve : 0x100000
   +0x07c HeapSegmentCommit : 0x2000
   +0x080 HeapDeCommitTotalFreeThreshold : 0x10000
```

```
+0x084 HeapDeCommitFreeBlockThreshold : 0x1000
+0x088 NumberOfHeaps     : 4
+0x08c MaximumNumberOfHeaps : 0x10
+0x090 ProcessHeaps      : 0x77537500  -> 0x00680000 Void
+0x094 GdiSharedHandleTable : 0x00260000 Void
+0x098 ProcessStarterHelper : (null)
+0x09c GdiDCAttributeList : 0x14
+0x0a0 LoaderLock        : 0x77537340 _RTL_CRITICAL_SECTION
+0x0a4 OSMajorVersion    : 6
+0x0a8 OSMinorVersion    : 1
+0x0ac OSBuildNumber     : 0x1db0
+0x0ae OSCSDVersion      : 0
+0x0b0 OSPlatformId      : 2
+0x0b4 ImageSubsystem    : 2
+0x0b8 ImageSubsystemMajorVersion : 6
+0x0bc ImageSubsystemMinorVersion : 1
+0x0c0 ActiveProcessAffinityMask : 1
+0x0c4 GdiHandleBuffer   : [34] 0
+0x14c PostProcessInitRoutine : (null)
+0x150 TlsExpansionBitmap : 0x77537268 Void
+0x154 TlsExpansionBitmapBits : [32] 1
+0x1d4 SessionId         : 1
+0x1d8 AppCompatFlags    : _ULARGE_INTEGER 0x0
+0x1e0 AppCompatFlagsUser : _ULARGE_INTEGER 0x0
+0x1e8 pShimData         : (null)
+0x1ec AppCompatInfo     : (null)
+0x1f0 CSDVersion        : _UNICODE_STRING ""
+0x1f8 ActivationContextData : 0x00040000 _ACTIVATION_CONTEXT_DATA
+0x1fc ProcessAssemblyStorageMap : 0x006840f8 _ASSEMBLY_STORAGE_MAP
+0x200 SystemDefaultActivationContextData : 0x00030000 _ACTIVATION_CONTEXT_DATA
+0x204 SystemAssemblyStorageMap : 0x006831a8 _ASSEMBLY_STORAGE_MAP
+0x208 MinimumStackCommit : 0
+0x20c FlsCallback       : 0x00684c58 _FLS_CALLBACK_INFO
+0x210 FlsListHead       : _LIST_ENTRY [ 0x684a38 - 0x684a38 ]
+0x218 FlsBitmap         : 0x77537270 Void
+0x21c FlsBitmapBits     : [4] 7
+0x22c FlsHighIndex      : 2
+0x230 WerRegistrationData : 0x001f0000 Void
+0x234 WerShipAssertPtr  : (null)
+0x238 pContextData      : 0x00050000 Void
+0x23c pImageHeaderHash  : (null)
+0x240 TracingFlags      : 0
+0x240 HeapTracingEnabled : 0y0
+0x240 CritSecTracingEnabled : 0y0
+0x240 SpareTracingBits : 0y00000000000000000000000000000000 (0)
```

我们感兴趣的是这里:

```
+0x088 NumberOfHeaps     : 4
+0x08c MaximumNumberOfHeaps : 0x10
+0x090 ProcessHeaps      : 0x77537500  -> 0x00680000 Void
```

ProcessHeaps 是一个指向 HEAP(一个指针指向一个堆)结构指针的数组.
我们一起来看一看这个数组:

```
0:000> dd 0x77537500
77537500  00680000 00010000 00210000 014d0000
77537510  00000000 00000000 00000000 00000000
77537520  00000000 00000000 00000000 00000000
77537530  00000000 00000000 00000000 00000000
77537540  00000000 77537340 7753ab08 77537220
77537550  00000000 00000000 00000000 00000000
77537560  77537220 0068aa10 00000000 00000000
77537570  00000000 00000000 00000000 00000000
```

我们可以像这样显示第一个 HEAP 结构:

```
0:000> dt _HEAP 00680000
ntdll!_HEAP
   +0x000 Entry            : _HEAP_ENTRY
   +0x008 SegmentSignature : 0xffeeffee
   +0x00c SegmentFlags     : 0
   +0x010 SegmentListEntry : _LIST_ENTRY [ 0x6800a8 - 0x6800a8 ]
   +0x018 Heap             : 0x00680000 _HEAP
   +0x01c BaseAddress      : 0x00680000 Void
   +0x020 NumberOfPages    : 0x100
   +0x024 FirstEntry       : 0x00680588 _HEAP_ENTRY
   +0x028 LastValidEntry   : 0x00780000 _HEAP_ENTRY
   +0x02c NumberOfUnCommittedPages : 0xe2
   +0x030 NumberOfUnCommittedRanges : 1
   +0x034 SegmentAllocatorBackTraceIndex : 0
   +0x036 Reserved         : 0
   +0x038 UCRSegmentList   : _LIST_ENTRY [ 0x69dff0 - 0x69dff0 ]
   +0x040 Flags            : 0x40000062
   +0x044 ForceFlags       : 0x40000060
   +0x048 CompatibilityFlags : 0
   +0x04c EncodeFlagMask   : 0x100000
   +0x050 Encoding         : _HEAP_ENTRY
   +0x058 PointerKey       : 0x7f3a4c89
   +0x05c Interceptor      : 0
   +0x060 VirtualMemoryThreshold : 0xfe00
   +0x064 Signature        : 0xeeffeeff
   +0x068 SegmentReserve   : 0x100000
   +0x06c SegmentCommit    : 0x2000
   +0x070 DeCommitFreeBlockThreshold : 0x200
   +0x074 DeCommitTotalFreeThreshold : 0x2000
   +0x078 TotalFreeSize    : 0x1c42
   +0x07c MaximumAllocationSize : 0x7ffdefff
   +0x080 ProcessHeapsListIndex : 1
   +0x082 HeaderValidateLength : 0x138
   +0x084 HeaderValidateCopy : (null)
   +0x088 NextAvailableTagIndex : 0
   +0x08a MaximumTagIndex  : 0
   +0x08c TagEntries       : (null)
   +0x090 UCRList          : _LIST_ENTRY [ 0x69dfe8 - 0x69dfe8 ]
   +0x098 AlignRound       : 0x17
   +0x09c AlignMask        : 0xfffffff8
   +0x0a0 VirtualAllocdBlocks : _LIST_ENTRY [ 0x6800a0 - 0x6800a0 ]
   +0x0a8 SegmentList      : _LIST_ENTRY [ 0x680010 - 0x680010 ]
   +0x0b0 AllocatorBackTraceIndex : 0
   +0x0b4 NonDedicatedListLength : 0
   +0x0b8 BlocksIndex      : 0x00680150 Void
   +0x0bc UCRIndex         : 0x00680590 Void
   +0x0c0 PseudoTagEntries : (null)
   +0x0c4 FreeLists        : _LIST_ENTRY [ 0x68aea8 - 0x690d20 ]
   +0x0cc LockVariable     : 0x00680138 _HEAP_LOCK
   +0x0d0 CommitRoutine    : 0x7f3a4c89     long  +7f3a4c89
   +0x0d4 FrontEndHeap     : (null)
   +0x0d8 FrontHeapLockCount : 0
   +0x0da FrontEndHeapType : 0 ''
   +0x0dc Counters         : _HEAP_COUNTERS
   +0x130 TuningParameters : _HEAP_TUNING_PARAMETERS
```

我们可以使用 mona.py 来获取有用的信息.首先我们来获取一些常规的信息:

```
0:000> !py mona heap
Hold on...
[+] Command used:
!py mona.py heap
Peb : 0x7ffdd000, NtGlobalFlag : 0x00000070
Heaps:
------
0x00280000 (1 segment(s) : 0x00280000) * Default process heap  Encoding key: 0x22fe19cd
0x00010000 (1 segment(s) : 0x00010000)   Encoding key: 0x47b6628d
0x004c0000 (1 segment(s) : 0x004c0000)   Encoding key: 0x58936ccc
0x005a0000 (1 segment(s) : 0x005a0000)   Encoding key: 0x6651d565

Please specify a valid searchtype -t
Valid values are :
   lal
   lfh
   all
   segments
   chunks
   layout
   fea
   bea

[+] This mona.py action took 0:00:00.062000
```

我们知道该进程当前一共有 4 个堆,这里 mona 显示出了每个堆段的信息.
我们还可以使用!heap:

```
0:000> !heap -m
Index   Address  Name      Debugging options enabled
  1:   00280000
    Segment at 00280000 to 00380000 (0001e000 bytes committed)
  2:   00010000
    Segment at 00010000 to 00020000 (00001000 bytes committed)
  3:   004c0000
    Segment at 004c0000 to 004d0000 (00004000 bytes committed)
  4:   005a0000
    Segment at 005a0000 to 005e0000 (00001000 bytes committed)
```

“-m”选项用来显示堆段.

想要查看特定的堆段,比如说 0x280000,我们可以用:

```
0:000> !py mona heap -h 280000 -t segments
Hold on...
[+] Command used:
!py mona.py heap -h 280000 -t segments
Peb : 0x7ffdd000, NtGlobalFlag : 0x00000070
Heaps:
------
0x00280000 (1 segment(s) : 0x00280000) * Default process heap  Encoding key: 0x22fe19cd
0x00010000 (1 segment(s) : 0x00010000)   Encoding key: 0x47b6628d
0x004c0000 (1 segment(s) : 0x004c0000)   Encoding key: 0x58936ccc
0x005a0000 (1 segment(s) : 0x005a0000)   Encoding key: 0x6651d565


[+] Processing heap 0x00280000
Segment List for heap 0x00280000:
---------------------------------
Segment 0x00280588 - 0x00380000 (FirstEntry: 0x00280588 - LastValidEntry: 0x00380000): 0x000ffa78 bytes

[+] This mona.py action took 0:00:00.047000
```

注意:mona 首先会显示所有堆的统计信息,然后紧接着才是我们指定堆的信息.我们也可以省略掉”-h 280000”,这样就可以获取所有堆段的信息了:

```
0:000> !py mona heap -t segments
Hold on...
[+] Command used:
!py mona.py heap -t segments
Peb : 0x7ffdd000, NtGlobalFlag : 0x00000070
Heaps:
------
0x00280000 (1 segment(s) : 0x00280000) * Default process heap  Encoding key: 0x22fe19cd
0x00010000 (1 segment(s) : 0x00010000)   Encoding key: 0x47b6628d
0x004c0000 (1 segment(s) : 0x004c0000)   Encoding key: 0x58936ccc
0x005a0000 (1 segment(s) : 0x005a0000)   Encoding key: 0x6651d565


[+] Processing heap 0x00280000
Segment List for heap 0x00280000:
---------------------------------
Segment 0x00280588 - 0x00380000 (FirstEntry: 0x00280588 - LastValidEntry: 0x00380000): 0x000ffa78 bytes

[+] Processing heap 0x00010000
Segment List for heap 0x00010000:
---------------------------------
Segment 0x00010588 - 0x00020000 (FirstEntry: 0x00010588 - LastValidEntry: 0x00020000): 0x0000fa78 bytes

[+] Processing heap 0x004c0000
Segment List for heap 0x004c0000:
---------------------------------
Segment 0x004c0588 - 0x004d0000 (FirstEntry: 0x004c0588 - LastValidEntry: 0x004d0000): 0x0000fa78 bytes

[+] Processing heap 0x005a0000
Segment List for heap 0x005a0000:
---------------------------------
Segment 0x005a0588 - 0x005e0000 (FirstEntry: 0x005a0588 - LastValidEntry: 0x005e0000): 0x0003fa78 bytes

[+] This mona.py action took 0:00:00.078000
```

mona.py 还可以查看内存块(chunks)中已分配的块.想要查看堆段中的堆块可以使用:

```
0:000> !py mona heap -h 280000 -t chunks
Hold on...
[+] Command used:
!py mona.py heap -h 280000 -t chunks
Peb : 0x7ffdd000, NtGlobalFlag : 0x00000070
Heaps:
------
0x00280000 (1 segment(s) : 0x00280000) * Default process heap  Encoding key: 0x22fe19cd
0x00010000 (1 segment(s) : 0x00010000)   Encoding key: 0x47b6628d
0x004c0000 (1 segment(s) : 0x004c0000)   Encoding key: 0x58936ccc
0x005a0000 (1 segment(s) : 0x005a0000)   Encoding key: 0x6651d565

[+] Preparing output file 'heapchunks.txt'
    - (Re)setting logfile heapchunks.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.

[+] Processing heap 0x00280000
Segment List for heap 0x00280000:
---------------------------------
Segment 0x00280588 - 0x00380000 (FirstEntry: 0x00280588 - LastValidEntry: 0x00380000): 0x000ffa78 bytes
    Nr of chunks : 524
    _HEAP_ENTRY  psize   size  unused  UserPtr   UserSize
       00280588  00000  00250   00001  00280590  0000024f (591) (Fill pattern,Extra present,Busy)
       002807d8  00250  00030   00018  002807e0  00000018 (24) (Fill pattern,Extra present,Busy)
       00280808  00030  00860   0001a  00280810  00000846 (2118) (Fill pattern,Extra present,Busy)
       00281068  00860  00740   0001a  00281070  00000726 (1830) (Fill pattern,Extra present,Busy)
       002817a8  00740  00058   0001c  002817b0  0000003c (60) (Fill pattern,Extra present,Busy)
       00281800  00058  00048   00018  00281808  00000030 (48) (Fill pattern,Extra present,Busy)
       00281848  00048  00090   00018  00281850  00000078 (120) (Fill pattern,Extra present,Busy)
       002818d8  00090  00090   00018  002818e0  00000078 (120) (Fill pattern,Extra present,Busy)
       00281968  00090  00238   0001c  00281970  0000021c (540) (Fill pattern,Extra present,Busy)
       00281ba0  00238  00060   0001e  00281ba8  00000042 (66) (Fill pattern,Extra present,Busy)
       00281c00  00060  00090   00018  00281c08  00000078 (120) (Fill pattern,Extra present,Busy)
       00281c90  00090  00028   00018  00281c98  00000010 (16) (Fill pattern,Extra present,Busy)
       00281cb8  00028  00060   0001a  00281cc0  00000046 (70) (Fill pattern,Extra present,Busy)
```

...

```
       0028f790  000b0  000b0   00018  0028f798  00000098 (152) (Fill pattern,Extra present,Busy)
       0028f840  000b0  000b0   00018  0028f848  00000098 (152) (Fill pattern,Extra present,Busy)
       0028f8f0  000b0  000b0   00018  0028f8f8  00000098 (152) (Fill pattern,Extra present,Busy)
       0028f9a0  000b0  000b0   00018  0028f9a8  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fa50  000b0  000b0   00018  0028fa58  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fb00  000b0  000b0   00018  0028fb08  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fbb0  000b0  000b0   00018  0028fbb8  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fc60  000b0  000b0   00018  0028fc68  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fd10  000b0  000b0   00018  0028fd18  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fdc0  000b0  000b0   00018  0028fdc8  00000098 (152) (Fill pattern,Extra present,Busy)
       0028fe70  000b0  000b0   00018  0028fe78  00000098 (152) (Fill pattern,Extra present,Busy)
       0028ff20  000b0  000b0   00018  0028ff28  00000098 (152) (Fill pattern,Extra present,Busy)
       0028ffd0  000b0  000b0   00018  0028ffd8  00000098 (152) (Fill pattern,Extra present,Busy)
       00290080  000b0  000b0   00018  00290088  00000098 (152) (Fill pattern,Extra present,Busy)
       00290130  000b0  000b0   00018  00290138  00000098 (152) (Fill pattern,Extra present,Busy)
       002901e0  000b0  000b0   00018  002901e8  00000098 (152) (Fill pattern,Extra present,Busy)
       00290290  000b0  000b0   00018  00290298  00000098 (152) (Fill pattern,Extra present,Busy)
       00290340  000b0  000b0   00018  00290348  00000098 (152) (Fill pattern,Extra present,Busy)
       002903f0  000b0  00218   00018  002903f8  00000200 (512) (Fill pattern,Extra present,Busy)
       00290608  00218  000b0   00018  00290610  00000098 (152) (Fill pattern,Extra present,Busy)
       002906b8  000b0  000b0   00018  002906c0  00000098 (152) (Fill pattern,Extra present,Busy)
       00290768  000b0  000b0   00018  00290770  00000098 (152) (Fill pattern,Extra present,Busy)
       00290818  000b0  000b0   00018  00290820  00000098 (152) (Fill pattern,Extra present,Busy)
       002908c8  000b0  000b0   00018  002908d0  00000098 (152) (Fill pattern,Extra present,Busy)
       00290978  000b0  000b0   00018  00290980  00000098 (152) (Fill pattern,Extra present,Busy)
       00290a28  000b0  000b0   00018  00290a30  00000098 (152) (Fill pattern,Extra present,Busy)
       00290ad8  000b0  000b0   00018  00290ae0  00000098 (152) (Fill pattern,Extra present,Busy)
       00290b88  000b0  000b0   00018  00290b90  00000098 (152) (Fill pattern,Extra present,Busy)
       00290c38  000b0  000b0   00018  00290c40  00000098 (152) (Fill pattern,Extra present,Busy)
       00290ce8  000b0  0d2f8   00000  00290cf0  0000d2f8 (54008) (Fill pattern)
       0029dfe0  0d2f8  00020   00003  0029dfe8  0000001d (29) (Busy)
       0x0029dff8 - 0x00380000 (end of segment) : 0xe2008 (925704) uncommitted bytes

Heap : 0x00280000 : VirtualAllocdBlocks : 0
   Nr of chunks : 0

[+] This mona.py action took 0:00:05.912000
```

使用!heap 也可以:

```
0:000> !heap -h 280000
Index   Address  Name      Debugging options enabled
  1:    00280000
    Segment at 00280000 to 00380000 (0001e000 bytes committed)
    Flags:                40000062
    ForceFlags:           40000060
    Granularity:          8 bytes
    Segment Reserve:      00100000
    Segment Commit:       00002000
    DeCommit Block Thres: 00000200
    DeCommit Total Thres: 00002000
    Total Free Size:      00001c48
    Max. Allocation Size: 7ffdefff
    Lock Variable at:     00280138
    Next TagIndex:        0000
    Maximum TagIndex:     0000
    Tag Entries:          00000000
    PsuedoTag Entries:    00000000
    Virtual Alloc List:   002800a0
    Uncommitted ranges:   00280090
    FreeList[ 00 ] at 002800c4: 00290cf0 . 0028ae70  (14 blocks)

    Heap entries for Segment00 in Heap 00280000
         00280000: 00000 . 00588 [101] - busy (587)
         00280588: 00588 . 00250 [107] - busy (24f), tail fill
         002807d8: 00250 . 00030 [107] - busy (18), tail fill
         00280808: 00030 . 00860 [107] - busy (846), tail fill
         00281068: 00860 . 00740 [107] - busy (726), tail fill
         002817a8: 00740 . 00058 [107] - busy (3c), tail fill
         00281800: 00058 . 00048 [107] - busy (30), tail fill
         00281848: 00048 . 00090 [107] - busy (78), tail fill
         002818d8: 00090 . 00090 [107] - busy (78), tail fill
         00281968: 00090 . 00238 [107] - busy (21c), tail fill
         00281ba0: 00238 . 00060 [107] - busy (42), tail fill
```

...

```
0028fdc0: 000b0 . 000b0 [107] - busy (98), tail fill
0028fe70: 000b0 . 000b0 [107] - busy (98), tail fill
0028ff20: 000b0 . 000b0 [107] - busy (98), tail fill
0028ffd0: 000b0 . 000b0 [107] - busy (98), tail fill
00290080: 000b0 . 000b0 [107] - busy (98), tail fill
00290130: 000b0 . 000b0 [107] - busy (98), tail fill
002901e0: 000b0 . 000b0 [107] - busy (98), tail fill
00290290: 000b0 . 000b0 [107] - busy (98), tail fill
00290340: 000b0 . 000b0 [107] - busy (98), tail fill
002903f0: 000b0 . 00218 [107] - busy (200), tail fill
00290608: 00218 . 000b0 [107] - busy (98), tail fill
002906b8: 000b0 . 000b0 [107] - busy (98), tail fill
00290768: 000b0 . 000b0 [107] - busy (98), tail fill
00290818: 000b0 . 000b0 [107] - busy (98), tail fill
002908c8: 000b0 . 000b0 [107] - busy (98), tail fill
00290978: 000b0 . 000b0 [107] - busy (98), tail fill
00290a28: 000b0 . 000b0 [107] - busy (98), tail fill
00290ad8: 000b0 . 000b0 [107] - busy (98), tail fill
00290b88: 000b0 . 000b0 [107] - busy (98), tail fill
00290c38: 000b0 . 000b0 [107] - busy (98), tail fill
00290ce8: 000b0 . 0d2f8 [104] free fill
0029dfe0: 0d2f8 . 00020 [111] - busy (1d)
0029e000:       000e2000      - uncommitted bytes.
```

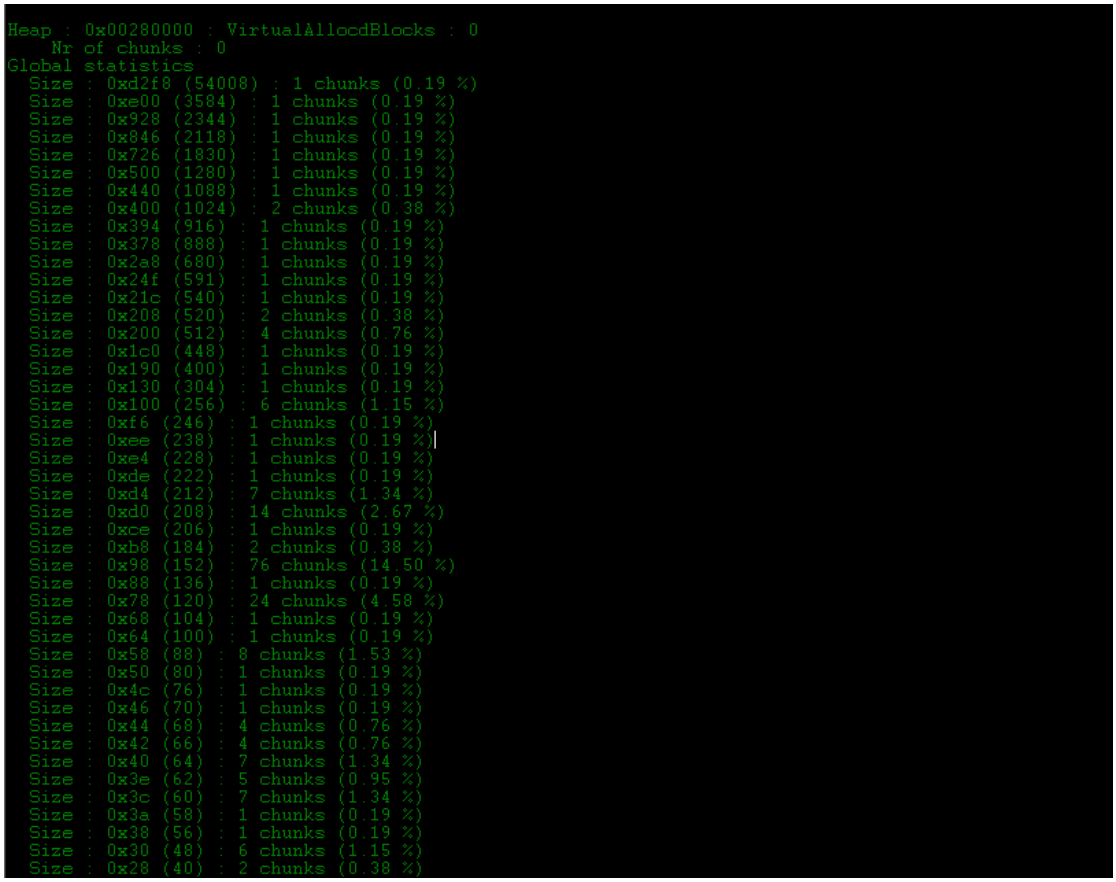
要查看统计信息的话,可以使用”-stat”选项:

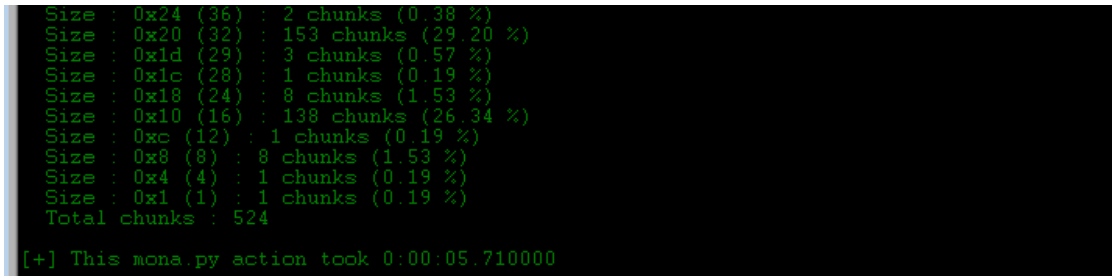
```
0:000> !py mona heap -h 280000 -t chunks -stat
Hold on...
[+] Command used:
!py mona.py heap -h 280000 -t chunks -stat
Peb : 0x7ffdd000, NtGlobalFlag : 0x00000070
Heaps:
------
0x00280000 (1 segment(s) : 0x00280000) * Default process heap  Encoding key: 0x22fe19cd
0x00010000 (1 segment(s) : 0x00010000)   Encoding key: 0x47b6628d
0x004c0000 (1 segment(s) : 0x004c0000)   Encoding key: 0x58936ccc
0x005a0000 (1 segment(s) : 0x005a0000)   Encoding key: 0x6651d565

[+] Preparing output file 'heapchunks.txt'
    - (Re)setting logfile heapchunks.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.

[+] Processing heap 0x00280000
Segment List for heap 0x00280000:
---------------------------------
Segment 0x00280588 - 0x00380000 (FirstEntry: 0x00280588 - LastValidEntry: 0x00380000): 0x000ffa78 bytes
    Nr of chunks : 524
    _HEAP_ENTRY  psize   size  unused  UserPtr   UserSize
    Segment Statistics:
    Size : 0xd2f8 (54008) : 1 chunks (0.19 %)
    Size : 0xe00 (3584) : 1 chunks (0.19 %)
    Size : 0x928 (2344) : 1 chunks (0.19 %)
    Size : 0x846 (2118) : 1 chunks (0.19 %)
    Size : 0x726 (1830) : 1 chunks (0.19 %)
    Size : 0x500 (1280) : 1 chunks (0.19 %)
    Size : 0x440 (1088) : 1 chunks (0.19 %)
    Size : 0x400 (1024) : 2 chunks (0.38 %)
    Size : 0x394 (916) : 1 chunks (0.19 %)
```

```
    Size : 0x394 (916) : 1 chunks (0.19 %)
    Size : 0x378 (888) : 1 chunks (0.19 %)
    Size : 0x2a8 (680) : 1 chunks (0.19 %)
    Size : 0x24f (591) : 1 chunks (0.19 %)
    Size : 0x21c (540) : 1 chunks (0.19 %)
    Size : 0x208 (520) : 2 chunks (0.38 %)
    Size : 0x200 (512) : 4 chunks (0.76 %)
    Size : 0x1c0 (448) : 1 chunks (0.19 %)
    Size : 0x190 (400) : 1 chunks (0.19 %)
    Size : 0x130 (304) : 1 chunks (0.19 %)
    Size : 0x100 (256) : 6 chunks (1.15 %)
    Size : 0xf6 (246) : 1 chunks (0.19 %)
    Size : 0xee (238) : 1 chunks (0.19 %)
    Size : 0xe4 (228) : 1 chunks (0.19 %)
    Size : 0xde (222) : 1 chunks (0.19 %)
    Size : 0xd4 (212) : 7 chunks (1.34 %)
    Size : 0xd0 (208) : 14 chunks (2.67 %)
    Size : 0xce (206) : 1 chunks (0.19 %)
    Size : 0xb8 (184) : 2 chunks (0.38 %)
    Size : 0x98 (152) : 76 chunks (14.50 %)
    Size : 0x88 (136) : 1 chunks (0.19 %)
    Size : 0x78 (120) : 24 chunks (4.58 %)
    Size : 0x68 (104) : 1 chunks (0.19 %)
    Size : 0x64 (100) : 1 chunks (0.19 %)
    Size : 0x58 (88) : 8 chunks (1.53 %)
    Size : 0x50 (80) : 1 chunks (0.19 %)
    Size : 0x4c (76) : 1 chunks (0.19 %)
    Size : 0x46 (70) : 1 chunks (0.19 %)
    Size : 0x44 (68) : 4 chunks (0.76 %)
    Size : 0x42 (66) : 4 chunks (0.76 %)
    Size : 0x40 (64) : 7 chunks (1.34 %)
    Size : 0x3e (62) : 5 chunks (0.95 %)
    Size : 0x3c (60) : 7 chunks (1.34 %)
    Size : 0x3a (58) : 1 chunks (0.19 %)
    Size : 0x38 (56) : 1 chunks (0.19 %)
    Size : 0x30 (48) : 6 chunks (1.15 %)
    Size : 0x28 (40) : 2 chunks (0.38 %)
    Size : 0x24 (36) : 2 chunks (0.38 %)
    Size : 0x20 (32) : 153 chunks (29.20 %)
    Size : 0x1d (29) : 3 chunks (0.57 %)
    Size : 0x1c (28) : 1 chunks (0.19 %)
    Size : 0x18 (24) : 8 chunks (1.53 %)
    Size : 0x10 (16) : 138 chunks (26.34 %)
    Size : 0xc (12) : 1 chunks (0.19 %)
    Size : 0x8 (8) : 8 chunks (1.53 %)
    Size : 0x4 (4) : 1 chunks (0.19 %)
    Size : 0x1 (1) : 1 chunks (0.19 %)
    Total chunks : 524
```




Mona.py 也可以用于在堆段中的堆块中搜索字符串,BSTRINGS 和虚表对象(vtable objects).为了查看这些信息,可以使用“-t layout”.这个函数将把数据写到 heapplayout.txt 文件中.

你还可以使用如下的附加选项:

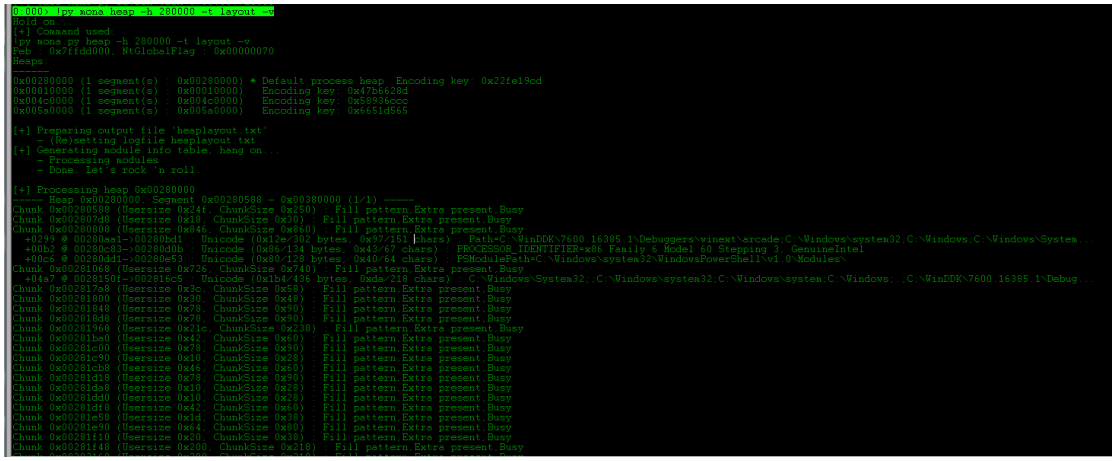
① -v:将数据输出到日志窗口中.

② -fast:跳过定位对象大小的环节.

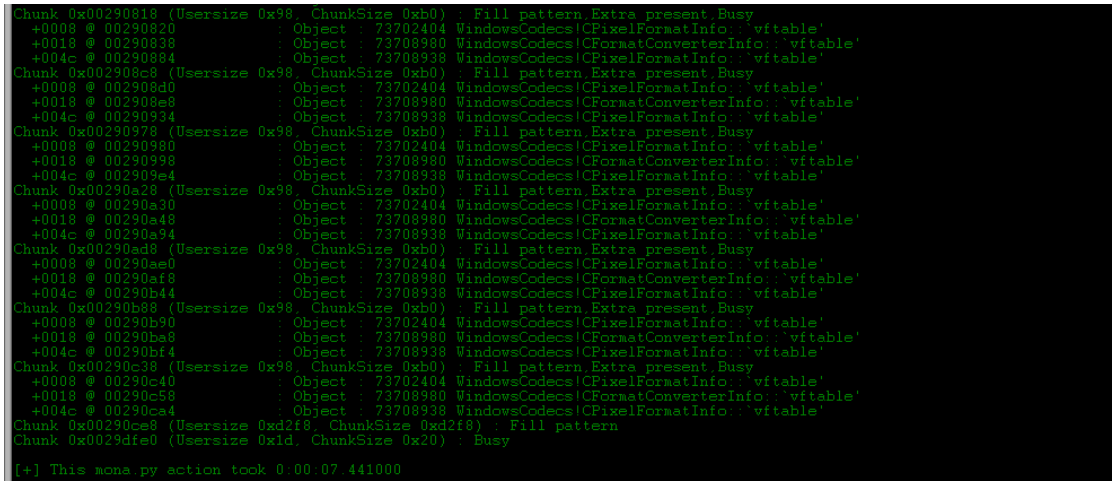
③ -size<sz>:当字符串的长度小于<sz>的时候跳过.

④ -after<val>:在搜索到包含<val>值的字符串或者虚表引用之前忽略掉堆块中的条目;搜索到之后,输出当前堆块的全部信息.

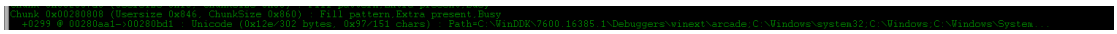
举例:




...




来看看从上面输出的信息中提取出来的两行:




第二行告诉我们:

① 这一项位于堆块起始地址偏移 0x299 字节的位置.

② 这一项从 0x280aa1 到 0x280bd1.

③ 这一项是一个 Unicode 字符串(长度为 302 个字节或者 151 个字符).

④ 这个字符串是:

“Path=C:\WinDDK\7600.16385.1\Debuggers\winext\arcade;C:\Windows\system32;C:\Windows;C:\Windows\System...”.