**Working set、Private bytes、Virtual Size三者的区别**

**Private Bytes：**进程占用内存、进程申请的内存和进程所依赖的动态库申请的内存总和，不包括进程所依赖的动态库占用的内存、mmap的内存。不一定在物理内存上，可以被交换到磁盘上，所以可以比Working Set大。由于也包括进程依赖动态库所申请的内存，所以不能判断内存泄漏是由进程导致的还是动态库导致的。

**Working Set：**进程和进程所依赖的动态库和mmap的内存占用的物理内存大小。

**Virtual Size：**进程和进程所依赖的动态库和mmap的内存一共所占用的虚拟内存，包括在物理内存上和磁盘上的总空间，所以一定比Working Set大，也比Private Bytes大。

（mmap名词解释：mmap是一种内存映射文件的方法。mmap将一个文件或者其它对象映射进内存。文件被映射到多个页上，如果文件的大小不是所有页的大小之和，最后一个页不被使用的空间将会清零。mmap在用户空间映射调用系统中作用很大。mmap 必须以PAGE\_SIZE为单位进行映射，而内存也只能以页为单位进行[映射](https://baike.baidu.com/item/%E6%98%A0%E5%B0%84)，若要映射非PAGE\_SIZE整数倍的地址范围，要先进行内存对齐，强行以PAGE\_SIZE的倍数大小进行映射。）

在Performance monitor中可以通过private bytes和Virtual bytes来衡量程序的内存使用. 在task manager中, 也有Memory Usage和VM Size两项. 但是仔细比较后会发现Memory Usage并不是对应private bytes, VM Size也不是对应Virtual Bytes.

其实, **task manager中的Memory Usage对应的是working set,** **VM Size对应的是private bytes**. 因此如果使用task manager观察内存使用, 应该注意到这个差别.

一个有趣的问题是, working set指目前程序所消耗的物理内存, private bytes值commit的内存, 那么为什么有些进程的working set比private bytes还大? 要回答这个问题, 需要仔细看看两者的定义:

所以, Working Set包含了可能被其他程序共享的内存, 而Private Bytes只包括被当前进程使用的内存。DLL是一个典型的可能被其他程序共享的资源. DLL的加载使用文件映像, 因此包含DLL的物理内存可以被同时映像到多个进程上. 所以在进程中加载DLL的内存只能算到working set上, 而不能被算到private bytes上.

在解决内存问题的时候Shared的部分一般可以不用考虑.

一个进程使用内存的时候, 它占用的内存会被分为两部分, 一部分是working set, 另一部分是private byte减去working set. 其中working set是贮存在物理内存中的, 而另一部分是paging file, 存在磁盘上.

一般来说把所有进程的working set加起来会比你机器上所拥有的物理内存要大, 这是因为有Shared的资源(比如DLL)的缘故.

在windows系统上，关于内存泄漏我们通常会听到这么两句话：  
1. 借助性能监视器，Private Bytes和Virtual Bytes至少有一个是一条斜向上曲线，大多数泄漏是这种情况；  
2. 如果Private Byte和Virtual Bytes一起上升，但是后者比前者上升得快或者比例超过3:1，说明不仅仅有内存泄漏，而且泄漏导致了内存碎片  
   
      但关键的是，这些指标到底是什么意思，这几个指标的这些变化趋势真的就能反映出进程有内存泄漏问题？如果能为什么能等等问题，其实没有多少人能够真正说得清楚，本文就试图通过这些指标入手，谈谈windows内存相关技术知识，但也不准备深入到内核层次深谈内存管理机制，只是会涉及我们平时涉及最多概念背后的故事。  
   
       首先还是要普及一下老生常谈：“在Windows系统中，任何一个进程都被赋予其自己的虚拟地址空间，该虚拟地址空间覆盖了一个相当大的范围，对于32位进程，其地址空间为232=4,294,967,296 Byte (4G)，这使得一个指针可以使用从0x00000000到0xFFFFFFFF的4GB范围之内的任何一个值。虽然每一个32位进程可使用4GB的地址空间，但并不意味着每一个进程实际拥有4GB的物理地址空间，该地址空间仅仅是一个虚拟地址空间，此虚拟地址空间只是内存地址的一个范围。进程的虚拟地址空间是为每个进程所私有的，在进程内运行的线程对内存空间的访问都被限制在调用进程之内，而不能访问属于其他进程的内存空间。这样，在不同的进程中可以使用相同地址的指针来指向属于各自调用进程的内容而不会由此引起混乱。”   
每个进程看到得虚拟地址空间有大量准确定义的区（area）构成，每个区都有专门的功能。从最低的地址看起：  
 • 程序代码和数据：代码是从同一固定地址开始，紧接着的是和C全局变量相对应的数据区。   
 • 堆：代码和数据区后紧随着的是运行时堆。作为调用malloc和free这样的C标准库函数，堆可以在运行时动态的扩展和收缩。  
 • 共享库：在地址空间的中间附近是一块用来存放像C标准库和数学库这样共享库的代码和数据的区域。  
 • 栈：位于用户虚拟地址空间顶部的是用户栈，编译器用它来实现函数调用。和堆一样每次我们从函数返回时，栈就会收缩。   
 • 内核虚拟存储器：内核是操作系统总是驻留在存储器中的部分。地址空间顶部的四分之一部分是为内核预留的。（对用户的程序来说是禁止访问的，操作系统的代码在此。内核对象也驻留在此）  
   
       最容易和上面所谓虚拟地址搞混的一个词就是“虚拟内存”，今天的windows操作系统能够使得磁盘空间看上去就像内存一样，磁盘上的文件通常称为页文件（pagefile），从应用程序的角度来看，页文件透明地增加了应用程序能够使用的内存的数量（突破物理内存大小的限制）。如果计算机拥有1G的RAM（物理内存），同时在硬盘上有一个1G的页文件，那么运行的应用程序就认为计算机总共拥有2G的RAM。  
       实际上并不是真正拥有2GB的RAM（微软不准备砸内存厂商的饭碗）。它的大致原理是将进程在物理内存中的各个部分保存到页文件中，当运行的应用程序需要时，再将页文件的各个部分重新加载到RAM中。举例：某进程试图访问的数据是在RAM中。在这种情况下，CPU将数据的虚拟地址映射到内存的物理地址中，然后执行需要的访问。线程试图访问的数据不在RAM中，而是存放在pagefile中的某个地方。这时，试图访问就称为页错误（page fault），CPU将把试图进行的访问通知操作系统。这时操作系统就寻找RAM中的一个内存空页。如果找不到空页，系统必须释放一个空页。如果一个页面尚未被修改，系统就可以释放该页面。但是，如果系统需要释放一个已经修改的页面，那么它必须首先将该页面从RAM拷贝到页交换文件中，然后系统进入该页文件，找出需要访问的数据块，并将数据加载到空闲的内存页面。然后，操作系统更新它的用于指明数据的虚拟内存地址现在已经映射到RAM中的相应的物理存储器地址中的表。这时CPU重新运行生成初始页面失效的指令，但是这次CPU能够将虚拟内存地址映射到一个物理RAM地址，并访问该数据块。  
   
**接下来分析一下进程中申请内存使用然后释放（或者不释放==!）是个什么情况：**为进程“分配内存”，这个概念可以细化：“预定一坨地址空间”，“提交一坨内存空间”，“将内存空间映射到主存”。而在程序中我们通常所访问的地址都必须是进程地址空间中被保留和提交的那段地址空间。  
     •预定地址空间Reserve：即从进程的4GB地址空间中保留一段地址空间，这个过程通过VirtualAlloc函数完成，并把分配类型参数设置为MEM\_RESERVE。这段空间的起始地址必须是系统分配粒度的整数倍，大小必须是系统页面大小的整数倍。  
     •提交内存空间Commit：即为进程已保留的地址空间映射机器的内存，这里要特别注意，所谓内存一般并不是机器的主存RAM，而只是机器的pagefile。这个过程同样又VirtualAlloc完成，只是把分配类型参数设置为MEM\_COMMIT。这段空间的起始地址和大小都必须是页面大小的整数倍。这样进程的对应被提交的区域就被映射到机器的虚拟内存上。  
     •将内存空间映射到主存：这点很重要，操作系统总是只有在进程提交的页面被访问时才将相应的页面加载到主存中，同时修改进程对应页面的地址空间映射。这时，进程的地址空间中的对应区域才和机器上的主存对应起来。  
   
解释了这些终于可以回过头来看看关于windows内存常常提及的几个指标了：  
     **Working Set：**Working Set is the current size, in bytes, of the Working Set of this process. The Working Set is the set of memory pages

touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the

Working Set of a process even if they are not in use. When free memory falls below a threshold, pages are trimmed from

 Working Sets. If they are needed they will then be soft-faulted back into the Working Set before leaving main memory. 此为官方解释，实际上该指标记录了所有映射到进程虚拟地址空间的物理内存RAM的大小（即：Task Manager中的Mem Usage），它不仅仅是用户方式分区部分的映射，而是整个进程地址空间的映射。即它同时包括内核方式分区中映射到RAM的部分。在用户方式分区部分只有在进程提交的页面被访问时才将相应的页面加载到主存中，而对于该部分的大小总是系统页面大小的整数倍。随着进程的不断运行，影响“Working Set”的因素包括：(1) 机器可用主存的大小 (2) 进程本身“Working Set”的大小范围。当机器的可用主存小于一定值（阙值）时，系统会释放一些老的最近没有被访问的页面，把这些页面通过交换文件交换到机器的虚拟内存中；当Working Set的大小大于该进程所设置的最大值时，同样会把一些老的页面交换到机器的虚拟内存中。当这些页面下次再被访问时，它们才加载到主存。  
     **Private Bytes**：Private Bytes is the current size, in bytes, of memory that this process has allocated that cannot be shared with other

processes该指标记录了进程用户方式分区地址空间中已提交的总的空间大小。无论是直接调用API申请的内存，被Heap Manager申请的内存，或者是CLR 的managed heap，都算在里面。  
     **Virtual Bytes**：Virtual Bytes is the current size, in bytes, of the virtual address space the process is using. Use of virtual address space

does not necessarily imply corresponding use of either disk or main memory pages. Virtual space is finite, and the process

 can limit its ability to load libraries.”该指标记录了当前进程申请成功的其虚拟地址空间的总的空间大小，包括DLL/EXE占用的地址和通过VirtualAlloc API Reserve（即不管有没有commit）的Memory Space数量。  
    补充一点：如两个进程都需要同一个DLL的支持，所以在进程运行过程中，这个DLL被映射到了两个进程的地址空间中，如果这个DLL的大小为4K，在两个进程中都要提交4K的虚拟地址空间来映射这个DLL。当第一个进程访问了这个DLL时，这个DLL被加载到机器主存中，这时，第二个进程也要访问该DLL，这时，系统就不会再加载一遍该DLL了，因为这个DLL已经在主存中了。当然上面所说的访问仅仅是读取的操作，如果这时候某个进程要修改DLL对应这段地址中的某个单元时，这时，系统必须为第二个进程分配另外的新页面，并把要修改位置对应的页面拷贝的这个新页面，同时，第二个进程中的这个DLL被映射到这个新页面上，这就是传说中的**写时拷贝（Copy on Write**）。  
   
      其实光是定义了、解释了这些概念，还是弄不清楚他们分别是对进程运行时哪些具体状态的写照、到底什么指标能够更准确的描述进程内存状况。  
•Private Bytes are what your app has actually allocated, but include pagefile usage;  
•Working Set is the non-paged Private Bytes plus memory-mapped files;  
•Virtual Bytes are the Working Set plus paged Private Bytes and standby list.  
       
        通过上面的描述，首先Working Set不是进程内存消耗的全部，该指标是动态的，在测量的过程中会不断变化。（变化的最小单位为4K）所以Working Set指标强调的是进程对机器主存的消耗，不是进程内存的全部信息。  
Private Bytes包含所有为进程提交的内存，包括机器主存和虚拟内存，可以认为它是进程对物理内存消耗，且该指标相对来说更加稳定。在程序产生内存泄漏时，该值一定是不断上涨的。所以一般更倾向于使用Private Bytes来定量进程的内存消耗和分析进程的内存泄漏。内存泄露时表现的现象是私有虚拟内存的递增，而不是工作集大小的递增。因为在某个点上，内存管理器会阻止一个进程继续增加物理内存大小，但它可以继续增大它的虚拟内存大小。