虚拟内存 /proc/vmstat

字段	说明
nr_free_pages	当前剩余内存页数
nr_zone_inactive_anon	zone的非活动匿名页页数

字段	说明
nr_zone_active_anon	zone的活动匿名页页数
nr_zone_inactive_file	zone的非活动文件页页数
nr_zone_active_file	zone的活动文件页页数
nr_zone_unevictable	zone的不能pageout/swapout的内存页页数
nr_zone_write_pending	zone的正在回写的内存页数
nr_page_table_pages	内存页表的页数
nr_free_cma	剩余的DMA内存
nr_inactive_anon	非活动匿名页数
nr_active_anon	活动匿名页数
nr_inactive_file	非活动文件页页数
nr_active_file	活动文件页页数
nr_unevictable	不能 pageout/swapout 的内存页数
nr_slab_reclaimable	可回收的slab内存页数
nr_slab_unreclaimable	不可回收的slab内存页数
nr_isolated_anon	被隔离的匿名页页数
nr_isolated_file	被隔离的文件页页数
nr_anon_pages	匿名页总页数
nr_mapped	用于映射的内存页数
nr_file_pages	文件页总页数
nr_dirty	脏页总页数
nr_writeback	回写的页数
nr_shmem	分配给共享内存的页数
nr_file_hugepages	大页面
nr_unreclaimable_pages	不可回收的页面
nr_dirty_threshold	脏页页数阈值
pgpgin	从启动到现在读入的内存页数
pgpgout	写入到交换分区的页数
pgfree	从启动到现在释放的页数
pgactivate	从启动到现在激活的页数
pgdeactivate	从启动到现在没激活的页数

字段	说明
pgfault	从启动到现在二级页面错误数
pgmajfault	启动到现在一级页面错误数
pgscan_kswapd	kswapd后台进程扫描的页面数
pgscan_direct	直接回收扫描的页面数
pgscan_direct_throttle	普通存储区被直接回收的页面数
pgsteal_kswapd	kswapd的回收量
pgsteal_direct	直接回收量
kswapd_low_wmark_hit_quickly	达到low水线的次数
kswapd_high_wmark_hit_quickly	达到high水线的次数
slabs_scanned	从启动到现在被扫描的切片数
kswapd_inodesteal	从启动到现在由kswapd回收用于其它目的的页面数
pageoutrun	从启动到现在通过 kswapd调用来回收的页面数

物理内存信息 /proc/meminfo

> **MemTotal**
> 物理内存大小。MemTotal并不等于所有内存条内存容量之和，是因为在系统加电之后，firmware和kernel本身需要占用一些内存，这些占用的内存不会被统计到meminfo文件当中，因此MemTotal表示的内存大小是去掉了这些内存之后剩余可供系统使用的物理内存总大小，在系统运行过程中，MemTotal的值固定不变。
>
> **MemFree**
> 当前系统的空闲内存大小，是完全没有被使用的内存
>
> **MemAvailable**
> 可用内存大小。MemFree表示的是当前系统的空闲内存大小，而MemAvailable表示的是当前系统的可用内存大小，这两个的含义是不同的。MemFree表示完全没有被使用的内存。但是实际上来说我们能够使用的内存不仅仅是现在还没有被使用的内存，还包括目前已经被使用但是可以被回收的内存，这部分内存加上MemFree才是我们实际可用的内存，cache、buffer、slab等其中都有一部分内存可以被回收，MemAvailable就是MemFree加上这部分可回收的内存之后的结果，当然因为这部分可回收的内存当前还没有被回收，因此只能够通过算法来估算出这部分内存的大小，所以MemAvailable是一个估算值，并不是真实的统计值。
>
> **Buffers**
> 直接对块设备进行读写操作使用的缓存。主要包括：直接读写块设备，文件系统元数据（比如superblock，不包括文件系统中文件的元数据）。它与Cached的区别在于，Cached表示的普通文件的缓存。
> Buffers占用的内存存在于lru list中，会被统计到Active(file)或者Inactive(file)中。
>
> **Cached**
> Cached是所有的文件缓存，Cached是Mapped的超集。Cached中不仅包含了mapped的页面，也包含了unmapped的页面。当一个文件不再和进程关联之后，在pagecache中的页面不会被马上回收，仍然存在于Cached中，还保留在lru list上，但是Mapped不再统计这部分内存。

Cached还包含tmpfs中文件，以及shared memory，因为shared memory在内核中也是基于tmpfs来实现的。

SwapCached
匿名页在必要的情况下，会被交换到Swap中，shared memory和tmpfs虽然不是匿名页，但是它们没有磁盘文件，所以也是需要交换分区的，为了方便说明，在这里我们将匿名页、shared memory和tmpfs统称为匿名页。因此SwapCached中可能包含有AnonPages和Shmem。
SwapCached可以理解为是交换区设备的page cache，只不过page cache对应的是一个个的文件，而swapcached对应的是一个个交换区设备。
并不是每一个匿名也都在swap cache中，只有以下情况中匿名页才在swap cache中：
1）匿名页即将被交换到swap分区上，这只存在于很短的一个时间段中，因为紧接着就会发生pageout将匿名页写入交换分区，并且从swap cache中删除；
2）曾经被写入到swap分区现在又被加载到内存中的页会存在与swap cache，直到页面中的内容发生变化，或者原来用过的交换分区空间被回收。
SwapCached实际的含义是：系统中有多少匿名页曾经被swap-out，现在又被swap-in并且swap-in之后页面中的内容一直没有发生变化。也就是说，如果这些页需要被重新swap-out的话，是不需要进行IO操作的。
需要注意的是，SwapCached和Cache是互斥的，二者没有交叉。当然SwapCached也是存在于lru list中的，它和AnonPages或者Shmem有交集。

Active
lru list组中active list对应的内存大小，这主要包括pagecache和用户进程的内存，不包括kernel stack和hugepages。active list中是最近被访问的内存页。
Active(anon)和Active(file)分别对应LRU_ACTIVE_ANON和LRU_ACTIVE_FILE这两个lru list，分别表示活跃的文件内存页和匿名页，它们的加和等于Active。文件页对应着进程的代码、映射的文件，匿名页对应的是如进程的堆、栈等内存。文件页在内存不足的情况下可以直接写入到磁盘上，直接进行pageout，不需要使用到交换分区swap，而匿名页在内存不足的情况下因为没有硬盘对应的文件，所以只能够写入到交换区swap中，称为swapout。

Inactive
lru list组中inactive list对应的内存大小，也是包括pagecache和用户进程使用的内存，不包括kernel stack和hugepages。Inactive list中是最近没有被访问的内存页，也是内存自动回收机制能够回收的部分。
Inactive(anon)和Inactive(file)分别对应LRU_INACTIVE_ANON和LRU_INACTIVE_FILE这两个例如list，分别表示最近一段时间没有被访问的匿名页和文件页内存，他们的加和等于Inactive。

Unevictable
Unevictable对应的是LRU_UNEVICTABLE链表中内存的大小，unevictable lru list上是不能够pageout和swapout的内存页。

Mlocked
Mlocked统计的是被mlock()系统调用锁定的内存大小，被锁定的内存因为不能够pageout/swapout，它是存在于LRU_UNEVICTABLE链表上。当然LRU_UNEVICTABLE链表上不仅包含Mlocked的内存。

Dirty
Dirty并未完全包括系统中所有的dirty pages，系统上所有的dirty pages应该还包括NFS_Unstable和Writeback，NFS_Unstable是发送给了NFS Server当时没有写入磁盘的缓存页，Writeback是正准备写磁盘的缓存。

AnonPages
AnonPages统计了匿名页。需要注意的是，shared memory和tmpfs不属于匿名页，而是属于Cached。Anonymous pages是和用户进程相关联的，一旦进程退出了，匿名页也就被释放了，不像是page cache，进程退出后仍然可以存在于缓存中。AnonPages中包含了THP使用的内存。

Mapped
Mapped是Cached的一个子集。Cache中包含了文件的缓存页，这些缓存页有一些是与正在运行的进程相关联的，如共享库、可执行文件等，有一些是当前不在使用的文件。与进程相关联的文件使用的缓存页就被统计到Mapped中。
进程所占的内存分为anonymous pages和file backed pages，所以理论上来讲：
所有进程占用的PSS之和 = Mapped + AnonPages

Shmem
Shmem统计中的内存是shared memory和tmpfs、devtmpfs之和，所有的tmpfs文件系统使用的空间都算入共享内存中。devtmpfs是/dev文件系统类型，也属于一种内存文件系统。
shared memory存在于shmget、shm_open和mmap(...MAP_ANONYMOUS|MAP_SHARED...)系统调用。
由于shared memory也是基于tmpfs实现的，所以这部分内存不算是匿名内存，虽然mmap使用了匿名内存标识符，因此shmem这部分内存被统计到了Cached或者Mapped中。但是shmem这部分内存存在于anon lru list中或者在unevictable lru list中，而不是在file lru list中，这一点需要注意。

Slab
Slab是分配块内存时使用的，详细的slab信息可以在/proc/slabinfo中看到，SReclaimable和SUnreclaim中包含了slab中可回收内存和不可回收内存，它们的加和应该等于Slab的值。

KernelStack
KernelStack是操作系统内核使用的栈空间，每一个用户线程都会被分配一个内核栈，内核栈是属于用户线程的，但是只有通过系统调用进入内核态之后才会使用到。KernelStack的内存不在LRU list中管理，也没有包含进进程的RSS和PSS中进行统计。

PageTables
PageTables用于记录虚拟地址和物理地址的对应关系，随着内存地址分配的增多，PageTables占用的内存也会增加。

NFS_Unstable
NFS_Unstable记录了发送给NFS server但是还没有写入硬盘的缓存。

Bounce
有些老设备只能够访问低端内存，比如16M以下的内存，当应用程序发出一个IO请求，DMA的目的地址却是高端内存时，内核将低端内存中分配一个临时buffer作为跳转，把位于高端内存的缓存数据复制到bounce中，这种额外的数据拷贝会降低性能，同时也会占用额外的内存。

AnonHugePages
AnonHugePages统计的是THP内存，而不是Hugepages内存。AnonHugePages占用的内存是被统计到进程的RSS和PSS中的。

CommitLimit
Commit相关内存涉及到进程申请虚拟内存溢出的问题。
当进程需要使用物理内存的时候，实际上内核给分配的仅仅是一段虚拟内存，只有当进程需要对内存进行操作的时候才会在缺页中断处理中对应分配物理内存，进程使用的物理内存是有限的，虚拟内存也是有限的，当操作系统使用了过多的虚拟内存的时候，也会差生问题，这个时候需要通过overcommit机制来判断。在/proc/sys/vm/下面有几个相关的参数：
overcommit_memory：overcommit情况发生时的处理策略，可以设置为0,1,2
0：OVERCOMMIT_GUESS 根据具体情况进行处理
1：OVERCOMMIT_ALWAYS 无论进程使用了多少虚拟内存都不进行控制，即允许overcommit出现
2：OVERCOMMIT_NEVER 不允许overcommit出现
在overcommit_memory中如果设置为2，那么系统将不会允许overcommit存在，如何判断当前是否发生了overcommit呢？就是判断当前使用内存是否超过了CommitLimit的限制。

> 当用户进程在申请内存的时候，内核会调用__vm_enough_memory函数来验证是否允许分配这段虚拟内存
>
> **Committed_AS**:当前已经申请的虚拟内存的大小。
>
> **VmallocTotal**：可用虚拟内存总大小，内核中常量
>
> **VmallocUsed**：内核常量0
>
> **VmallocChunk**：内核常量0
> 可以在/proc/vmallocinfo中看到所有的vmalloc操作。一些驱动或者模块都有可能会使用vmalloc来分配内存。
> grep vmalloc /proc/vmallocinfo | awk '{total+=$2}; END {print total}'
>
> **HardwareCorrupted**
> 当系统检测到内存的硬件故障时，会把有问题的页面删除掉，不再使用，/proc/meminfo中的HardwareCorrupted统计了删除掉的内存页的总大小。相应的代码参见 mm/memory-failure.c: memory_failure()
>
> **AnonHugePages**
> AnonHugePages统计的是透明大页的使用。它和大页不同，大页不会被统计到RSS/PSS中，而AnonHugePages则存在于RSS/PSS中，并且它完全包含在AnonPages中
> HugePages_Total、HugePages_Free、HugePages_Rsvd、HugePages_Surp
> Hugepages在/proc/meminfo中是独立统计的，既不会进入rss/pss中，也不计入lru active/inactive， 也不会被计入cache/buffer。如果进程使用hugepages，它的rss/pss也不增加。
> THP和hugepages是不同的，THP的统计值是在/proc/meminfo中的AnonHugePages，在/proc/pid/smaps中也有单个进程的统计，这个统计值和进程的rss/pss是有重叠的，如果用户进程使用了THP，那么进程的RSS/PSS也会增加，这和Hugepages是不同的。
> HugePages_Total对应内核参数vm.nr_hugepages，也可以在运行的系统之上直接修改，修改的结果会立即影响到空闲内存的大小，因为HugePages在内核上是独立管理的，只要被定义，无论是否被使用，都不再属于free memory。当用户程序申请Hugepages的时候，其实是reserve了一块内存，并没有被真正使用，此时/proc/meminfo中的HugePages_Rsvd会增加，并且HugePages_Free不会减少。只有当用户程序真正写入Hugepages的时候，才会被消耗掉，此时HugePages_Free会减少，HugePages_Rsvd也会减少。

内核使用内存

> slab + VmallocUsed + PageTables + KernelStack + HardwareCorrupted + Bounce + X
> X表示直接通过alloc_pages/__get_free_pages分配的内存，这部分内存没有在/proc/meminfo中统计。

用户使用内存

> 用户使用内存可以有几种不同的统计方式：

> 根据lru进行统计
> Active + Inactive + Unevictable + HugePages_Total * Hugepagesize
> 根据cache统计
> 当SwapCached为0的时候，用户进程使用的内存主要包括普通文件缓存Cached、块设备缓存Buffers、匿名页AnonPages和大页
> Cached + AnonPages + Buffers + HugePages_Total * Hugepagesize
> 当SwapCached不是0 的时候，SwapCached中可能包含Shmem和AnonPages，这时候SwapCached有一部分可能与AnonPages重叠。
> 根据RSS/PSS统计
> 所有进程使用PSS加和加上unmapped的部分、再加上buffer和hugepages

∑Pss + （Cached - mapped） + Buffers + (HugePages_Total * Hugepagesize)
所有进程使用的Pss可以通过将/proc/pid/smaps中的Pss加和得到。

水位控制 /proc/zoneinfo

Linux中物理内存的每个zone都有自己独立的min, low和high三个档位的watermark值，在代码中以struct zone中的_watermark[NR_WMARK]来表示。

- WMARK_MIN：最低水位，代表内存显然已经不够用了。这里要分两种情况来讨论，一种是默认的操作，此时分配器将同步等待的内存回收，在进行内存分配，也就是 direct reclaim。还有一种特殊情况，如果内存分配的请求是带了 PF_MEMALLOC(kswapd)标志位的，并且现在空余内存的大小可以满足本次内存分配的需求，那么也将是先分配，再回收。
- WMARK_LOW：低水位，代表内存已经开始吃紧，需要启动回收页内核线性kswaped去回收内存
- WMARK_HIGH：高水位，代表内存还是足够的。

spanned_pages: 代表的是这个zone中所有的页，包含空洞，计算公式是: zone_end_pfn - zone_start_pfn

present_pages： 代表的是这个zone中可用的所有物理页，计算公式是：spanned_pages-hole_pages

managed_pages： 代表的是通过buddy管理的所有可用的页，计算公式是：present_pages - reserved_pages

三者的关系是： spanned_pages > present_pages > managed_pages

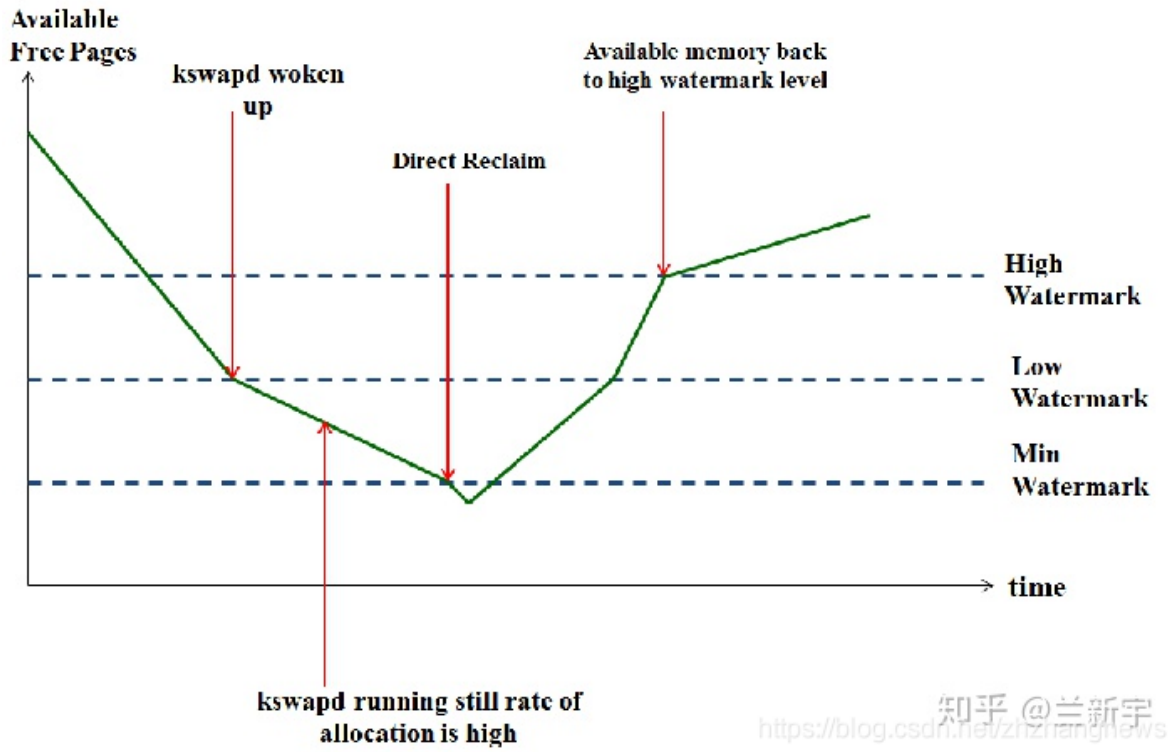
超过高水位的页数计算方法是：managed_pages减去watermark[HIGH]

lowmem_reserve: 这个zone区域保留的内存，当系统内存出现不足的时候，系统就会使用这些保留的内存来做一些操作，比如使用保留的内存进程用来可以释放更多的内存
free_area：用于维护空闲的页，其中数组的下标对应页的order数。最大order目前是11。
free_are的结构体
struct free_area {
struct list_head free_list[MIGRATE_TYPES];
unsigned long nr_free;
};

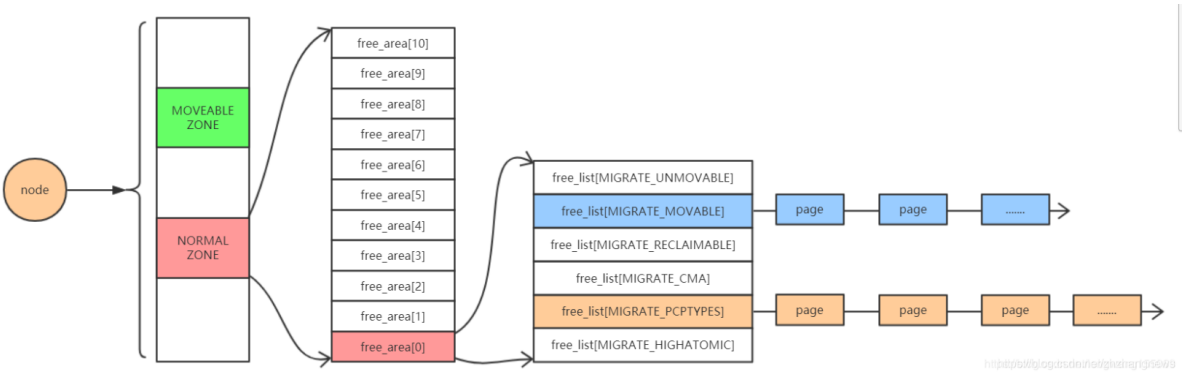
free_list：用于将各个order的free page链接在一起
nr_free: 代表这个order中还有多个空闲page

在进行内存分配的时候，如果分配器（比如 buddy allocator ）发现当前空余内存的值低于"low"但高于"min"，说明现在内存面临一定的压力，那么在此次内存分配完成后，kswapd将被唤醒，以执行内存回收操作。在这种情况下，内存分配虽然会触发内存回收，但不存在被内存回收所阻塞的问题，两者的执行关系是异步的。

> 这里所说的"空余内存"其实是一个zone总的空余内存减去其lowmem_reserve的值。对于kswapd来说，要回收多少内存才算完成任务呢？只要把空余内存的大小恢复到"high"对应的watermark值就可以了，当然，这取决于当前空余内存和"high"值之间的差距，差距越大，需要回收的内存也就越多。"low"可以被认为是一个警戒水位线，而"high"则是一个安全的水位线。



只有处于 min 和 low 这段区域才是 kswapd 的活动空间，低于了 min 会触发 direct reclaim，高于 low 有不会唤醒 kswapd



> 按照可移动性将内存页分为以下三个类型：
> UNMOVABLE：在内存中位置固定，不能随意移动。kernel分配的内存基本属于这个类型；
> RECLAIMABLE：不能移动，但可以删除回收。例如文件映射内存；
> MOVABLE：可以随意移动，用户空间的内存基本属于这个类型