通过Linux系统的内核观察_proc_pid_statm PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022__E9_80_9A_E8_BF_87Linu_c103_252890.htm 输出解释 CPU 以及CPU0。。。的每行的每个参数意思（以第一行为例）为： 参数 解释 /proc//status Size (total pages) 任务虚拟地址空间的大小 VmSize/4 Resident(pages) 应用程序正在使用的物理内存的大小 VmRSS/4 Shared(pages) 共享页数 0 Trs(pages) 程序所拥有的可执行虚拟内存的大小 VmExe/4 Lrs(pages) 被映像到任务的虚拟内存空间的库的大小 VmLib/4 Drs(pages) 程序数据段和用户态的栈的大小 （VmData VmStk ）4 dt(pages) 脏页数量 通过内核代码，我们可以更加清楚的了解其含义： 显示该信息主要是通过 proc_pid_statm 该函数来实现的。如果对proc的机制不了解，请参考《Linux设备驱动程序》。 其调用过程 ：proc_pid_statm－>statm_pmd_range->statm_pte_range。目的是从地址区间逐渐转化成具体的每个页表。阅读代码，只需了解一个大概，不用了解很多细节，要比写起来轻松许多。其中totals，pages，shared，dirty的是通过虚拟地址的页表来进行判断。 do { pte_t page = *pte. struct page *ptpage. address = PAGE_SIZE. pte . if (pte_none(page)) continue. *total. //是合法的页都计算在内。 if (!pte_present(page)) continue. ptpage = pte_page(page). if ((!VALID_PAGE(ptpage)) || PageReserved(ptpage)) continue. *pages. //只有页表中含有present标记的，计算在内。 if (pte_dirty(page)) *dirty. //页表中dirty标记，计算在内。 if (page_count(pte_page(page)) > 1) *shared. //页表的所有者超过1的，就认为共享。} while

(address vm_flags amp. VM_GROWSDOWN) //该线性区间的flags标志为向下增长。 drs = pages. /* stack */ else if (vma->vm_end > 0x60000000) //结尾线性地址大于0x60000000。 lrs = pages. /* library */ else //这块区间应该是数据区与堆。 drs = pages. vma = vma->vm_next.}pages=trs drs lrs因此说，trs drs lrs 与totals，pages，shared，dirty两组，分别从两个角度观察内存。