在Linux下，glibc 的malloc提供了下面两种动态内存管理的方法：堆内存分配和mmap的内存分配，此两种分配方法都是通过相应的Linux 系统调用来进行动态内存管理的。具体使用哪一种方式分配，根据glibc的实现，主要取决于所需分配内存的大小。一般情况中，应用层面的内存从进程堆中分配，当进程堆大小不够时，可以通过系统调用brk来改变堆的大小，但是在以下情况，一般由mmap系统调用来实现应用层面的内存分配：A、应用需要分配大于1M的内存，B、在没有连续的内存空间能满足应用所需大小的内存时。

 (1)、调用brk实现进程里堆内存分配   
    在glibc中，当进程所需要的内存较小时，该内存会从进程的堆中分配，但是堆分配出来的内存空间，系统一般不会回收，只有当进程的堆大小到达最大限额时或者没有足够连续大小的空间来为进程继续分配所需内存时，才会回收不用的堆内存。在这种方式下，glibc会为进程堆维护一些固定大小的内存池以减少内存脆片。   
  (2)、使用mmap的内存分配   
    在glibc中，一般在比较大的内存分配时使用mmap系统调用，它以页为单位来分配内存的（在Linux中，一般一页大小定义为4K），这不可避免会带来内存浪费，但是当进程调用free释放所分配的内存时，glibc会立即调用unmmap，把所分配的内存空间释放回系统。

这里我们讨论的都是虚拟内存的分配（即应用层面上的内存分配），主要由glibc来实现，它与内核中实际物理内存的分配是不同的层面，进程所分配到的虚拟内存可能没有对应的物理内存。如果所分配的虚拟内存没有对应的物理内存时，操作系统会利用缺页机制来为进程分配实际的物理内存

Android并没有采用glibc作为C库，而是采用了Google自己开发的Bionic Libc

Uclibc

<https://blog.csdn.net/maokelong95/article/details/51989081>

开源社区公开了很多现成的内存分配器（Memory Allocators，以下简称为**分配器**）：

* dlmalloc – 第一个被广泛使用的通用动态内存分配器；
* ptmalloc2 – glibc 内置分配器的原型；
* jemalloc – FreeBSD ＆ Firefox 所用分配器；
* tcmalloc – Google 贡献的分配器；
* libumem – Solaris 所用分配器；

Linux 的早期版本采用 dlmalloc 作为它的默认分配器，但是因为 ptmalloc2 提供了多线程支持，所以 后来 Linux 就转而采用 ptmalloc2 了。多线程支持可以提升分配器的性能，进而间接提升应用的性能。

在 dlmalloc 中，当两个线程同时 malloc 时，只有一个线程能够访问临界区（critical section）——这是因为所有线程**共享**用以缓存已释放内存的「空闲列表数据结构」（freelist data structure），所以使用 dlmalloc 的多线程应用会在 malloc 上耗费过多时间，从而导致整个应用性能的下降。

在 ptmalloc2 中，当两个线程同时调用 malloc 时，内存均会得以立即分配——每个线程都维护着单独的堆，各个堆被独立的空闲列表数据结构管理，因此各个线程可以并发地从空闲列表数据结构中申请内存。这种为每个线程维护独立堆与空闲列表数据结构的行为就