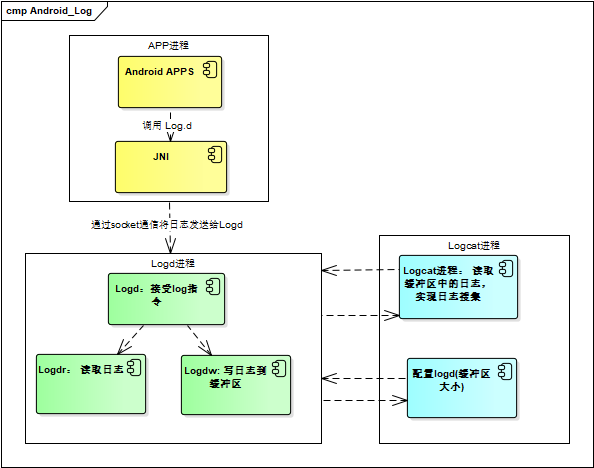
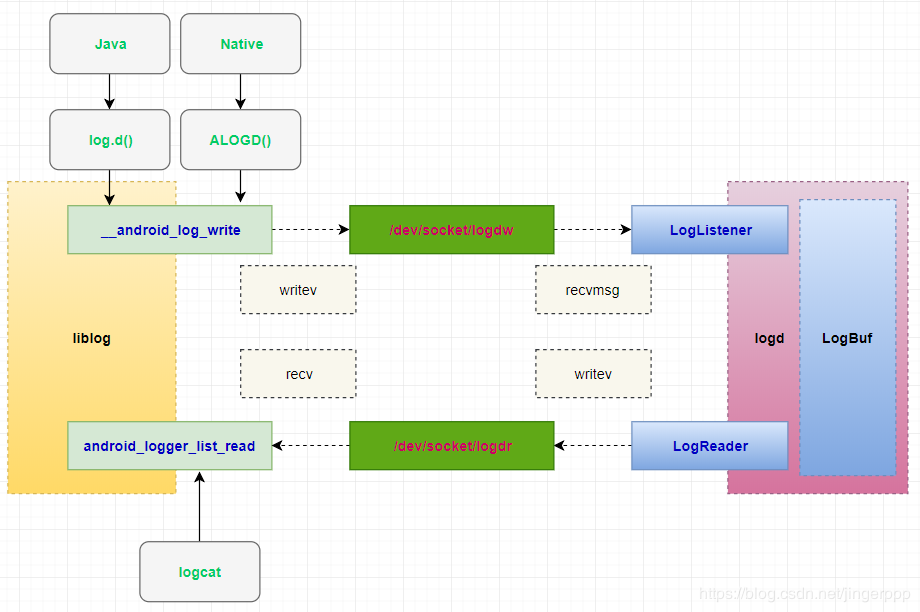
Android 日志架构



具体的流程可以看下面的这张图：



这里涉及到三个进程：

APP进程： 调用Log的接口打日志，最终通过soctket通信发送给Logd进程

Logd进程：有一个缓冲区用于存储日志（环形缓冲区，当满时会冲掉旧的日志）

Logcat进程：可以在adb shell中创建（可以创建多个），查看缓冲区中日志的进程（Android Studio的logcat也是一个Logcat进程）。

环形缓冲区是一个逻辑上的循环队列，写者可以往里面写东西，而一旦有内容会通知等待队列里的读者们来读取内容，否则等待队列列处于阻塞状态。 这就意味Logcat进程读取也

是阻塞形式的。

基于缓存的日志方案

普通应用

该方案会在系统的Log类上封装一层，应用调用封装层的日志接口。封装层将业务log缓存到一个StringBuffer中，当量达到一定阈值后，再刷到文件中。到达一定条件上传到服务端。

系统应用

在应用中开启Logcat进程，不断的读取Logd进程中的日志到缓存中，当日质量达到一定的阈值后再刷如文件。这种方案可以搜集到所有应用的日志。

缺点：

1.写文件 + 加密 会出现cpu峰值

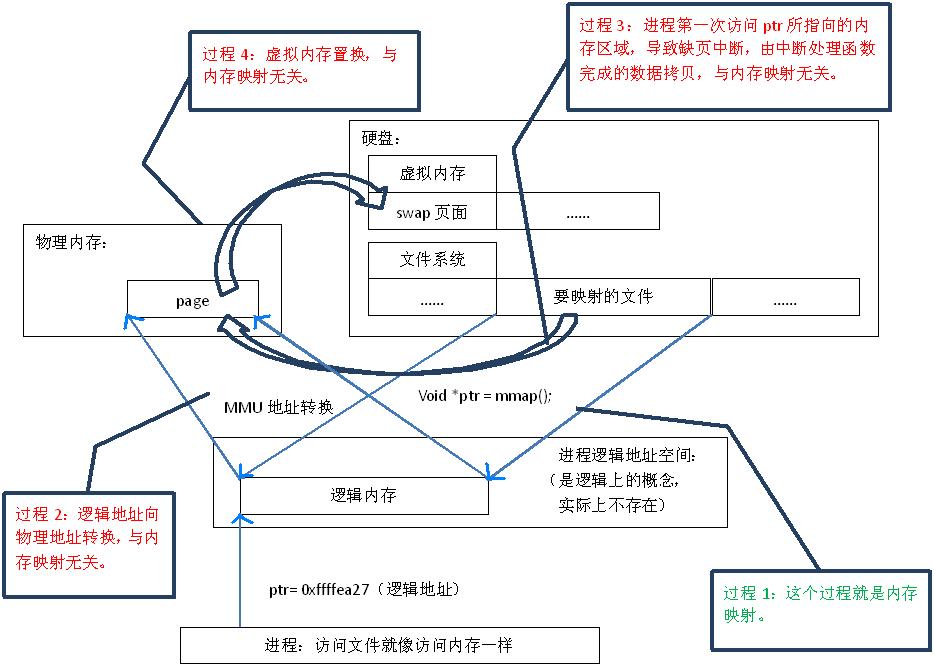
2.崩溃或者进程意外退出 缓存中的日志可能无法刷到文件中，关键崩溃信息可能丢失。

3.对于智能设备中专门做一个日志进程去搜集所有应用的日志，这种方案其实是简单可行的。

基于内存映射mmp的方案

内存映射，简而言之就是将用户空间的一段内存区域映射到内核空间，映射成功后，用户对这段内存区域的修改可以直接反映到内核空间，同样，内核空间对这段区域的修改也直接反映用户空间。那么对于内核空间<---->用户空间两者之间需要大量数据传输等操作的话效率是非常高的。

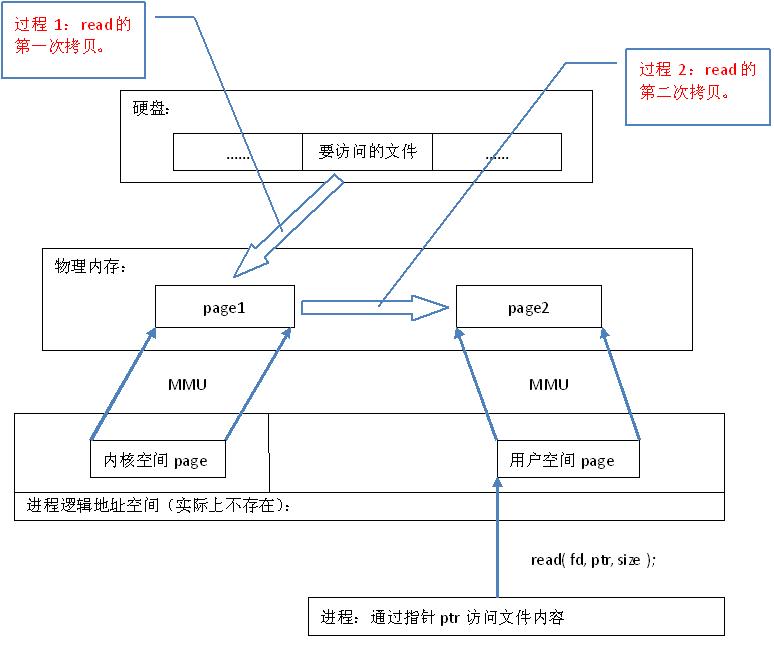
以下是一个把普遍文件映射到用户空间的内存区域的示意图。



通过mmp读取磁盘文件：

如果在拷贝数据时，发现物理内存不够用，则会通过虚拟内存机制（swap）将暂时不用的物理页面交换到硬盘上，上图步骤4所示。这个过程也与内存映射无关。

通过系统read/write文件原理：



它首先将文件内容从硬盘拷贝到内核空间的一个缓冲区，如图过程1，然后再将这些数据拷贝到用户空间，如图过程2，在这个过程中，实际上完成了 两次数据拷贝 ；

而mmap()也是系统调用，mmap()中没有进行数据拷贝，真正的数据拷贝是在缺页中断处理时进行的，由于mmap()将文件直接映射到用户空间，所以中断处理函数根据这个映射关系，直接将文件从硬盘拷贝到用户空间，只进行了 一次数据拷贝 。因此，内存映射的效率要比read/write效率高。

mmp作为日志方案优势

1.读写文件比普通文件操作效率更高

2.不会丢日志（进程退出时能刷日志到映射的地址中） 会写日志的时机：

内存不足

进程退出

调用 msync 或者 munmap

不设置 MAP\_NOSYNC 情况下 30s-60s(仅限FreeBSD)

3. 对于CPU峰值问题，参考微信XLOG解决方案如下： 追加每行日志时，先压缩后加密（避免了对整个文件的压缩/加密）