YMMLogLib 运满满日志系统

`YMMLogLib`

1、现状

现阶段，我们运满满iOS客户端没有任何一套完整的日志记录系统，当用户反馈自己的app出了问题（不只是crash的问题），联系到开发同学，开发同学只能根据用户的描述，尝试复现问题，如果恰巧问题可以复现，那就去解决；如果实在复现不了，就先放在一旁观察，佛系debug。有同学可能会问，为何必须要复现才能解决，你看到了问题现状，去查下代码不就知道问题所在了么？实际情况是这样的，在一个较大的系统里，导致同一个问题的原因非常多，而且可能千差万别，如果不能找到正确的路径复现，那对于开发同学来讲，修复bug是非常困难的一件事情，当然也不是说不可能，基本上就和200米外盲狙吃鸡胜利的概率差不多大吧。

2、技术实现

我们需要的是寻求一个高性能、高可用的日志记录方案，首先可以保证用户的日志可以高速录入文件；而且即使用户遭遇了严苛的crash，也能保证不丢日志。
调研了业界主流的较为可行的日志系统记录方案，主要的日志系统有：

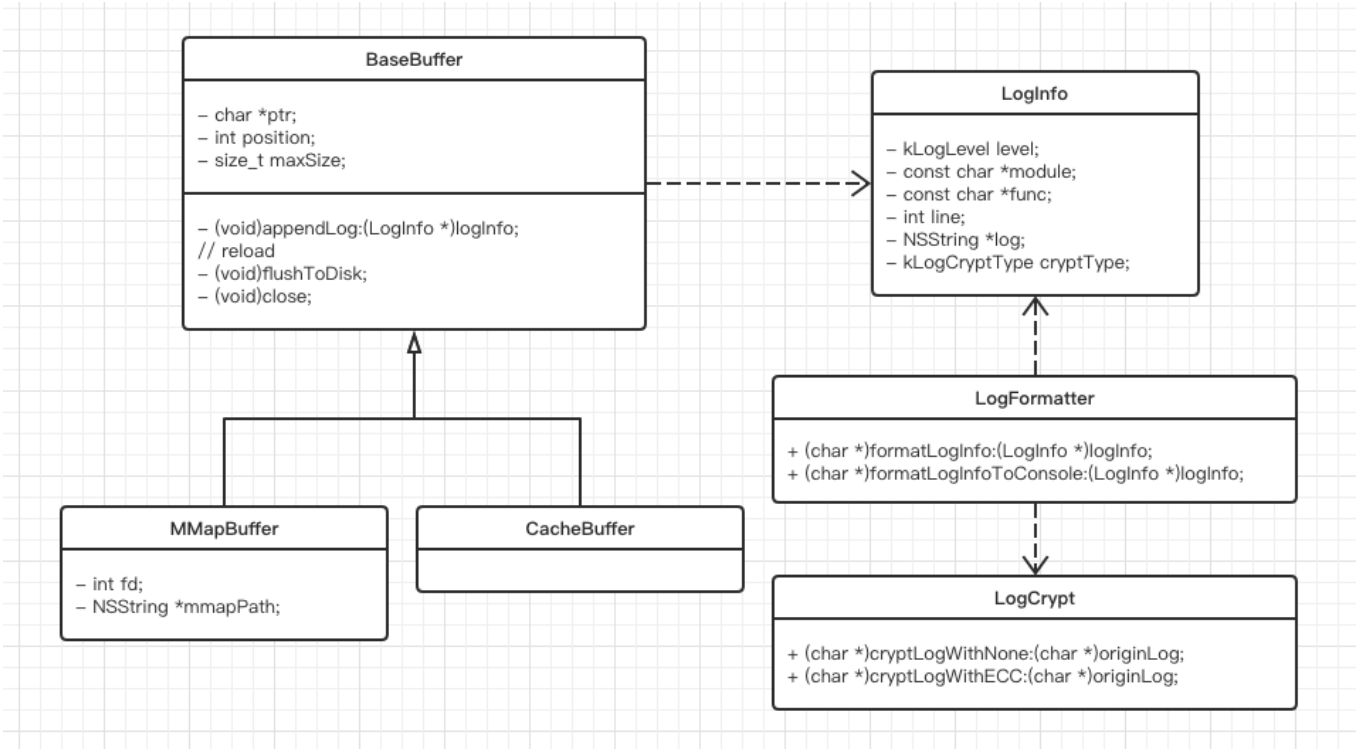
- **实时写文件** ：优点是不会丢日志，但是效率低下
- **内存缓存写文件** ：优点是效率很高，缺点是在异常情况下丢日志
- **内存映射写文件** ：效率较高，不会丢日志，但是编程要求较高

经过方案对比，为了满足高性能高可用与内存开销的权衡，最终采用通过内存映射的方式进行日志记录。在业界通过这种方式进行日志记录已经有较为成熟的方案，如微信的开源组件mars，然而由于mars组件不只是日志系统，将mars编译后，framework占到了15M，可想而知，如果直接引入过来，会导致安装包过大，可惜了之前做的APP瘦身工作，因此决定自己写一套轻量级的日志记录系统。

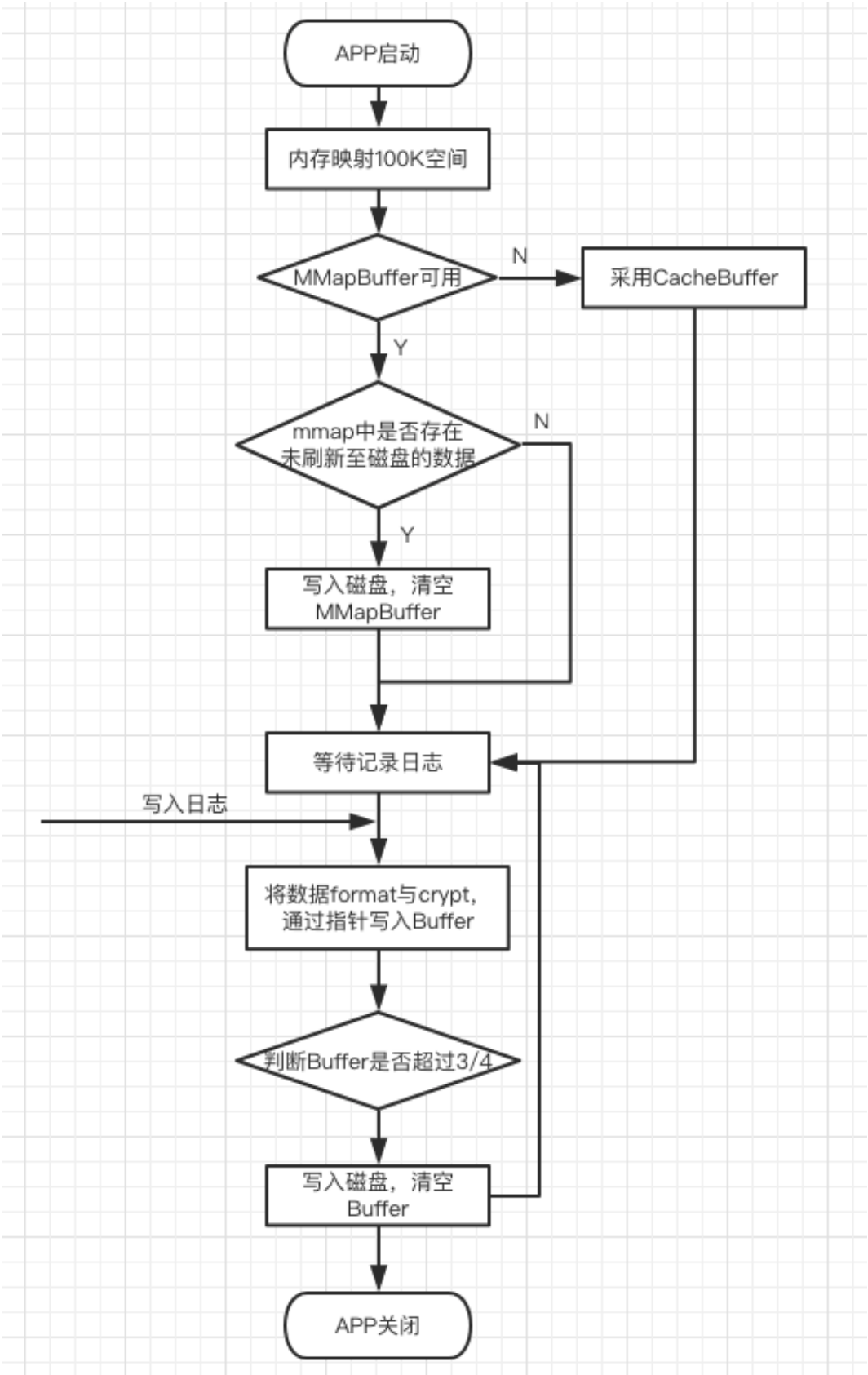
内存映射（MemoryMapped， mmap）是将一个文件或者其它对象映射到进程的地址空间，实现文件磁盘地址和进程虚拟地址空间中一段虚拟地址的一一对应关系。实现这样的映射关系后，进程就可以采用指针的方式读写操作这一段内存，而系统会自动回写至对应的文件磁

盘上，即完成了对文件的操作而不必再调用read/write等系统文件函数。常规文件操作需要从磁盘到页缓存再到用户主存的两次数据拷贝；而mmap操控文件，只需要经历从磁盘到用户主存的一次数据拷贝过程。也就是，mmap是实现了用户空间和内核空间的数据直接交互，省去了空间不同数据不通的繁琐过程，因此mmap效率更高。

整体方案的类图如图所示：



流程图如图所示：



- 系统启动后，会先去映射100k大小的连续内存空间至磁盘上，如果没有映射成功，即mmap不可用时，采用备用方案2，降级成内存缓存CacheBuffer写文件的方式去记录日

志。如果映射成功了，判断映射后的MMapBuffer里是否存在还未刷入系统磁盘的上次遗留的缓存日志，如果有，则写入磁盘并清空缓存信息。

- 有新日志需要写入时，系统会先格式化日志内容，增添日志级别、时间、方法名称、当前代码行数等信息；其次对日志进行加密；最后写入高速缓冲区MMapBuffer。
- 当高速缓冲区内容写到一定阈值时（Buffer已写入的容量 > 总容量*3/4），将缓存写入磁盘空间，清空缓存信息。

使用mmap映射一块固定长度的文件，这样保证每条log第一时间都被写入磁盘，由于每次将log写入目标文件时都会清空高速缓冲区，所以高速缓冲区的内容可以认为没有被写入文件，每次启动时可以检查缓冲区，若有数据则将它先写入目标文件，达到不丢日志的效果。同时，由于操作文件节省了页缓存的步骤，因此在写入性能上，和直接操作内存相差无几。

缓存回写主要有四个时机：

- 缓存空间超过了预设的3/4
- 收到系统内存警告时
- 遇到fatal错误日志时
- 退后台时

最终日志加密前的格式为：
[I] 2018-07-11 +8.0 16:25:12.397 [YMMTransactionModule] [-[ViewController mmapWriteFile:] 72] 这是打印的日志内容
[日志级别] 时间 [日志模块（选填）] [方法名] 行数 日志内容

3、性能对比

为了对比本文提出的轻量级日志记录方案的性能，我们同时选取微信开源组件mars、实时写文件这两种方案进行对比。每一条日志我们选取200字节，通过卡死主线程的方式反复记录1千-5万条日志，相当于共记录了200k-10M空间大小的日志量，每次实验进行5次并取平均值作为最终的结果。性能对比如下图所示。

方案/组数 (时间单位：毫秒)	**1k**	**2k**	**5k**	**2w**	**5w**
实时写文件	166	335	810	3343	8445
微信mars组件	53	97	327	996	2403

方案/组数 (时间单位：毫秒)	1k	2k	5k	2w	5w
YMMLogLib	49	113	276	1087	2476

从表中可以看出，实时写文件的性能最差，微信mars组件与我们的YMMLogLib组件在记录日志的性能上相差无几，均好于实时写文件方案70%以上。

4、踩过的坑

- **效率**
  由于日志记录组件有可能会需要在短时间内同时记录成百上千条日志，因此性能上需要做到足够好，尽量选择效率高的方法。所以很多方法尽量用底层的C来实现，而不是用OC来实现，比如字符串拼接，一律使用string.h头文件中定义的方法，而不是用[NSString stringWithFormat:]。
- **strlen**
  在写代码过程中，遇到了一个很有意思的问题，花了很长时间去定位与解决，值得拿出来和大家一起分享，并在后期注意合理的避免。这个问题表象原因看起来是加密后，无法用私钥解密，然而就开始查问题，查了很多问题后发现，是因为日志加密后，里面会出现各种各样的'\0'，这就导致，在利用strlen来计算字符串长度时，会提前结束，直接导致从缓冲区刷新到文件中的字段丢失，那自然就不能成功解密。解决方法是，把所有的strlen全部替换掉，将预先计算好的长度写在char字符串约定好的字节，这样去获取长度时，直接把这部分长度拿出来，前后加上头和尾的长度，就是需要录入到文件里的长度了。
- **NSString与char*计算长度结果不同**
  小辉反馈一例日志被截断的情况，很是诡异，复盘后发现对于中文的情况，通过NSString的length方法获取的长度，与转义成UTF8String通过strlen获取的长度不同导致的。对于一个中文汉字，NSString的长度是1，而转义成char*后的长度就是3了。因此在计算总长度时，可能会出现由于汉字的出现导致长度计算变少，最终导致log被提前截断的情况发生。

```
(lldb) po @"我".length
1

(lldb) po strlen([@"我" UTF8String])
0x0000000000000003

(lldb) po strlen("我")
0x0000000000000003
```

5、接入 & 使用方法

- 将YMMLogLib clone到本地。
- 在main函数里初始化日志模块：

```
//日志记录地址
NSString* logPath = [[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0] stringByAppendingString:@"/log/"];
//设置不要同步
const char* attrName = "com.apple.MobileBackup";
u_int8_t attrValue = 1;
setxattr([logPath UTF8String], attrName, &attrValue, sizeof(attrValue), 0, 0);
#if DEBUG
//对于debug模式，开启console控制台打印
[Logger openConsoleLog:YES];
#else
//对于其他模式，关闭console控制台打印
[Logger openConsoleLog:NO];
#endif
//初始化
[Logger appendOpenFile:logPath];
```

日志目前会按照日期命名，自动清理7天前的日志。
- 在AppDelegate里的applicationWillTerminate中结束日志系统：

```
- (void)applicationWillTerminate:(UIApplication *)application {
    [Logger closeBuffer];
}
```

- 在需要用到日志的地方，引入头文件，即可。建议在pch文件中直接包含此头文件(如果是pod管理，需要在主工程中的每一个pod中的pch文件中引入)。

```
#import <YMMLogLib/yLog.h>
... ...
MMInfo(@"这是测试日志");
```

目前提供了5个日志级别：MMDebug、MMInfo、MMWarning、MMError、MMFatal，日志的严重程度逐级加深。对于MMDebug，只在Debug与ADHOC环境才会记录到日志文件里去。
还提供了模块化日志打印的方法，方便模块中问题的定位与查找。如：

```
MMModuleInfo("TransacionModule", @"这是测试日志");
```

为了方便起见，可以自己写一个宏，包起来，如：

```
#define TransInfo(...)  MMModuleInfo("TransacionModule", __VA_ARGS__)

TransInfo(@"这是测试日志");
```

这样每次在写日志时，只要写TransInfo，即可把所有的日志都带
上“TransacionModule”模块标签。

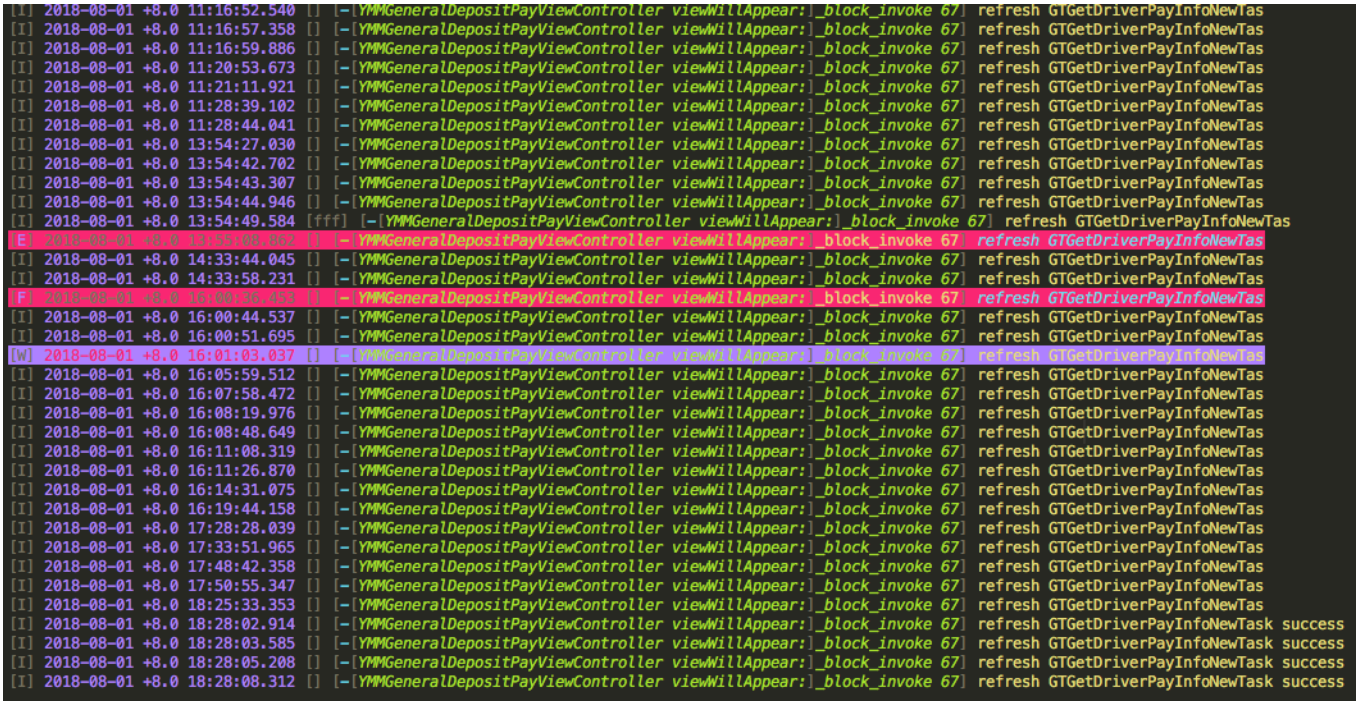
6、上传 & 解密 & 查看日志

上传
司机6.3.1货主5.3.1与之后版本的app的上传入口在“我的-更多-联系客服（长按2秒）” 里，入口比较隐蔽，普通用户不容易发现，操作门槛较高，不容易造成误触，而且也比较容易让客服和用户去沟通、引导上传日志。
解密
在拉到用户日志后，需要对用户日志进行解密，解密是通过python脚本进行的，脚本文件放置在工程目录crypt下decode_crypt_log_file.py文件。拿到加密文件后，执行：

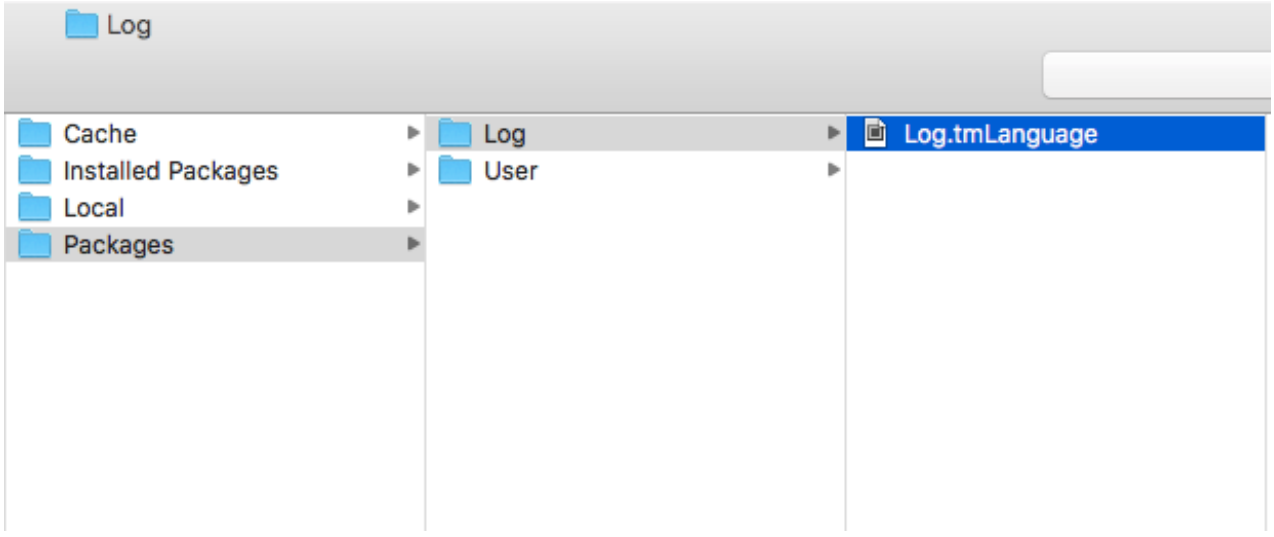
```
python decode_crypt_log_file.py （加密文件）.ylog （解密文件）.log
```

查看日志

然后用任何一个文件查看工具打开即可。这里特别推荐sublimeText工具，因为我写了一个查看日志高亮的染色脚本，可以更方便的分模块去定位每一条日志信息，对于Error、Fatal和Warning信息可以便捷查看，如下图所示。



染色脚本放在工程根文件夹下，“Log.tmLanguage”。首先打开sublimeText的Preferences选择Browse packages，然后新建Log文件夹，把这个文件放进去，重启sublimeText即可，这样sublimeText会对ylog和log后缀名的文件做捕获。文件位置如图所示：



目前日志系统还处于不断完善中，有任何问题和想法欢迎联系交易平台周暄承。谢谢~