1：在使用PyObject的时候，由于对其不熟悉，导致失败过很多次，在多次调试以及官方文档的帮助下，我注意到了PyObject的ob\_refcnt和ob\_reftype这两个属性，于是在CPython类中加强了对PyObject是否有效的检测，成功解决了问题

2：调Python的aes加密时，其输入变量应该是char\*，但是我的定义是struct，所以这里碰见了一些问题我的思路分别如下

A：将struct强转为char\*，理论上来说是没有错的，不经过加密还是可以相互转换的，但是进过加密后导致无法得出正确的明文，这里我百思不得其解，我的猜想是内存对其造成的，但是理由不够充分。

B：调用的boost：：serialization进行操作，通过将其序列化后可以得到一个字符串，将这个字符串传输给python模块进行加密，解密反着来即可。我的项目中就是采用这种方案，实现简便。

C：google protocolbuf 据说这种方案是比较理想的序列化方案，但是由于需要安装和对struct的proto文件书写，我放弃了

3：在开发中遇到的小问题抽象类的必须将定义放置在其它调用类前，不能前向声明

4：在开发线程池模块时，\_\_move\_idle\_list函数如何判断，最后我的方案是在CWorkThread中采用一个状态为来判断，如果线程完成功能后，将m\_state标志位置为IDLE，否则BUSY，在CThreadPool运行函数\_\_invoke\_move\_idle\_list完成对线程的回收操作，这就是将线程池中工作线程vector设为CWorkThread\*的好处，但是坏处是每次run的时候需要遍历busy工作vector，但是好在busy 工作vector不大

5：解决线程ID的获取，主要是用于在日志文件中记录

A：#define gettidv1(1) syscall(\_\_NR\_gettid)

B：#define gettidv2(1) syscall(SYS\_gettid)

为什么不实用thread\_self的原因，thread\_self会有重复的，在线程结束后线程ID会复用，而且ID不是很human

6：每当套接字可读时，在我的模式下会从线程池中选出一个空闲线程对sockfd进行操作，现在要避免这种操作，我的想法是在CThreadPool中建立一个链接表，链接thread和当前操作的sockfd

7：由于设计的原因导致在产生具体的事件处理对象之前必须知道event的m\_type类型，所以我的第一想法是更具ANTP的报文格式现将所需要的内容读取出来，但是这样的做法貌似没有成功，第二次recv的时候，对出现各种问题，出现问题的原因是在recv函数时，将内核缓冲的去的部分数据搬运到用户接受的buff的时候，会严格要求字节对齐，因为没recv一次内核缓冲区的数据就会少一点，在用户buff中就必须要在适当的位置再次读取，否则内容会出现问题，后面看到了MSG\_PEEK这个标志，就采取了这种偷窥的方法先从内核缓冲区中偷窥（不会移除出内核缓冲区）想要取得的数据