三种并发方案

并发方案是有最优解的，不用纠结到底该怎么用

在python的世界里永远都存在最优解去解决各种问题

一、进程

进程 process 对应的模块是multiprocessing

每一个进程都有独立内存空间 内存之间是相互隔离 但不同进程之间没有办法直接互通信息

如果要实现多进程之间的通信，需要RPC通讯 Remote Procedure Call 通过网络端口传输信息

在python中 每一个进程都一个自己独立的全局锁

multiprocessing是可以利用多核的，例如如果有8核，如果python启动8个进程，那么可以将8个核同时全部用起来

所以，multiprocessing既可以用来跑CUP密集型任务，也可以用来跑IO密集型任务

由于每个进程之间是完全隔离的，其中某个进程崩溃对其他进程导致的影响非常有限

二、比进程轻量一点儿的是线程

Thread 对应的模块是threading

1.线程的内存空间是共享的

目前绝大多数的交易程序的架构如下

例如：CTP的接口API中的某些线程负责和服务器的通讯，不同线程之间的数据是可以直接访问的

正因为不同的线程都可以访问同一个数据，所以在所有的编程语言里都需要对这种线程并行的情况进行加锁，否则将导致逻辑、数据混乱

但是因为全局锁GIL的存在，在python中的多线程，只能利用到单核，即同一时间，同一个进程内的多个线程，只能有一个线程在CPU上运行着，线程必须先拿到全局锁之后才能开始运行，一般情况下是运行python虚拟机上的500个指令，运行完毕之后，这个线程会释放全局锁，然后由操作系统来随机调度其他的线程来拿到这把全局锁，谁先抢到，谁开始运行

所以在python中的CPU密集型任务，开的线程越多就越慢，在python中开10个线程去跑10个策略是没有任何意义的，这样做只会使得整个进程运行更慢

什么时候可以用线程呢？主要针对IO密集型任务，比如从数据库读取数据，例如从rqdata下载数据，或者开发网站，收到用户访问数据的请求

对量化交易来说，多线程没有什么意义

只有线程独立拿到的threadlocal，即线程自己的数据是隔离的

其他情况下如果想要访问全局数据，这个不是隔离的

这种情况下崩溃，影响中等，主要取决于线程所影响的数据，如果所影响的数据是全局数据，那么产生的影响会比较大，如果只是threadlocal数据崩溃，那么只对一个线程有影响

数据的共享性和安全性难以兼顾，共享性强，安全性必然更差，隔离程度高，安全性必然更高

三、协程 coroutine

Coroutine 对应的模块是 gevent、asyncio

协程内存空间是完全共享的

协程就是在线程里运行着的不同的函数而已，每一个协程就是一堆保存了上下文数据的函数，所以既然是在单线程里面，锁都不用加，因为同一时间只有一个协程在跑

所以其实协程也只能利用CPU的单核，所以也只能针对IO密集型任务才有用，对于计算密集型没有意义

而且协程还涉及到用户态的上下文切换，尽管这种切换比线程还是要更快一些，但也是切换，只要有切换就会有额外的开销，只要有额外的开销，对CPU密集型任务就没有意义，它并没有加快量化策略的运行速度，反而会拖慢速度

协程崩溃的影响是最重的，这就使得协程在量化交易中没有应用的空间

前后大概花了一年多的时间才确定，进程的分布式方案才是可行的，因为这个方案能解决GIL锁的问题，多线程也好，协程也好，对量化交易来说是没有意义的

四、总结

进程：

可以做分布式扩展：作为大机构，如果策略比较多了，就可以采用分布式扩展来跑多个交易策略

线程：

逻辑流的拆分，例如CTP收到行情数据发到上层应用去处理

适合IO任务，例如启动回测，或者现在米筐数据等

如果不启动线程，主界面就会卡死，单独启动一个线程，免得界面卡死

协程：

在量化交易中几乎没有什么作用

但是异步事件回调，例如event\_engine就是最大化利用了协程，这样可以快速响应收到的数据