在多线程编程下，常常出现A线程要等待B线程条件完成后再继续进行，这里等待方式有两种：

1.使用锁+轮询

使用这种方法可以很简单的实现，但是会有一定的性能消耗，其还有一个点要好好把握，就是一次轮询没有结果后相隔多久进行下一次的轮询，**间隔时间太短，消耗的CPU资源较多，间隔时间太长，不能很及时的响应请求。**

所以这种方法不是推荐。

2.使用条件变量的线程同步（推荐）

采用阻塞和消息方式可以极大程度上减少资源的浪费以及增加实时性

pthread_cond_wait

多线程的条件变量

条件变量是利用线程间共享的 *全局变量* 进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"（给出条件成立信号）。为了防止竞争， *条件变量* 的使用总是和一个 *互斥锁* 结合在一起。

pthread_cond_timedwait()函数阻塞住调用该函数的线程，等待由cond指定的条件被触发（pthread_cond_broadcast() or pthread_cond_signal()）。

当pthread_cond_timedwait()被调用时，调用线程必须已经锁住了mutex。函数pthread_cond_timedwait()会对mutex进行【解锁和执行对条件的等待】（原子操作）。这里的原子意味着：解锁和执行条件的等待是原则的，一体的。（In this case, atomically means with respect to the mutex andthe condition variable and other access by threads to those objectsthrough the pthread condition variable interfaces.）

如果等待条件满足或超时，或线程被取消，调用线程需要在线程继续执行前先自动锁住mutex，如果没有锁住mutex，产生EPERM错误。即，该函数返回时，mutex已经被调用线程锁住。

等待的时间通过abstime参数（绝对系统时间，过了该时刻就超时）指定，超时则返回ETIMEDOUT错误码。开始等待后，等待时间不受系统时钟改变的影

响。

尽管时间通过秒和纳秒指定，系统时间是毫秒粒度的。需要根据调度和优先级原因，设置的时间长度应该比预想的时间要多或者少点。可以通过使用系统时钟接口gettimeofday()获得timeval结构体。