知识点的整---线程池的实现:
线程池--线程的池子(很多线程)
场景:淘宝--双十一---很多剁手者，都会买东西，向淘宝发出购物请求---淘宝服务器就要针对这些请求进行处理服务器上肯定是多执行流并行/并发处理

若是请求来了就创建执行流进行处理，存在两个问题:
1.若峰值压力下，会创建线程过多，有可能导致资源耗尽
2.任务处理过程中，线程的创建与销毁成本过高
在服务器启动的时候，就创建大量线程+创建一个线程安全的队列作为线程池
任务来了，就将任务抛入线程池的任务队列中，线程池中的一个线程会循环去任务队列中获取任务进行处理

线程池的实现:大量的线程+线程安全的任务队列　　应用场景:大量的数据请求的多任务并发处理

```
typedef void ('task handler t)(int data);
class ThreadTask{
    private:
        int_ data:// 要处理的数据
        task_ handler t. handler; //数据如何处理的方法
   public:
        void Run() {_ handler( data);} //线程池中的线程只需要调用这个接口就可以完成数据的处理
}
class ThreadPool
{
      private:
         int_ thr_ max; //最大线程数量(要创建的线程的个数)
         std:queue< ThreadTask>_queue (任务的信息都包含哪些:要处理的数据，不同的数据需要有不
         同的处理方法)
         //创建线程的时候，需要传入线程入口函数，导致线程池中的线程都是同一个完成任务的套路
         ，无法做到灵活应变
```

//相当于最好是抛入线程池的任务中就能告诉线程:要处理什么数据，以及如何处理的方法;线程

只需要调用方法处理数据即可

//线程池中的线程不管什么数据，不管如何处理，只管调用任务对象中的Run接口就可以

pthread_ mutex _t_ mutex; //用于实现队列操作的互斥

pthread_cond_t_cond_ pool;// 线程池中的线程都是消费者,若没有任务就等待在这里

```
public:
    Threadol(int thr_count){....资源初始化...}
    bool TaskPush(const ThreadTask &task);
private:
   static void *thread_ routine(void *arg){....循环从队列中获取任务，调用任务对象的Run方法完成任务处理....}
}
```

注意的思路:

1.通过任务类，实现向线程池抛入任务的时候，既抛入数据也抛入处理方法的思路

2.线程池只需要向外提供一-个任务入队接口即可，任务的处理都是在线程池内部完成的

3.线程池中的线程都只需要获取任务对象之后调用Run接口就可以实现任务处理

注意事项:

1.线程创建函数pthread_create要求传入的入口函数只有一个参数void (*thread_ routine)(void *arg);但是若入口函数是一个类的成员函数，则默认会有一个隐藏参数this指针导致函数参数类型不匹配;因此需要将这个入口函数在类内定义为静态函数

2.静态函数无法直接访问线程池对象;因此将这个对象的this指针通过线程入口函数传入;在线程内部就可以访问到这个对象了;但是类外使用，因此无法直接访问对象的私有成员，因此对私有成员的访问都需要通过公有接口来实现