类ThreadPoolExecutor

线程池中最为重要的一个变量，高3位代表运行状态，低29位代表线程数量
private final AtomicInteger ctl = new AtomicInteger(ctlOf(RUNNING, 0));
线程池处在RUNNING状态时，能够接收新任务，以及对已添加的任务进行处理。
private static final int RUNNING   = -1 << COUNT_BITS;
线程池处在SHUTDOWN状态时，不接收新任务，但能处理已添加的任务
private static final int SHUTDOWN   = 0 << COUNT_BITS;
线程池处在STOP状态时，不接收新任务，不处理已添加的任务，并且会中断正在处理的任务
private static final int STOP   = 1 << COUNT_BITS;
当所有的任务已终止，ctl记录的”任务数量”为0，线程池会变为TIDYING 状态。当线程池变为TIDYING状态时，会执行钩子函数terminated()
private static final int TIDYING   = 2 << COUNT_BITS;
线程池彻底终止，就变成TERMINATED状态。
private static final int TERMINATED = 3 << COUNT_BITS;

线程池实例化的7个重要的形参
public ThreadPoolExecutor(int corePoolSize, //线程池中的核心线程数
                          int maximumPoolSize, //线程池中允许的最大线程数
                          long keepAliveTime, //线程池维护线程所允许的空闲时间
                          TimeUnit unit, //keepAliveTime的单位
                          BlockingQueue<Runnable> workQueue, //用来保存等待被执行的任务的阻塞队列，且任务必须实现Runable接口，比如：
ArrayBlockingQueue，LinkedBlockingQuene
                          ThreadFactory threadFactory, //ThreadFactory类型的变量，用来创建新线程
                          RejectedExecutionHandler handler) //线程池的饱和拒绝策略

线程池的4个拒绝策略
1、AbortPolicy：直接抛出异常，默认策略；
2、CallerRunsPolicy：由调用者所在的线程来执行任务；
3、DiscardOldestPolicy：丢弃阻塞队列中靠最前的任务，将当前任务加入队列；
4、DiscardPolicy：直接丢弃任务；
实现RejectedExecutionHandler接口，可以自定义饱和策略

线程池执行任务的流程：
1. 判断当前线程池中存在的核心线程数 < 线程池的核心线程数corePoolSize，T:重新创建一个新的核心线程，由该核心线程执行任务，F:进行操作2
2. 判断添加任务到阻塞队列workQueue操作，T:将任务放入队列，等待核心线程执行，F:进行操作3
3. 判断当前线程池中存在的所有线程数 < 线程池的核心线程数maximumPoolSize,T:重新创建一个新的线程，由该线程执行任务，F:进行操作4
4. 执行拒绝策略