1） 什么是线程池？

线程池是一种多线程执行任务的处理形式， 引入它的目的是降低线程创建和销毁的开销 。 在客户端可以向线程池提交任务， 获取线程的状态， 管理线程池； 而线程池则负责使用线程执行任务， 客户端不必关心任务

打个比方来说： 如果一家公司有1000个客户， 如果每个客户都在不断地拨打这家公司的客户， 那么没有线程池的话， 我们需要为每个顾客创建一个线程处理，结束之后立刻销毁， 那么我们需要创建整整1000个线程， 而线程的创建，对应着内核也会创建一个线程， 这个资源开销实在太大了 。 而有了线程池， 相当于我们可以准备10个线程， 当有空时， 客户的热线得到处理； 无空时，等待一个线程空闲， 然后让其运行， 这大大减小了系统的资源开销

2）构造方法

new ThreadPoolExecutor

(coreSize , maxSize , keepAliveTime , timeUnit , BlockingQueue<Runnable> , ThreadFactory , RejectedHandler)

典型的四种线程池子newSingleThreadExecutor() ，newFixedThreadPool() , newCachedThreadPool() , newScheduledThreadPool()其实也只是上述构造方法引入不同的参数而已。

先简要介绍一下几个参数：

1. coreSize 线程池核心大小， 这里的线程除非调用者注定shutDown()，不然会一直存活着。

2 maxSize 线程池最大容纳的线程数目大小

3. keepAliveTime , timeUnit 线程池中数目超过超过coreSize ,并且这些线程在XX时间内没有接受处理新的任务， 就会被销毁， 直至线程数目 <= coreSize为止

4.ThreadFactory 线程创建的工厂

5. BlockingQueue<> 阻塞队列。一般分成

a.ArrayBlockingQueue. 有界， 预先定义，不可再改

b.LinkedBlockingQueue. 即可有界，也可无界。

c. synchrousBlockingQueue. 长度为1,适用于生产者和消费者速度差异不大的情况

6. RejectedHandler。 线程池中Therad数目已经到达最大值， 这时在提交任务， 有4种解决方法：

a. AbsortPolicy 直接抛弃任务 并且抛出异常

b. DiscardPolicy 抛弃任务， 但是无异常

c. DiscardOldestPolicy 抛弃队列中最早加入的任务

d. CallerRunnerPolicy。 交给直接调用者处理

3）工作过程： a.在初始时候， 默认client每次提交一个任务， 线程池就会创建一个线程， 直至线程数目到达coreSize大小。

（当然也可以使用preStartThread()和 preStartAllThread() 预先创建线程。 前者预先创一个 而后者创建coreSize个线程， 以作预热）

b. 之后， 在提交任务， 则会放入到工作队列中， 直至工作队列已满（也有无界组合队列linkedBlockingQueue,注意内存的爆炸）。

c. 在之后， 提交任务， 线程池会继续创建线程， 直至线程数目到达maxSize.

d. 若还有任务提交， 则交由RejectHandler处理了

4） 常用线程池。

ExecutorService pool = Executors.XXXXX();

a. newSingleThreadExecutor() : 一个单线程池， 保证顺序执行

b. newFixedThreadPool() : 指定固定线程数目的线程池， 适用于已知并发压力

c.newCachededThreadPool(): 可以无限扩大的线程池， 最好只执行短时间小任务，否则容易OOM， 默认不工作的线程时间是60s

d. newScheduledThreadPool() : 可以延期， 周期性执行并发任务

5）常用方法

a. shutDown() , pool不再接受新的任务， 但也不会强制终止正执行和已经提交的任务

b. shunDownNow(). kill掉未开始执行的任务， 并且返回这些任务的列表， 中断正在执行的

List<Runnable> res = pool.shutDownNow();

c.execute(). 提交任务 submit().提交任务， 但是会返回处理结果， 本质上还是调用execute() ,但是会用future保存结果

d. getPoolSize() , getQueue() , getActiviteCount() , getCompletedeTask().....