***Java 并发编程 ： 线程池***

Java提供了java.util.concurrent.Executor接口的实现用于创建线程池。

**假设一个服务器完成一项任务所需时间为：T1 创建线程的时间，T2 在线程中执行任务的时间，T3 销毁线程的时间**

**T1 + T3 远远大于T2.则可以采用线程池**

**一个线程池包括以下四个基 ptg本组成部分：**

1. **线程管理器（Threadpool）:用于创建并管理线程池，包括创建线程池，销毁线程池，添加新任务。**
2. **工作线程（PoolWorker）：线程池中线程，在没有任务时处于等待状态，可以循环的执行任务；**
3. **任务接口（Task）：每个任务必须实现的接口，以供工作线程调度任务的执行，它主要规定了任务的入口，任务执行完后的收尾工作，任务的执行状态等；**
4. **任务队列（taskQueue）：用于存放没有处理的任务。提供一种缓冲机制。**

**1. newSingleThreadExecutor**

创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

2.**newFixedThreadPool**

创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

**3. newCachedThreadPool**

创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，

那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以[智能](http://lib.csdn.net/base/aiplanning" \o "人工智能规划与决策知识库" \t "_blank)的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于[操作系统](http://lib.csdn.net/base/operatingsystem)（或者说JVM）能够创建的最大线程大小。

4.**newScheduledThreadPool**

创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求

**ThreadPoolExecutor 继承自 AbstractExecutorService 实现了 ExecutorService 接口，ScheduledThreadPoolExecutor 继承自 ThreadPoolExecutor 实现了 ExecutorService 和 ScheduledExecutorService 接口**

Java 线程池**ThreadPoolExecutor**

ThreadPoolExecutor继承了**AbstractExecutorService**类，并提供了四个构造器，事实上，通过观察每个构造器的源码具体实现，发现前面三个构造器都是调用的第四个构造器进行的初始化工作。

corePoolSize：核心池的大小，默认情况下，在创建了线程池后，线程池中的线程数为0，当有任务来之后，就会创建一个线程去执行任务，当线程池中的线程数目达到corePoolSize后，就会把到达的任务放到缓存队列当中。

maximumPoolSize：线程池最大线程数，这个参数也是一个非常重要的参数，它表示在线程池中最多能创建多少个线程；

keepAliveTime：表示线程没有任务执行时最多保持多久时间会终止。默认情况下，只有当线程池中的线程数大于corePoolSize时，keepAliveTime才会起作用，直到线程池中的线程数不大于corePoolSize，即当线程池中的线程数大于corePoolSize时，如果一个线程空闲的时间达到keepAliveTime，则会终止，直到线程池中的线程数不超过corePoolSize。

unit：参数keepAliveTime的时间单位（天，小时，分钟，。。。，微秒）

workQueue：一个阻塞队列，用来存储等待执行的任务

**ArrayBlockingQueue：一个由数组支持的有界阻塞队列，规定大小的BlockingQueue,其构造函数必须带一个int参数来指明其大小.其所含的对象是以FIFO(先入先出)顺序排序的。**

**LinkedBlockingQueue：大小不定的BlockingQueue,若其构造函数带一个规定大小的参数,生成的BlockingQueue有大小限制,若不带大小参数,所生成的BlockingQueue的大小由Integer.MAX\_VALUE来决定.其所含的对象是以FIFO(先入先出)顺序排序的。**

**PriorityBlockingQueue：类似于LinkedBlockQueue,但其所含对象的排序不是FIFO,而是依据对象的自然排序顺序或者是构造函数的Comparator决定的顺序。**

**SynchronousQueue：特殊的BlockingQueue,对其的操作必须是放和取交替完成的。**

**LinkedBlockingQueue 可以指定容量，也可以不指定，不指定的话，默认最大是Integer.MAX\_VALUE,其中主要用到put和take方法，put方法在队列满的时候会阻塞直到有队列成员被消费，take方法在队列空的时候会阻塞，直到有队列成员被放进来。**

**5.任务拒绝策略**

**当线程池的任务缓存队列已满并且线程池中的线程数目达maximumPoolSize，如果还有任务到来就会采取任务拒绝策略，通常有以下四种策略：**

**ThreadPoolExecutor.AbortPolicy:丢弃任务并抛出RejectedExecutionException异常。**

**ThreadPoolExecutor.DiscardPolicy：也是丢弃任务，但是不抛出异常。**

**ThreadPoolExecutor.DiscardOldestPolicy：丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）**

**ThreadPoolExecutor.CallerRunsPolicy：由调用线程处理该任务**

**threadFactory**：线程工厂，主要用来创建线程；

handler：表示当拒绝处理任务时的策略，有以下四种取值

ThreadPoolExecutor.AbortPolicy:丢弃任务并抛出RejectedExecutionException异常。

ThreadPoolExecutor.DiscardPolicy：也是丢弃任务，但是不抛出异常。

ThreadPoolExecutor.DiscardOldestPolicy：丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）

ThreadPoolExecutor.CallerRunsPolicy：由调用线程处理该任务

2. 深入理解线程池的底层原理

1. 线程池的状态

在ThreadPoolExecutor中定义了一个volatile变量（使用volatile保持各个线程之间的可见性），另外定义了几个static final变量表示线程池的各个状态：

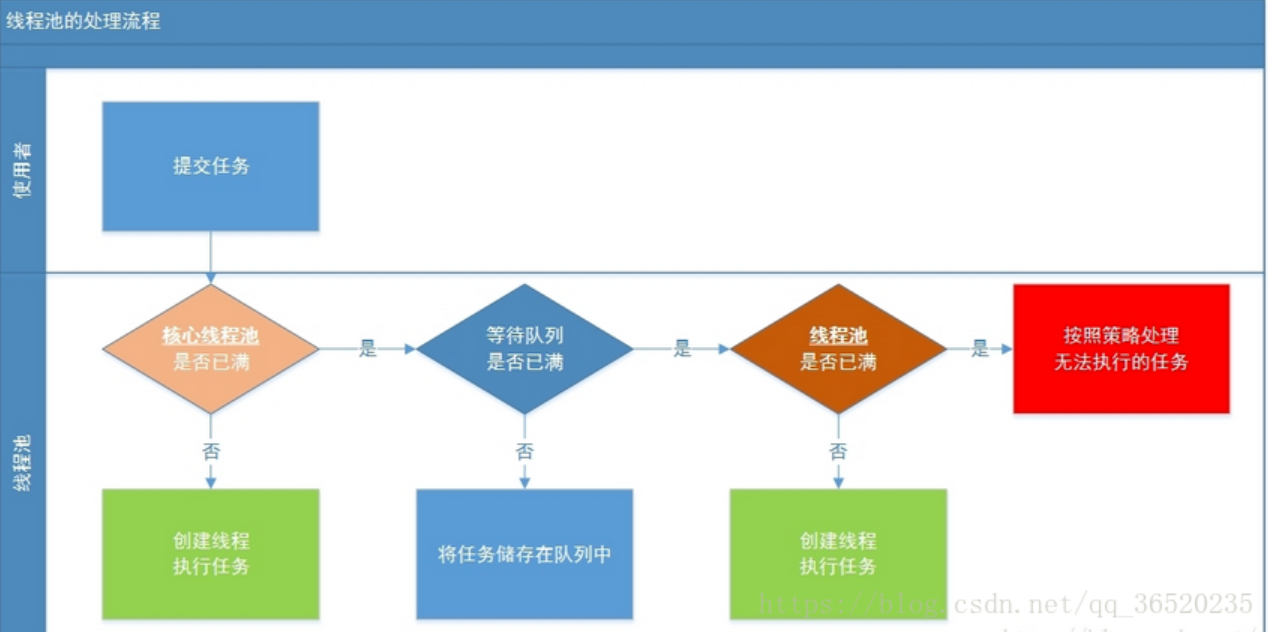
当创建线程池后，初始时，线程池处于RUNNING状态；

如果调用了shutdown()方法，则线程池处于SHUTDOWN状态，此时线程池不能够接受新的任务，它会等待所有任务执行完毕；

如果调用了shutdownNow()方法，则线程池处于STOP状态，此时线程池不能接受新的任务，并且会去尝试终止正在执行的任务；

当线程池处于SHUTDOWN或STOP状态，并且所有工作线程已经销毁，任务缓存队列已经清空或执行结束后，线程池被设置为TERMINATED状态。

2.任务的执行



第一步：当有任务提交过来的时候其实也就是当执行excute（）的时候，首先会先判断当前正在运行的线程是不是大于核心线程池（corePoolSIze），（1）如果大于说明核心线程池已经满了，然后把当前线程放到阻塞对列中：（2）如果小于说明核心线程池还有剩余，就直接创建一个线程执行任务，也就是调用addIfUnderCorePoolSize(command)：

在addIfUnderCorePoolSize中通过lock 和 unlock（）锁住相关代码；在这个里面又调用了addThread；这个threadFactory创建一个线程，并对coolpoolsize+1.

在这个里面通过getTask（），从缓冲队列中获取任务去执行。

Java线程中的feature？

Feature接口是在jdk1.5之后被引入，设计初衷是对将来的某个时刻发生的结果进行建模。它建模了一种异步计算，返回一个执行计算的引用结果。当运算结束后，这个引用被返回给调用方。

在Future接口中声明了5个方法，下面依次解释每个方法的作用：

cancel 方法用来取消任务，如果取消任务成功则返回true，如果取消任务失败则返回false。参数mayInterruptIfRunning表示是否允许取消正在执行却没有执行完毕的任务，如果设置true，则表示可以取消正在执行过程中的任务。如果任务已经完成，则无论mayInterruptIfRunning为true还是false，此方法肯定返回false，即如果取消已经完成的任务会返回false；如果任务正在执行，若mayInterruptIfRunning设置为true，则返回true，若mayInterruptIfRunning设置为false，则返回false；如果任务还没有执行，则无论mayInterruptIfRunning为true还是false，肯定返回true。

isCancelled方法表示任务是否被取消成功，如果在任务正常完成前被取消成功，则返回 true。

isDone方法表示任务是否已经完成，若任务完成，则返回true；

get()方法用来获取执行结果，这个方法会产生阻塞，会一直等到任务执行完毕才返回；

get(long timeout, TimeUnit unit)用来获取执行结果，如果在指定时间内，还没获取到结果，就直接返回null。

　　也就是说Future提供了三种功能：

　　1）判断任务是否完成；

　　2）能够中断任务；

　　3）能够获取任务执行结果。

因为Future只是一个接口，所以是无法直接用来创建对象使用的，因此就有了下面的FutureTask。

FutureTask是Future接口的一个唯一实现类



Java 线程之间的通信方式？

1. 同步 synorchinzed

如果两个线程共同持有一个对象，尽管这两个线程需要调用不同的方法，但是它们是同步执行的。比如线程B要等待线程A执行完后methodB（）方法才能执行methodA（）方法。

1. 等待通知机制 wait/notify

当前线程A调用了wait（）方法进入等待状态， 而另一个线程B调用了对象的notify（）或者notifyall（）方法，线程A收到通知后进而继续执行后续的操作。

1. 管道输入输出流

主要分为两种面向字节（PipedOutputStream；。。。Input）和面向字符（PipedReader）。

4) Thread.join

如果一个线程Thread。Join；其含义位当前线程A等待thread线程终止后才从thread。Join后返回。（其实就是join()方法将挂起**调用线程**的执行，直到**被调用的对象**完成它的执行。）这里执行join()时关键字synchronized，锁是该对象实例（this）,通过Object的本地方法wait（秒数），而Wait()方法调用时释放锁资源，当main线程调用一个线程的join方法，main线程会获得线程对象的锁,调用wait(等待时间)，直到该对象唤醒main线程（时间到了或者线程执行结束）。

1. threadlocal

threadlocal线程本地变量，其为变量在每个线程中都创建了一个副本；每个线程可以访问自己内部的副本变量；各个线程之间互不干扰。

Threadlocal中有get（），set（），remove（）方法。

首先看get（）方法；获取当前对象的threadlocal，然后从map中获取k—V的Entry节点。从Entry中获取的value值返回；若map为空，则返回空。

每一个thread对象都有一个threadlocalmap，当创建一个threadlocalmap时，就会将该threadlocal对象添加到map中；主键就是threadlocal；值可以时任意类型的。

Threadlocalmap解决hash冲突的方法是线性探测，简单的步长加1或减1.

Threadlocalmap中key是弱引用，value是强引用。当threadlocal没有外部对象引用时，发生gc会回收，而value不会被回收。最终导致内存泄漏。