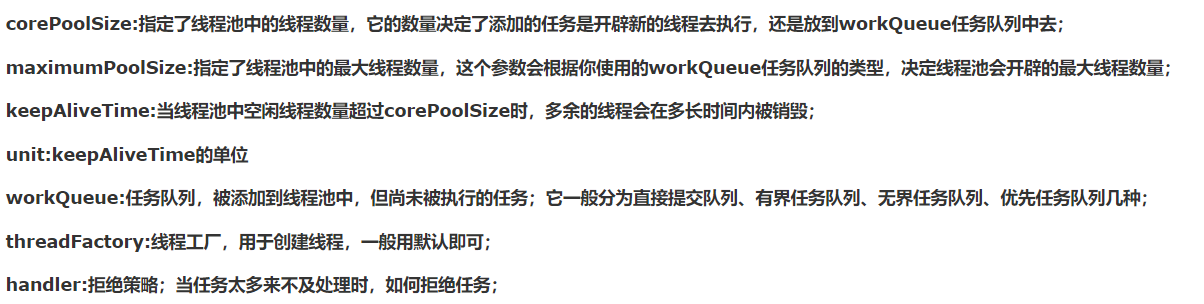
参考 https://www.cnblogs.com/dafanjoy/p/9729358.html

**ThreadPoolExecutor构造方法**



**ThreadFactory**

**作用：用来创建线程，接口只有个newThread(Runnable r);**

**给线程命名**

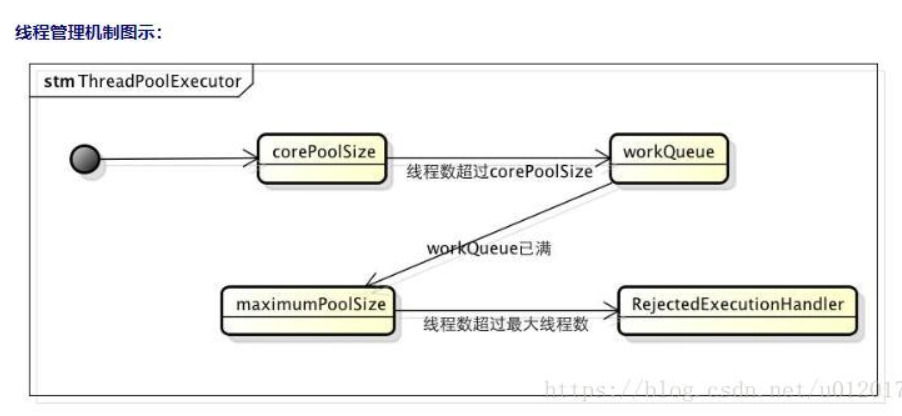
**查看创建线程数（atomicInteger）**

**给线程设置是否守护线程**

**设置线程优先级 //都是Thread本身具有的方法**

<https://www.jianshu.com/p/b812b68b04fb>

**ThreadPoolExecutor执行的策略**



**线程数量未达到核心线程数，则新建一个线程执行任务**

**线程数量达到了corePools，则将任务移入队列等待**

**队列已满，新建非核心线程执行任务**

**队列已满，线程数又达到了最大线程数，就会由拒绝策略**

**workQueue任务队列**

**阻塞队列（BlockingQueue）**是一个支持两个附加操作的队列

在队列为空时，获取元素的线程会等待队列变为非空

当队列满时，存储元素的线程会等待队列可用

常用于生产者和消费者的场景

分为直接提交队列、有界任务队列、无界任务队列、优先任务队列

* 直接提交队列：SynchronousQueue

它没有容量，提交的任务不会被缓存起来，而是会被马上执行

每个插入操作必须等待另一个线程的移除操作，反之亦然

使用SynchronousQueue队列，提交的任务不会被保存，总是会马上提交执行。如果用于执行任务的线程数量小于maximumPoolSize，则尝试创建新的进程，如果达到maximumPoolSize设置的最大值，则根据你设置的handler执行拒绝策略

* 有界的任务队列 ArrayBlockingQueue

若有新的任务需要执行时，线程池会创建新的线程，直到创建的线程数量达到corePoolSize时，则会将新的任务加入到等待队列中。若等待队列已满，即超过ArrayBlockingQueue初始化的容量，则继续创建线程，直到线程数量达到maximumPoolSize设置的最大线程数量，若大于maximumPoolSize，则执行拒绝策略。在这种情况下，线程数量的上限与有界任务队列的状态有直接关系，如果有界队列初始容量较大或者没有达到超负荷的状态，线程数将一直维持在corePoolSize以下，反之当任务队列已满时，则会以maximumPoolSize为最大线程数上限

* **无界的任务队列 LinkedBlockingQueue**

使用无界任务队列，线程池的任务队列可以无限制的添加新的任务，而线程池创建的最大线程数量就是你corePoolSize设置的数量，也就是说在这种情况下maximumPoolSize这个参数是无效的，哪怕你的任务队列中缓存了很多未执行的任务，当线程池的线程数达到corePoolSize后，就不会再增加了；若后续有新的任务加入，则直接进入队列等待，当使用这种任务队列模式时，一定要注意你任务提交与处理之间的协调与控制，不然会出现队列中的任务由于无法及时处理导致一直增长，直到最后资源耗尽的问题

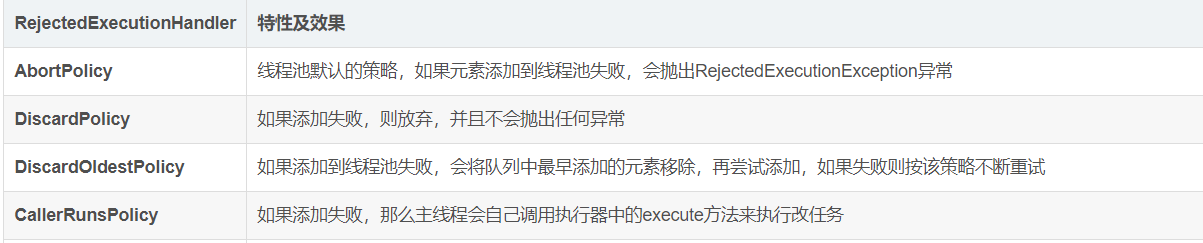
如果不指定队列的容量大小，也就是使用默认的Integer.MAX\_VALUE

* **优先任务队列 PriorityBlockingQueue**

元素必须实现Comparable接口，使用元素的compareTo方法来确定元素执行的优先级

是一个特殊的无界队列，它其中无论添加了多少个任务，线程池创建的线程数也不会超过corePoolSize的数量，只不过其他队列一般是按照先进先出的规则处理任务，而PriorityBlockingQueue队列可以自定义规则根据任务的优先级顺序先后执行

**拒绝策略 RejectedExecutionHandler 处理线程池"超载"的情况**

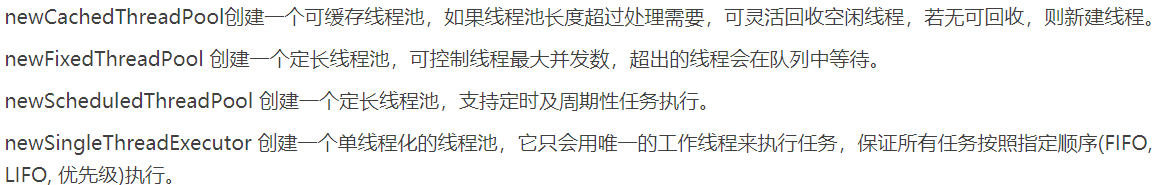


**CallerRunsPolicy**

**在任务被拒绝添加后，会调用当前线程池的所在的线程去执行被拒绝的任务**(mian，缺点是会阻塞主线程)

**Executors 类提供4种线程池**

实际还是利用 ThreadPoolExecutor 类实现 （根据传入的任务队列类型）



可控制线程最大并发数（同时执行的线程数）

**new Thread的弊端如下：**

**a. 每次new Thread新建对象，性能差**

**b. 线程缺乏统一管理，可能无限制新建线程，相互之间竞争，及可能占用过多系统资源导致死机或oom内存溢出**

**c. 缺乏更多功能，如定时执行、定期执行、线程中断**

**什么是线程池**

**线程池是一种对象池，在程序启动时就开辟一块内存空间，里面存放一些(未死亡)的线程，池中线程的执行调度由池管理器来处理。当有线程任务时，从池中取一个，执行完成后线程对象归池，这样可以避免反复创建线程对象所带来的性能开销，节省了系统的资源**

**线程池的好处：**

**a. 重用存在的线程，减少线程创建、消亡的开销，性能好**

**b. 可以控制最大并发线程数，根据系统的承受能力，调整线程池中工作线线程的数目**

**c. 可以对线程进行一些简单的管理，延时执行、定期执行等**

**一个线程池包括以下四个基本组成部分：**

**线程池管理器：用于创建并管理线程池，包括创建、销毁线程池，添加任务**

**工作线程： 线程池中线程，在没有任务时处于等待状态，可以循环执行任务**

**任务接口： 每个任务必须实现的接口，以供工作线程调度任务的执行Runnble/Callable**

**任务队列： 用于存放没有处理的任务。提供一种缓冲机制**

**向线程池提交任务**

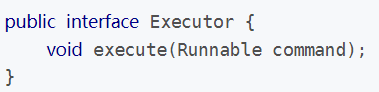
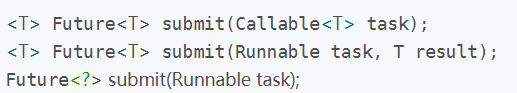
**可以通过execute()或submit()向线程池提交任务**

**execute()方法没有返回值，所以无法判断任务是否被线程池执行成功**

**submit()方法返回一个future,那么我们可以通过这个future来判断任务是否执行成功，通过future的get方法来获取返回值**

Future也是一个接口，通过它可以获得任务执行的返回值

future.get()具有阻塞性 等到任务执行完毕，拿到了返回的返回值，主线程才会继续运行

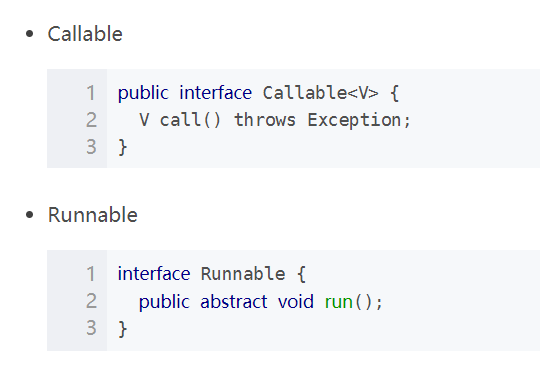
 

**Callable能接受一个泛型，然后在call方法中返回一个这个类型的值**

**Runnable的run方法没有返回值**

**Callable的call方法可以抛出异常，而Runnable的run方法不会抛出异常**

**Callable和Runnable**



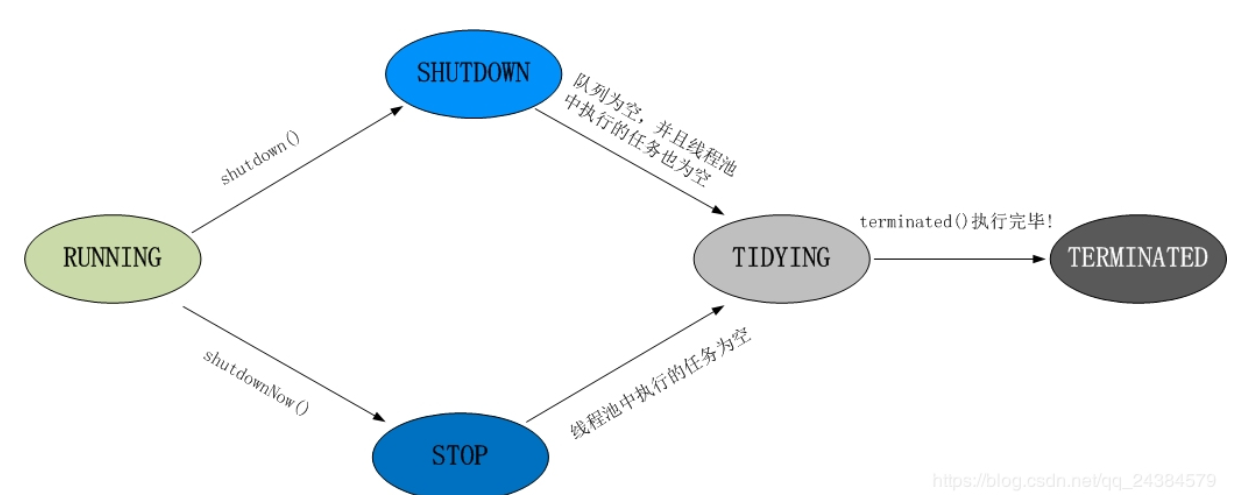
**线程池的关闭**

**我们可以通过shutdown()或shutdownNow()方法来关闭线程池，不过它们也有所不同**

**shutdown只是将线程池的状态设置为SHUTWDOWN状态，正在执行任务的线程会继续执行下去，没有被执行的则中断，线程池直到所有任务执行完才会停止**

**shutdownNow线程池立刻变成STOP状态，通过调用Thread.interrupt()方法来尝试中断所有的任务，返回值List<Runnable>未执行任务的列表**

**线程池的状态**



**1.RUNNING**

**状态说明：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**

**状态切换：线程池的初始化状态是RUNNING。线程池被一旦被创建，就处于RUNNING状态，并且线程池中的任务数为0**

**2.SHUTDOWN**

**状态说明：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**

**状态切换：调用线程池的shutdown()接口时，线程池由RUNNING -> SHUTDOWN**

**3.STOP**

**状态说明：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**

**状态切换：调用线程池的shutdownNow()接口时，线程池由(RUNNING or SHUTDOWN ) -> STOP**

**4.TIDYING**

**状态说明：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terminated()函数**

terminated()在ThreadPoolExecutor类中是空的，若用户想在线程池变为TIDYING时，进行相应的处理；可以通过重载terminated()函数来实现

**状态切换：当线程池在SHUTDOWN状态下，阻塞队列为空并且线程池中执行的任务也为空时，就会由 SHUTDOWN -> TIDYING**

**当线程池在STOP状态下，线程池中执行的任务为空时，就会由STOP -> TIDYING**

**5.TERMINATED**

**状态说明：线程池彻底终止，就变成TERMINATED状态。**

**状态切换：线程池处在TIDYING状态时，执行完terminated()之后，就会由 TIDYING -> TERMINATED**

控制状态的属性叫ctl，它是一个AtomicInteger类型的变量，它包含两个概念：

workerCount：表明当前有效的线程数

runState：表明当前线程池的状态，是否处于Running，Shutdown，Stop，Tidying，Terminate五种状态

回调函数又称是钩子函数

**Executor和Executors的区别**

Executor 接口对象能执行我们的线程任务；

Executors 工具类可以按照需求创建不同的线程池，来满足业务的需求

ExecutorService 接口继承了Executor接口并进行了扩展，提供了更多的方法，我们能够获得任务执行的状态并且可以获取任务的返回值