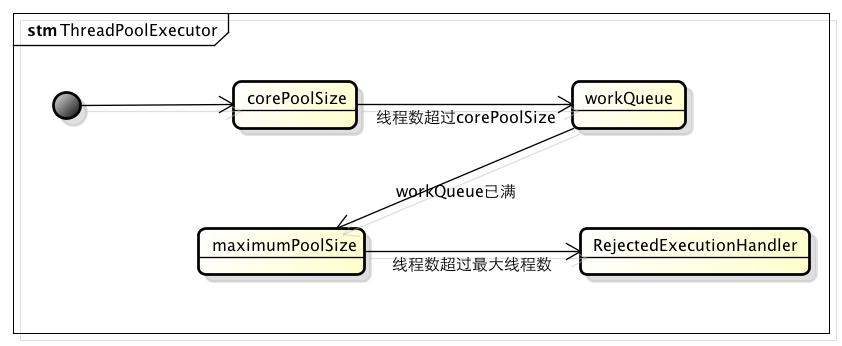
**ThreadPoolExecutor**

|  |  |  |
| --- | --- | --- |
| 参数名 |  | 作用 |
| corePoolSize |  | 核心线程池大小 |
| maximumPoolSize |  | 最大线程池大小(线程总数 = 核心线程数 + 非核心线程数) |
| keepAliveTime |  | 线程池中超过corePoolSize数目的空闲线程最大存活时间；可以allowCoreThreadTimeOut(true)使得核心线程有效时间 |
| TimeUnit |  | keepAliveTime时间单位 |
| workQueue |  | 阻塞任务队列 |
| threadFactory |  | 新建线程工厂 |
| RejectedExecutionHandler |  | 当提交任务数超过maxmumPoolSize+workQueue之和时，任务会交给RejectedExecutionHandler来处理 |



**1、构造一个固定线程数目的线程池，配置的corePoolSize与maximumPoolSize大小相同，同时使用了一个无界LinkedBlockingQueue存放阻塞任务，因此多余的任务将存在再阻塞队列，不会由RejectedExecutionHandler处理** 

public static ExecutorService **newFixedThreadPool**(int nThreads) {

return new ThreadPoolExecutor(nThreads, nThreads,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

}

**2、构造一个缓冲功能的线程池，配置corePoolSize=0，maximumPoolSize=Integer.MAX\_VALUE，keepAliveTime=60s,以及一个无容量的阻塞队列 SynchronousQueue，因此任务提交之后，将会创建新的线程执行；线程空闲超过60s将会销毁** 

public static ExecutorService **newCachedThreadPool**() {

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

**3、构造一个只支持一个线程的线程池，配置corePoolSize=maximumPoolSize=1，无界阻塞队列LinkedBlockingQueue；保证任务由一个线程串行执行** 

public static ExecutorService **newSingleThreadExecutor**() {

return new FinalizableDelegatedExecutorService

(new ThreadPoolExecutor(1, 1,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>()));

}

**4、构造有定时功能的线程池，配置corePoolSize，无界延迟阻塞队列DelayedWorkQueue；有意思的是：maximumPoolSize=Integer.MAX\_VALUE，由于DelayedWorkQueue是无界队列，所以这个值是没有意义的** 

public static ScheduledExecutorService **newScheduledThreadPool**(int corePoolSize) {

return new ScheduledThreadPoolExecutor(corePoolSize);

}

public static ScheduledExecutorService **newScheduledThreadPool**(

int corePoolSize, ThreadFactory threadFactory) {

return new ScheduledThreadPoolExecutor(corePoolSize, threadFactory);

}

public **ScheduledThreadPoolExecutor**(int corePoolSize,

ThreadFactory threadFactory) {

super(corePoolSize, Integer.MAX\_VALUE, 0, TimeUnit.NANOSECONDS,

new DelayedWorkQueue(), threadFactory);

}

**常用的workQueue类型：**

**SynchronousQueue**：这个队列接收到任务的时候，会直接提交给线程处理，而不保留它，如果所有线程都在工作怎么办？那就新建一个线程来处理这个任务！所以为了保证不出现<线程数达到了maximumPoolSize而不能新建线程>的错误，使用这个类型队列的时候，maximumPoolSize一般指定成Integer.MAX\_VALUE，即无限大（缓存线程池）

**LinkedBlockingQueue**：这个队列接收到任务的时候，如果当前线程数小于核心线程数，则新建线程(核心线程)处理任务；如果当前线程数等于核心线程数，则进入队列等待。由于这个队列没有最大值限制，即所有超过核心线程数的任务都将被添加到队列中，这也就导致了maximumPoolSize的设定失效，因为总线程数永远不会超过corePoolSize（固定大小的线程池）

**ArrayBlockingQueue**：可以限定队列的长度，接收到任务的时候，如果没有达到corePoolSize的值，则新建线程(核心线程)执行任务，如果达到了，则入队等候，如果队列已满，则新建线程(非核心线程)执行任务，又如果总线程数到了maximumPoolSize，并且队列也满了，则发生错误

**DelayQueue**：队列内元素必须实现Delayed接口，这就意味着你传进去的任务必须先实现Delayed接口。这个队列接收到任务时，首先先入队，只有达到了指定的延时时间，才会执行任务

**ThreadFactory**

创建线程的方式，这是一个接口，你new他的时候需要实现他的Thread newThread(Runnable r)方法，一般用不上。

**RejectedExecutionHandler**

这玩意儿就是**抛出异常专用**的，比如上面提到的两个错误发生了，就会由这个handler抛出异常，根本用不上。

**ThreadPoolExecutor.execute(Runnable command)方法即可向线程池内添加一个任务。**

**ThreadPoolExecutor的策略**

1.线程数量未达到corePoolSize，则新建一个线程(核心线程)执行任务

2.线程数量达到了corePools，则将任务移入队列等待

3.队列已满，新建线程(非核心线程)执行任务

4.队列已满，总线程数又达到了maximumPoolSize，就会由(RejectedExecutionHandler)抛出异常

##### 可缓存线程池CachedThreadPool()

**可变线程数，可以将用完的线程重新利用。线程空闲超过60s将会销毁**

##### FixedThreadPool 定长线程池

**固定线程池，使用完毕必须手动关闭线程池，否则会一直在内存中存在**

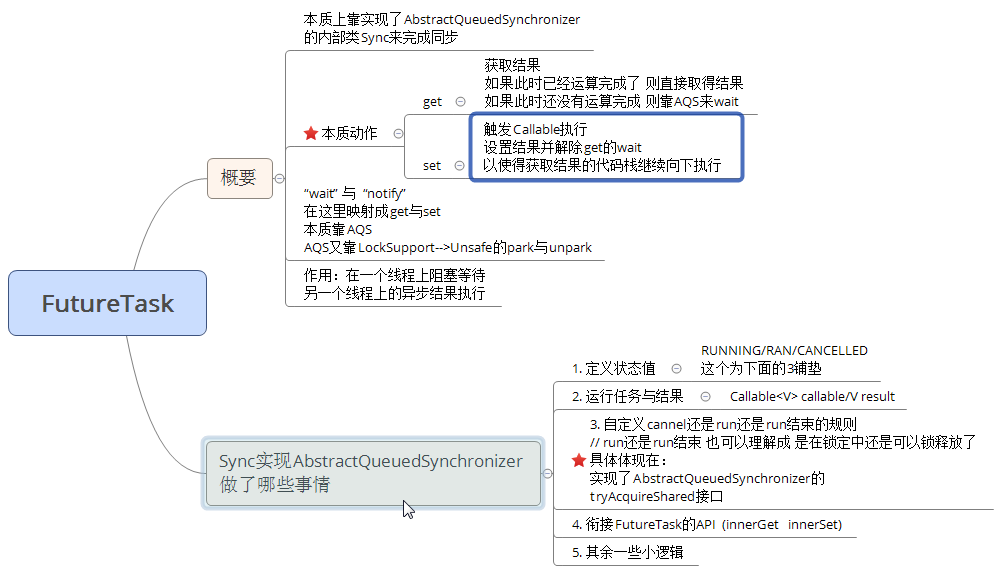
##### SingleThreadPool

**有且仅有一个工作线程执行任务，所有任务按照指定顺序执行，即遵循队列的入队出队规则。也需要手动关闭线程池。**

SingleThreadExecutor可以确保任何线程中都只有唯一的任务在运行。（多个线程使用同一文件系统时，可以用SingleThreadExecutor来保持同步）

##### ScheduledThreadPool

**定时执行线程，ScheduledExecutorService**。**也需要手动关闭线程池。**



**这是Callable创建线程返回值阻塞的原理**

**CompletableFuture非阻塞（未学）**