1. **几种线程池的区别**

* **newFixedThreadPool**

newFixedThreadPool创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。

* **newCachedThreadPool**

1).工作线程的创建数量几乎没有限制(其实也有限制的,数目为Interger. MAX\_VALUE), 这样可灵活的往线程池中添加线程。   
2).如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为1分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

* **newSingleThreadExecutor**

newSingleThreadExecutor创建一个单线程化的Executor，即只创建唯一的工作者线程来执行任务，如果这个线程异常结束，会有另一个取代它，保证顺序执行(我觉得这点是它的特色)。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的 。

* **newScheduleThreadPool**

newScheduleThreadPool创建一个定长的线程池，而且支持定时的以及周期性的任务执行，类似于Timer。(这种线程池原理暂还没完全了解透彻)

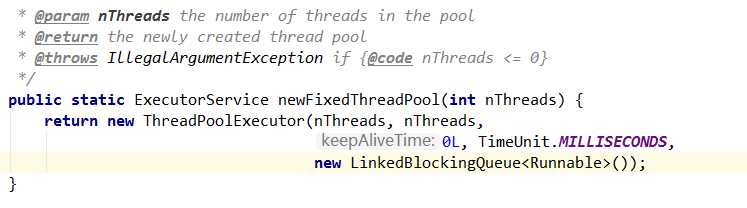
* **newWorkStealingPool**

**1.1** **newFixedThreadPool**

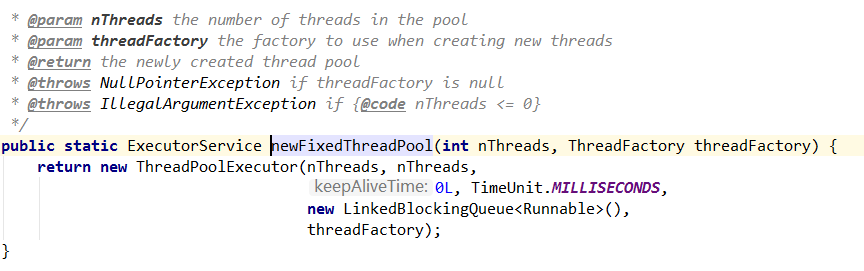
**查看**newFixedThreadPool的构造函数：

NewFixedThreadPool的构造函数有两个：

1)



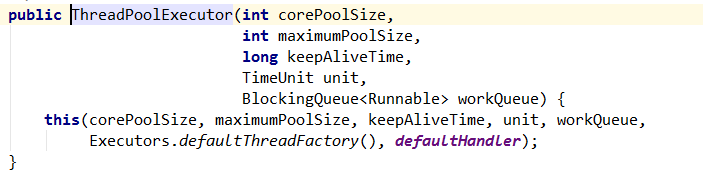
2)



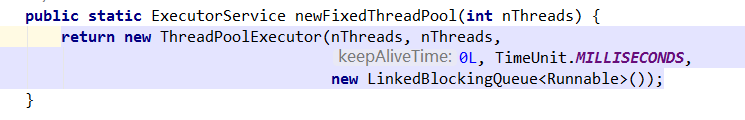
**1.1.1 构造函数情况1**

现在先看第一个构造函数，参数只有一个nThreads，线程池中的线程数，超出线程数的部分会加入LinkedBlockingQueue的等待队列，而LinkedBlockingQueue的理论值最大为int的最大值，那么现在思考一个问题，如果超出这个队列后，还有任务继续过来会怎样，可以查看跟踪查看Executors.newFixedThreadPool(num)和ThreadPoolExecutor的源码：

进行Executors.newFixedThreadPool(num)的时候，代码会返回一个ThreadPoolExecutor实例，查看ThreadPoolExecutor的构造函数，



对比newFixedThreadPool的构造函数：



* ThreadPoolExecutor线程池几个参数如下：

核心线程数：nThreads；

最大线程数：nThreads；

keepAliveTime：0L；(超过keepAliveTime的时间，如果还没有新任务，则释放核心线程数之外的线程，保留核心线程数数目的线程)

时间单位：秒；

等待队列：BlockingQueue；

* 从ThreadPoolExecutor构造函数可以看到这里的核心线程数和最大线程数是相同的；
* 线程池执行器逻辑：

1. 在一开始的时候，线程池是完全没有线程的；
2. 当有任务需要执行，则开始创建线程，并且当线程数少于核心线程数时，该部分的线程会一直保持正常状态而不会回收，线程数量在核心数量以内的，一旦创建就会保持这个数目的线程；
3. 当线程数已经大于或等于核心线程数，但是又继续有新的任务进来，那么会比较当前运行的线程数是否小于最大线程数，如果小于，则继续创建新的线程，超出核心线程数部分的空闲线程，会在超过keepAliveTime时自动回收；
4. 如果正在运行的线程任务已经大于等于最大线程数，而又继续有新的线程进来，那么这些任务就会被加入到等待队列，等待队列的理论值为整数型int的最大值；
5. 如果等待队列已经满了，那么会检查此刻是否有空闲线程，如果有则把任务交给空闲线程去执行，如果没有则抛出异常，告诉使用者已经不能再接收新的任务了；

* 关于队列满了后并且没有空闲的线程，默认的情况下才会抛出异常，因为ThreadPoolExecutor的构造函数实际上可以提供预定义处理策略：

1) 默认的ThreadPoolExecutor.AbortPolicy 处理程序遭到拒绝将抛出运行时RejectedExecutionException；

2) ThreadPoolExecutor.CallerRunsPolicy 线程调用运行该任务的 execute 本身。此策略提供简单的反馈控制机制，能够减缓新任务的提交速度；

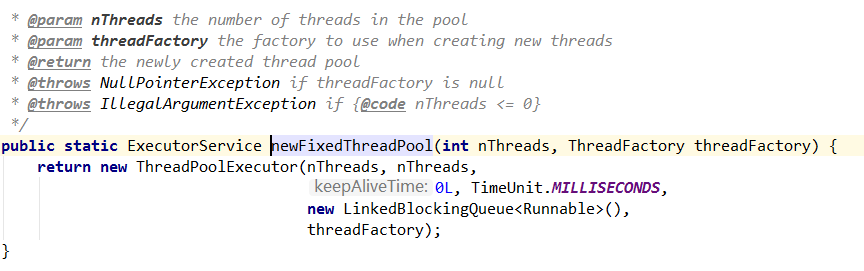
3) ThreadPoolExecutor.DiscardPolicy 不能执行的任务将被删除；

4) ThreadPoolExecutor.DiscardOldestPolicy 如果执行程序尚未关闭，则位于工作队列头部的任务将被删除，然后重试执行程序（如果再次失败，则重复此过程）；

测试demo：NewFixedThreadPoolDemoTest；

**1.1.2 构造函数情况2**

1.1.1已经讨论了不带ThreadFactory构造函数的时候，现在来讨论带有ThreadFactory参数的构造函数，比较两者的差异；



* ThreadFactory的作用是生产线程，可以对生产的线程进行计数统计、设置分组、优先级等等操作；
* 那么其实两者的区别就在于是否拥有一个生产线程的线程工厂；

**1.2 newCachedThreadPool**

newCachedThreadPool跟newFixedThreadPool的构造函数一样有两个，既然从上面可以得知两个的区别，那么在测试中就直接使用带有ThreadFactory参数的构造函数，这样方便监控执行任务的线程是新建的，还是使用了空闲线程；测试demo：NewnewCachedThreadPoolDemoTest；

>>经过测试demo：NewnewCachedThreadPoolDemoTest的测试，可以得出结论：

当有任务来时，线程池会产生线程来执行任务，而当任务执行完成后，线程成为空闲线程，如果有新的任务来临，那么系统就会把该空闲线程用来继续执行任务，而当没有新的任务来临，空闲线程超过了时间（默认是一分钟）没有执行任务，那么该线程就会回收，下次再有任务来临时，系统里如没有了空闲线程，那么就会重新创建新的线程来执行任务；跟newFixedThreadPool的区别在于，线程池创建线程没有核心线程数、最大线程数的概念，线程能够持续创建的线程数是无限的（理论上是有的，数目为整数型的最大值）。

**1.3 newSingleThreadPool**

测试demo：NewSingleThreadPoolDemoTest

* 同样有两个构造函数，一个是只有一个参数：核心线程数，一个是有两个参数：核心线程数和线程工厂；这里的测试使用了拥有线程工厂的核心线程数，方便观察线程是否是新建；

>>根据测试结果可以得知，线程数目的创建会根据核心线程数来确定，线程数有多少，则会一共可能只会创建多少，所有的任务都交由这几个线程来进行排队顺序执行；执行逻辑：当有新的线程来时，会观察是否存在空闲线程，如果没有空闲线程，则看目前创建的线程数是否已经达到了构造函数设置的线程数，如果没有达到，那么会继续创建线程来执行该任务，如果线程数已经饱和了，数目已达到了核心数，那么会把任务放到先到的任务后面，进行排队，等待有空闲线程到来后，再按照先到先来的顺序执行任务；注意：核心线程在空闲的时候也不会被回收，会一直保存正常状态等待任务到来；

**1.4 newScheduleThreadPool**

测试demo：NewScheduledThreadPoolDemoTest

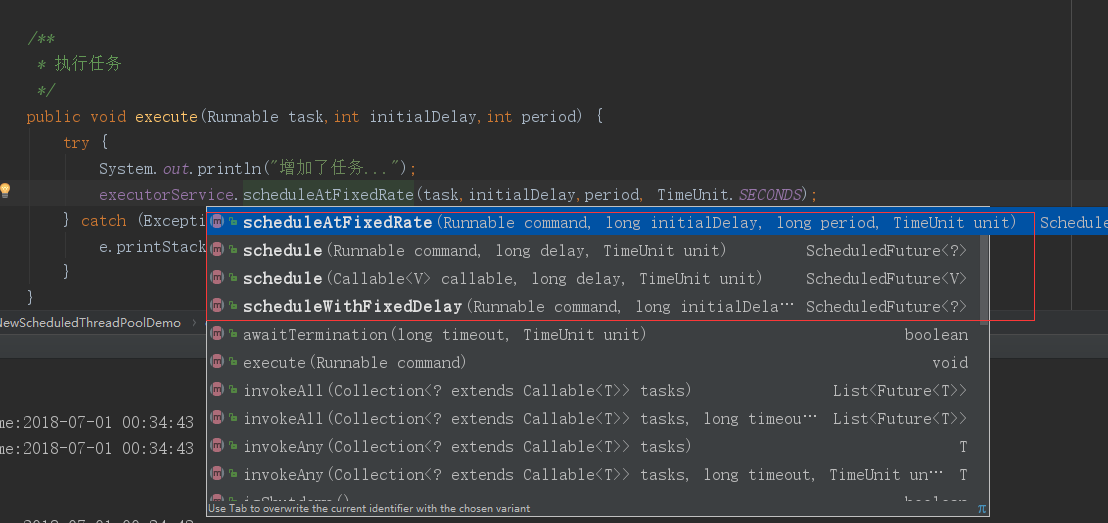
* 同样有两个构造函数，一个是只有一个参数：核心线程数，一个是有两个参数：核心线程数和线程工厂；这里的测试使用了拥有线程工厂的核心线程数，方便观察线程是否是新建；

在使用的过程中，newScheduleThreadPool跟其他线程池不同的地方在于：

1）executorService = Executors.newScheduledThreadPool(core\_thread\_size);

这里的executorService是ScheduledExecutorService,方便使用定时相关的方法，而不是单纯的execute方法；

2）ScheduledExecutorService有很多定时周期执行的方法，设置任务的周期时间，起始延时时间等；



>>根据测试结果,可以得出newScheduleThreadPool的执行逻辑，这里时使用了

executorService.scheduleAtFixedRate(task,initialDelay,period, TimeUnit.SECONDS);方法来进行测试，

所以这里的逻辑讲解是对scheduleAtFixedRate方法的讲解，逻辑如下：

核心线程数定义了线程池的线程数量，任务再多，但线程数不能超过设置的核心线程数，跟newSingleThreadPool一样，没有最大线程数的概念。场景举例：核心线程数设置了2，按顺序建立了5个任务，任务名为：任务一、任务二、任务三、任务四、任务五，每个任务起始延迟2秒，周期执行5秒。那么首先，系统会建立两个线程（核心线程数为2，线程数最多也只能建立2），这两个线程会先安排任务一和任务二进行执行，起始延迟2秒，执行任务一和二的内容，执行完毕后，接着任务一和任务二进入挂起状态，转向执行任务三和任务四，这里任务三和任务四的起始起始延迟时间不是2秒，而是2秒减去任务三和四距离任务execute那一刻的时间，任务五执行的时候起始延迟时间也是如此计算，等待任务三和任务四执行完成后，任务三和任务四同样被挂起，线程转向找寻进行任务五和任务一，任务一原本如果是进程一执行，那么这时还是进程一执行任务一，而进程二执行任务五。这五个任务就是按照以上所说的进行轮转执行。

**1.4.1newScheduleThreadPool和timer的区别（待探讨）**

**1.5 newWorkStealingPool**

**注：newWorkStealingPool在jdk8才有的并发线程池**

测试demo：NewWorkStealingPoolDemoTest

* 同样有两个构造函数，一个是没有参数：核心线程数，一个是有一个参数：parallelism；这里的测试会对两个构造函数都进行测试；

>>根据测试的结果，配合网上论坛https://segmentfault.com/a/1190000011120556的讲解，可以得出执行的逻辑：

先讨论newWorkStealingPool不带参数的构造函数，线程池会根据cpu核心数n建立n个任务队列LinkedBlockingDeque,这n个任务队列里面的任务是并发执行，每个队列都有一个线程监听着，若A线程的队列里如没有了任务，A线程是不会去B线程负责的队列里面寻找，而是会阻塞等待A线程负责的队列的任务到来。因为任务所耗的时长不同，所以多个任务的时候是无法保证顺序执行，有可能任务1还在执行的时候，任务12也已经开始执行。

场景讲解：

例如电脑的核心数为6，那么newWorkStealingPool会建立6个任务队列，而这时候，我有20个任务要执行，那么此时，这20任务会先加到某个等待队列里面，接着从这等待队列中拿出6个任务给到这6个任务队列中，剩下的14个任务队列并没有继续分配，而是等待某个队列的任务执行完成，再拿出一个任务分配给该已经执行原本任务的队列。例如线程A已经执行完成了它负责队列里面的任务，那么此时线程池会从等待队列中顺序调度出一个任务给到线程A负责的队列。

而带有parallelism参数构造函数的情况，parallelism指定了创建多少个并发任务的队列，即使parallelism大于了cpu的核心数，系统仍然会创建parallelism数目的队列，并且这些队列的任务都并发执行。

**待解决问题：**

* 守护线程；
* 回想短信网关连接，模拟实现网关监控的守护线程；
* 查看多线程资料，了解yield等方法的使用；
* 四种线程池各自的优点和缺点；