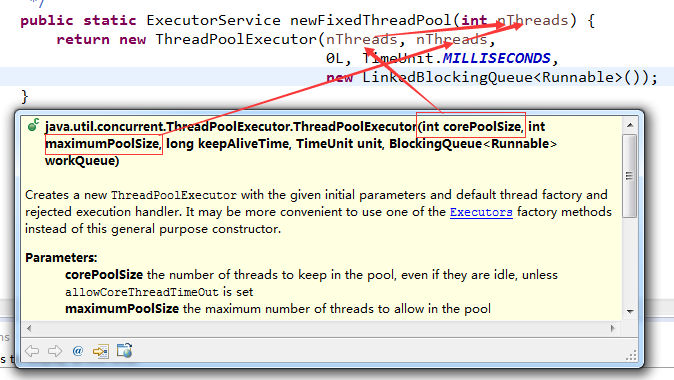
# newFixedThreadPool

代码

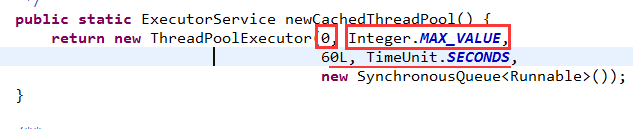
ExecutorService newFixedThreadPool = Executors.newFixedThreadPool(10);



LinkedBlockingQueue 线程暂缓到无界队列中

# newCachedThreadPool

ExecutorService newCachedThreadPool = Executors.newCachedThreadPool();

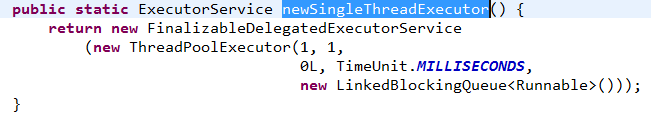


SynchronousQueue 同步队列。来一个任务就直接创建线程(当前没有空闲线程)

。

# newSingleThreadExecutor

ExecutorService newSingleThreadExecutor = Executors.newSingleThreadExecutor();

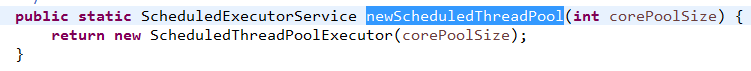


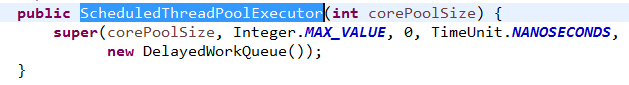
以上三者的实现基本是差不多的，只是 new ThreadPoolExecutor

对象的时候构造函数里的参数不太一样。

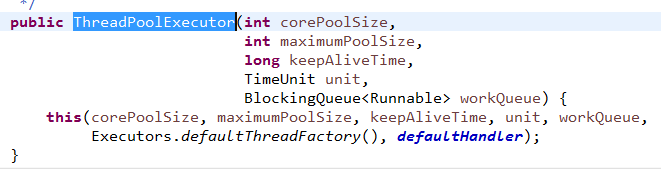
# newScheduledThreadPool

ScheduledExecutorService newScheduledThreadPool = Executors.newScheduledThreadPool(10);





super 其实还是 ThreadPoolExecutor ，但是使用了 DelayedWorkQueue ，带有延时的队列。所以可以像定时器般表现。



四种行为的底层的表现形式还是差不多的。只是传入的参数不一致，导致结构有所不同，所以外层表现出特定的功能。

**public** ThreadPoolExecutor(**int** corePoolSize, //1,核心线程数

**int** maximumPoolSize, //2,最大线程数

**long** keepAliveTime, //3,线程保持活跃

TimeUnit unit, //4,时间单位

BlockingQueue<Runnable> workQueue, //5,阻塞队列，放置等待的线程

ThreadFactory threadFactory,//6,

RejectedExecutionHandler handler //7,)

如果真实情况不符合以上四种行为，可以自定义 ThreadPoolExecutor

1,核心线程数，线程池初始化的时候就初始化的线程数量

5,

6,

7, 处理在执行被阻塞时使用的处理程序,因为线程的界限和队列容量满了

即缓存的线程不应该让其核心代码执行。RejectedExecutionHandler 就是做这种事情。