### 线程池全面解析

# 线程池概念

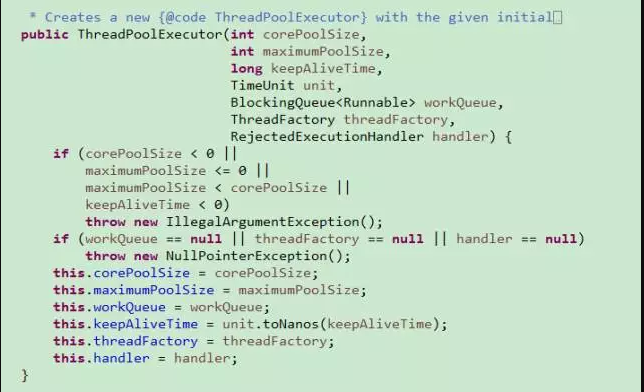
很简单，简单看名字就知道是装有线程的池子，我们可以把要执行的多线程交给线程池来处理，和连接池的概念一样，通过维护一定数量的线程池来达到多个线程的复用。

# **线程池的好处**

我们知道不用线程池的话，每个线程都要通过new Thread(xxRunnable).start()的方式来创建并运行一个线程，线程少的话这不会是问题，而真实环境可能会开启多个线程让系统和程序达到最佳效率，当线程数达到一定数量就会耗尽系统的CPU和内存资源，也会造成GC频繁收集和停顿，因为每次创建和销毁一个线程都是要消耗系统资源的，如果为每个任务都创建线程这无疑是一个很大的性能瓶颈。所以，线程池中的线程复用极大节省了系统资源，当线程一段时间不再有任务处理时它也会自动销毁，而不会长驻内存。

# **线程池核心类**

在java.util.concurrent包中我们能找到线程池的定义，其中ThreadPoolExecutor是我们线程池核心类，首先看看线程池类的主要参数有哪些。



* corePoolSize：线程池的核心大小，也可以理解为最小的线程池大小。
* maximumPoolSize：最大线程池大小。
* keepAliveTime：空余线程存活时间，指的是超过corePoolSize的空余线程达到多长时间才进行销毁。
* unit：销毁时间单位。
* workQueue：存储等待执行线程的工作队列。
* threadFactory：创建线程的工厂，一般用默认即可。
* handler：拒绝策略，当工作队列、线程池全已满时如何拒绝新任务，默认抛出异常。

# **线程池工作流程**

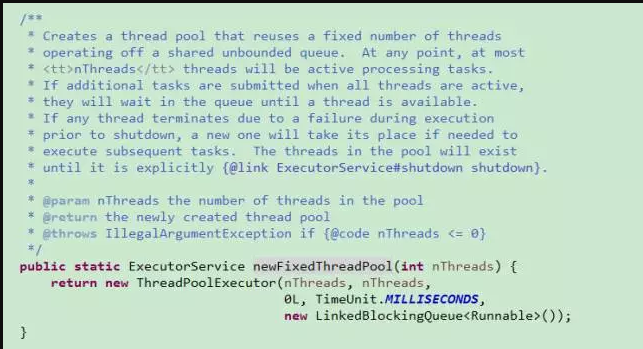
* 如果线程池中的线程小于corePoolSize时就会创建新线程直接执行任务。
* 如果线程池中的线程大于corePoolSize时就会暂时把任务存储到工作队列workQueue中等待执行。
* 如果工作队列workQueue也满时：当线程数小于最大线程池数maximumPoolSize时就会创建新线程来处理，而线程数大于等于最大线程池数maximumPoolSize时就会执行拒绝策略。

# **线程池分类**

Executors是jdk里面提供的创建线程池的工厂类，它默认提供了4种常用的线程池应用，而不必我们去重复构造。

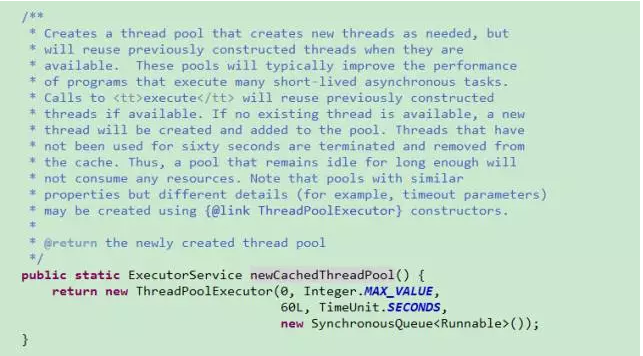
* newFixedThreadPool

固定线程池，核心线程数和最大线程数固定相等，而空闲存活时间为0毫秒，说明此参数也无意义，工作队列为最大为Integer.MAX\_VALUE大小的阻塞队列。当执行任务时，如果线程都很忙，就会丢到工作队列等有空闲线程时再执行，队列满就执行默认的拒绝策略。



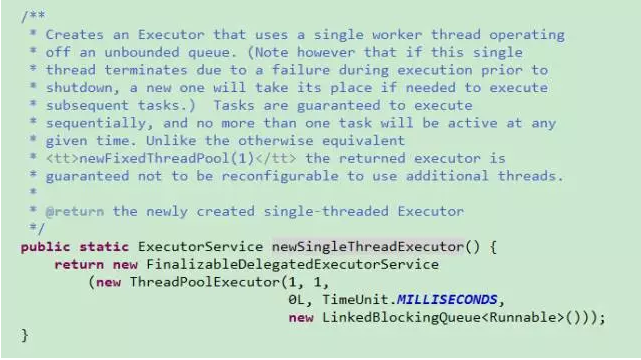
* newCachedThreadPool

带缓冲线程池，从构造看核心线程数为0，最大线程数为Integer最大值大小，超过0个的空闲线程在60秒后销毁，SynchronousQueue这是一个直接提交的队列，意味着每个新任务都会有线程来执行，如果线程池有可用线程则执行任务，没有的话就创建一个来执行，线程池中的线程数不确定，一般建议执行速度较快较小的线程，不然这个最大线程池边界过大容易造成内存溢出。



* newSingleThreadExecutor

单线程线程池，核心线程数和最大线程数均为1，空闲线程存活0毫秒同样无意思，意味着每次只执行一个线程，多余的先存储到工作队列，一个一个执行，保证了线程的顺序执行。



* newScheduledThreadPool

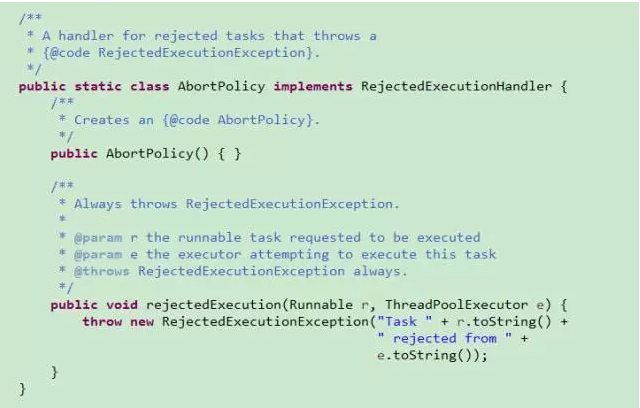
调度线程池，即按一定的周期执行任务，即定时任务，对ThreadPoolExecutor进行了包装而已。

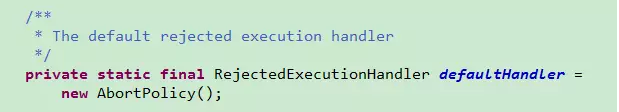


# **拒绝策略**

* AbortPolicy

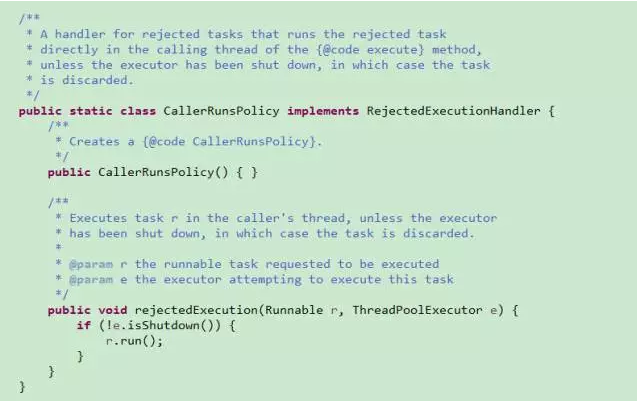
简单粗暴，直接抛出拒绝异常，这也是默认的拒绝策略。





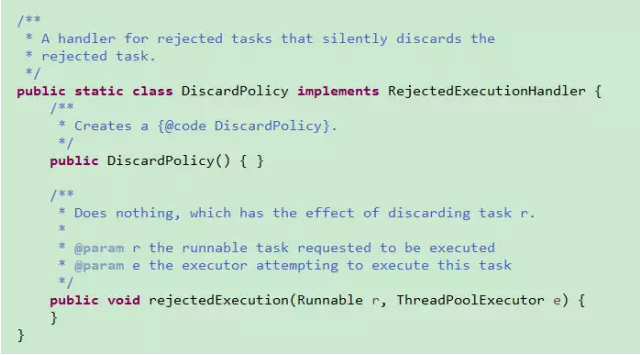
* CallerRunsPolicy

如果线程池未关闭，则会在调用者线程中直接执行新任务，这会导致主线程提交线程性能变慢。



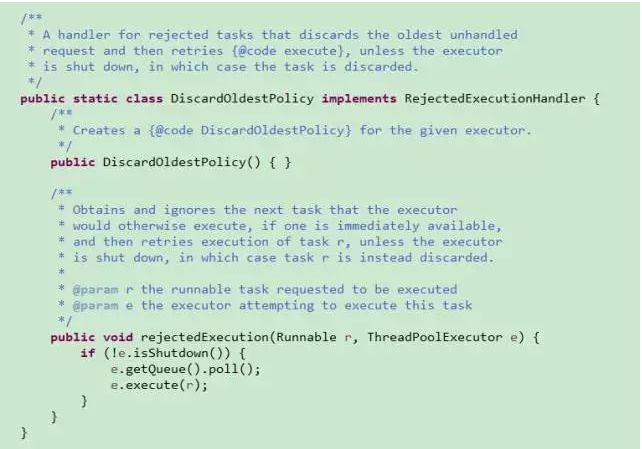
* DiscardPolicy

从方法看没做任务操作，即表示不处理新任务，即丢弃。



* DiscardOldestPolicy

抛弃最老的任务，就是从队列取出最老的任务然后放入新的任务进行执行。



# **如何提交线程**

如可以先随便定义一个固定大小的线程池

ExecutorService es = Executors.newFixedThreadPool(3);

提交一个线程

es.submit(xxRunnble);

es.execute(xxRunnble);

submit和execute分别有什么区别呢？

execute没有返回值，如果不需要知道线程的结果就使用execute方法，性能会好很多。

submit返回一个Future对象，如果想知道线程结果就使用submit提交，而且它能在主线程中通过Future的get方法捕获线程中的异常。

# **如何关闭线程池**

es.shutdown();

不再接受新的任务，之前提交的任务等执行结束再关闭线程池。

es.shutdownNow();

不再接受新的任务，试图停止池中的任务再关闭线程池，返回所有未处理的线程list列表。