1. 线程池的定义

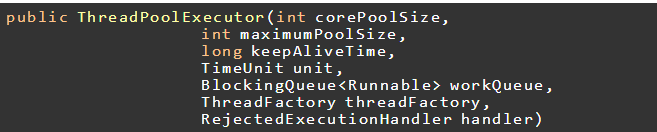
管理一组工作线程的池子。通过线程池复用线程有以下几点优点：

1. 减少资源创建 => 减少内存开销，创建线程占用内存
2. 减少系统开销 => 创建线程需要时间，会延迟处理请求
3. 提高系统稳定性 => 避免无限制创建线程引起OOM
4. Executors创建线程池的方式

根据返回的对象类型创建线程池可分为三类：

1. 创建返回ThreadPoolExecutor对象
2. 创建返回ScheduleThreadPoolExecutor对象
3. 创建返回ForkJoinPool对象
4. ThreadPoolExecutor对象

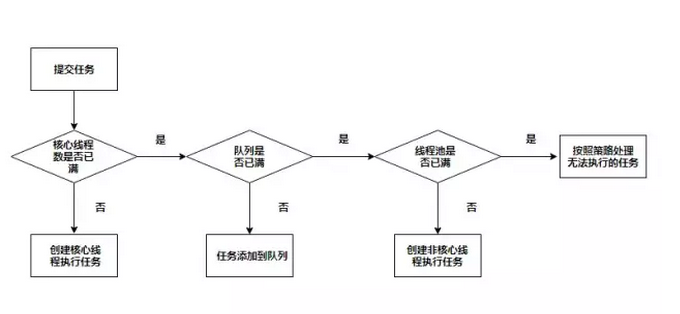
ThreadPoolExecutor的构造函数共有四个，但最终调用的都是同一个：



构造函数参数说明：

1. corePoolSize => 线程池核心线程数量
2. maximumPoolSize => 线程池最大线程数量
3. keepAliveTime => 空闲线程存活时间
4. unit => 时间单位
5. workQueue => 线程池所使用的缓存队列
6. threadFactory => 线程池创建线程使用的工厂
7. handler => 线程池对拒绝任务的处理策略

线程池执行任务逻辑和线程池参数的关系图：



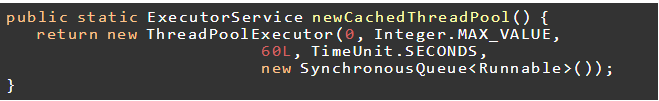
1. 判断核心线程数是否已满，核心线程数大小和corePoolSize参数有关，未满则创建线程执行任务。
2. 若核心线程池已满，则判断队列是否已满，队列大小和workQueue参数有关，若未满则加入队列中。
3. 若队列已满，判断线程池是否已满，线程池和maximumPoolSize参数有关，若未满创建线程执行任务。
4. 若线程池已满，则采用拒绝策略处理无法执行的任务，拒绝策略和handler参数有关。

线程池的拒绝策略：

1. AbortPolicy：丢弃任务并抛出异常
2. DiscardPolicy：丢弃任务，但是不抛出异常
3. DiscardOldestPolicy：丢弃队列最前面的任务，然后重新提交被拒绝的任务。
4. CallerRunsPolicy：由提交任务的线程处理该任务
5. 线程池默认的拒绝策略是AbortPolicy

Executors创建返回ThreadPoolExecutor对象的方法共有三种：

1. newCachedThreadPool方法



CachedThreadPool是一个根据需要创建新线程的线程池

corePoolSize => 0，核心线程池的数量为0

maximumPoolSize => Integer.MAX\_VALUE，可以认为最大线程数是无限的

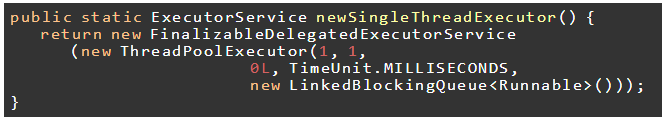
keepAliveTime => 60L

unit => 秒

workQueue => SynchronousQueue，一个不存储元素的阻塞队列

当一个任务提交时，corePoolSize为0不创建核心线程，SynchronousQueue是一个不存储元素的队列，可以理解为队列永远是满的，因此最终会创建非核心线程来执行任务。对于非核心线程空闲60s时将被回收。因为Integer.MAX\_VALUE非常大，可以认为是可以无限创建线程的，在资源有限的情况下容易引起OOM异常

1. newSingleThreadExecutor方法



SingleThreadPoolExecutor是单线程线程池，只有一个核心线程

corePoolSize => 1，核心线程为1

maximumPoolSize =>1，最大线程数为1

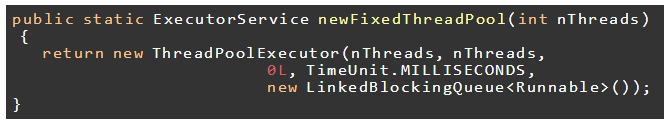
keepAliveTime => 0L

unit => 毫秒

workQueue => LinkedBlockingQueue，基于链表的无界的阻塞队列

当一个任务提交时，首先会创建一个核心线程来执行任务，如果超过核心线程的数量，将会放入队列，因为L;inkedBlockingQueue是长度为Integer.MAX\_VALUE的队列，可以任务是无界队列，因此往队列中可以插入无限多的任务，在资源有限的时候容易引起OOM异常，同时因为无界队列，maximumPoolSize和keepAliveTime参数将无效，压根就不会创建非核心线程。

1. newFixedThreadPool方法



FixedThreadPool是固定核心线程的线程池，固定核心线程数由用户传入

corePoolSize => nThreads，用户传入

maximumPoolSize => nThreads，用户传入

keepAliveTime => 60L

unit => 毫秒

workQueue => LinkedBlockingQueue，基于链表的无界的阻塞队列

它和SiongleThreadExecutor类似，唯一的区别就是核心线程不同，并且由于使用LinkedBlockQueue，在资源有限的时候容易引起OOM异常

1. 总结
2. FixedThreadPool和SingleThreadExecutor => 允许的请求队列长度为Integer.MAX\_VALUE，可能会堆积大量的请求，从而引起OOM异常
3. CachedThreadPool => 允许创建的线程数为Integer.MAX\_VALUE，可能会创建大量的线程，从而引起OOM异常