竞争加剧,markword记录指向mutex互斥锁的指针,竞争线程进入锁的等待队列不消耗CPU 重量级锁 Concurrent HashMap CopyOnWriteArrayList

Java多线程

、 并发容器 ThreadLocal BlockingQueue 手写线程池 使用单个worker线程的Executor corePoolSize和maximumPoolSize被设置为1。 newSingleThreadExecutor = SingleThreadExecutor使用无界队列LinkedBlockingQueue作为线程池的工作队列 如果当前运行的线程数少于core PoolSize(即线程池中无运行的线程),则创建一个新线程来执行任务 corePoolSize和maximumPoolSize都被设置为创建FixedThreadPool时指定的参数nThreads keepAliveTime设置为OL,意味着多余的空闲线程会被立即终止 newFixedThreadPool 如果当前运行的线程数少于core PoolSize ,则创建新线程来执行任务。 FixedThreadPool使用无界队列LinkedBlockingQueue作为线程池的工作队列(队列的容量为Integer.MAX_VALUE) 根据需要创建新线程的线程池 Executors corePoolSize被设置为0,即corePool为空 maximumPoolSize被设置为Integer.MAX_VALUE,即maximumPool是无界的。 newCachedThreadPool keepAliveTime设置为60L,意味着CachedThreadPool中的空闲线程等待新任务的最长时间为60秒 CachedThreadPool使用没有容量的SynchronousQueue作为线程池的工作队列,但CachedThreadPool的maximumPool是无界的。这意味 着,如果主线程提交任务的速度高于maximumPool中线程处理任务的速度时,CachedThreadPool会不断创建新线程。极端情况下, CachedThreadPool会因为创建过多线程而耗尽CPU和内存资源。 DelayQueue是一个无界队列 _.1. 当调用ScheduledThreadPoolExecutor的scheduleAtFixedRate()方法或者scheduleWith-FixedDelay()方法时,会向 newScheduledThreadPoolScheduledThreadPoolExecutor的DelayQueue添加一个实现了RunnableScheduledFutur接口的ScheduledFutureTask。 2. 线程池中的线程从DelayQueue中获取ScheduledFutureTask, 然后执行任务。 一 核心线程数,线程数定义了最小可以同时运行的线程数量。 · corePoolSize — —— 线程池中允许存在的工作线程的最大数量 maximumPoolSize -线程池中的线程数量大于 core PoolSize 的时候,如果这时没有新的任务提交,核心线程外的线程不会立即销毁,而是会等待,直到等待的时间超 · KeepAliveTime - 过了 keepAliveTime才会被回收销毁; keepAliveTime 参数的时间单位。 当新任务来的时候会先判断当前运行的线程数量是否达到核心线程数,如果达到的话,任务就会被存放在队列中。 ArrayBlockingQueue:是一个基于数组结构的有界阻塞队列,此队列按FIFO(先进先出)原则对元素进行排序。 LinkedBlockingQueue:一个基于链表结构的无界阻塞队列,此队列按FIFO排序元素,吞吐量通常要高于ArrayBlockingQueue。静态工厂方法 Executors.newFixedThreadPool()使用了这个队列。 ThreadPoolExecut or SynchronousQueue:一个不存储元素的阻塞队列。每个插入操作必须等到另一个线程调用移除操作,否则插入操作一直处于阻塞状态,吞吐量 通常要高于Linked-BlockingQueue,静态工厂方法Executors.newCachedThreadPool使用了这个队列。 PriorityBlockingQueue:一个具有优先级的无限阻塞队列。 · 不使用默认的Default ThreadFact ory ———— 名字没有辨识度 都是 Thread-Pool-number threadFactory = 一 用于设置创建线程的工厂,可以通过线程工厂给每个创建出来的线程设置更有意义的名字。 线程池任务队列超过 maxinumPoolSize 之后的拒绝策略 - ThreadPoolExecutor.AbortPolicy: 抛出 RejectedExecutionException来拒绝新任务的处理。 - ThreadPoolExecutor.CallerRunsPolicy:调用执行自己的线程运行任务.不处理新任务,但是这种策略会降低对于新任务提交速度,影响程序的 线程池 整体性能。另外,这个策略喜欢增加队列容量。如果您的应用程序可以承受此延迟并且你不能任意丢弃任何一个任务请求的话,你可以选择这个策 - ThreadPoolExecutor.DiscardPolicy:不处理新任务,直接丢弃掉。 - ThreadPoolExecutor.DiscardOldestPolicy: 此策略将丢弃最早的未处理的任务请求。 1)线程池判断核心线程池里的线程是否都在执行任务。如果不是,则创建一个新的工作线程来执行任务。如果核心线程池里的线程都在执行任 - 1. 判断corePoolSize -线程池的处理流程 3)线程池判断线程池的线程是否都处于工作状态。如果没有,则创建一个新的工作线程来执行任务。如果已经满了,则交给饱和策略来处理这个 Ϲ 3. 判断maximumPoolSize 💳 ー execute()方法用于提交不需要返回值的任务 , ーーーー execute()方法用于提交不需要返回值的任务 , 所以无法判断任务是否被线程池执行成功。execute()方法输入的任务是一个Runnable类的实例。 ・向线程池提交任务 线程池的实现原理 - 线程池会返回一个future类型的对象,通过这个future对象可以判断任务是否执行成功,并且可以通过future的get()方法来获取返回值, submit ()方法用于提交需要返回值的任务。 -- get()方法会阻塞当前线程直到任务完成,而使用get (long timeout , TimeUnit unit) 方法则会阻塞当前线程一段时间后立即返回 , 这时候有可 能任务没有执行完。 一一 首先将线程池的状态设置成STOP,然后尝试停止所有的正在执行或暂停任务的线程,并返回等待执行任务的列表 ── 将线程池的状态设置成SHUTDOWN状态,然后中断所有没有正在执行任务的线程。 关闭线程池 - 只要调用了这两个关闭方法中的任意一个, is Shut down方法就会返回true。 - 当所有的任务都已关闭后,才表示线程池关闭成功,这时调用isTerminaed方法会返回true。 - 至于应该调用哪一种方法来关闭线程池,应该由提交到线程池的任务特性决定,通常调用shutdown方法来关闭线程池,如果任务不一定要执行 完,则可以调用shut downNow方法 《阿里巴巴Java开发手册》中强制线程池不允许使用 Executors 去创建,而是通过 ThreadPoolExecutor 的方式,这样的处理方式让写的同学更 加明确线程池的运行规则,规避资源耗尽的风险 Executors 各个方法的弊端: - newFixedThreadPool 和 newSingleThreadExecutor: 阿里巴巴规约 主要问题是堆积的请求处理队列可能会耗费非常大的内存,甚至 OOM。 - newCachedThreadPool和 newScheduledThreadPool: 主要问题是线程数最大数是 Integer.MAX_VALUE, 可能会创建数量非常多的线程, 甚至 OOM。 的超纖弱地是透微 - RUNNING: 这是最正常的状态,接受新的任务,处理等待队列中的任务。 - SHUTDOWN: 不接受新的任务提交,但是会继续处理等待队列中的任务。 线程池都有哪些状态? —— - STOP:不接受新的任务提交,不再处理等待队列中的任务,中断正在执行任务的线程。 - TIDYING: 所有的任务都销毁了, workCount 为 0, 线程池的状态在转换为 TIDYING 状态时, 会执行钩子方法 terminated()。 - TERMINATED: terminated()方法结束后,线程池的状态就会变成这个。 Executor vs ExecutorService 1. ExecutorService 接口继承了 Executor 接口,是 Executor 的子接口 · 2. Executor接口定义了 execute()方法用来接收一个Runnable接口的对象,而 ExecutorService接口中的 submit()方法可以接受Runnable和 Executor VS ExecutorService VS Executors Callable接口的对象。 3. Executor 中的 execute() 方法不返回任何结果,而 ExecutorService 中的 submit()方法可以通过一个 Future 对象返回运算结果。4. 除了允许客户端提交一个任务,ExecutorService 还提供用来控制线程池的方法。比如:调用 shut Down() 方法终止线程池。 · Executors 类提供工厂方法用来创建不同类型的线程池。 - - AtomicMarkableReference(通过引入一个 boolean来反映中间有没有变过), 解决ABA问题的原子类 -- AtomicStampedReference (通过引入一个 int 来累加来反映中间有没有变过) 原子类 atomic原理: 类似于分段锁,将值放到数组里,例如四个位置,分成了4个锁,不同的线程去锁不同的位置,这样同时就可以有4个线程的递增可以执行,最后把这些数 - 组内的数求和得到的就是最终的结果, 在高并发的情况下有用, 因为总的递增次数是一样的但是并行度变成了原来数组长度的倍数, 效率在高并发的情况下更好 Just In Time Compiler 即时编译器 - JVM是解释执行, 来一条指令, 解释成机器码执行 编译器 对于热点代码,解释成机器码,下次再遇到就不需要解释,直接执行即可,增加了效率 https://blog.csdn.net/qq_35190492/article/details/104691668? task-blog-2~all~top_positive~default-2-104691668.pc_search_download_positive&utm_term=AQS&spm=1018.2226.3001.4187