public class ThreadPoolExecutor extends AbstractExecutorService

AbstractExecutorService是一个实现了ExecutorService接口中部分方法的抽象类，ExecutorService接口是对Executor接口的拓展。



# 接口说明

An [ExecutorService](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html) that executes each submitted task using one of possibly several pooled threads, normally configured using [Executors](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html) factory methods.

ThreadPoolExecutor实现了ExecutorService接口，使用线程池中的任一线程执行提交的任务，通常使用Executors类中的工厂方法配置并获取ThreadPoolExecutor类的实例。

Thread pools address two different problems: they usually provide improved performance when executing large numbers of asynchronous tasks, due to reduced per-task invocation overhead, and they provide a means of bounding and managing the resources, including threads, consumed when executing a collection of tasks. Each ThreadPoolExecutor also maintains some basic statistics, such as the number of completed tasks.

ThreadPoolExecutor的主要作用：

1. 当需要执行大量异步任务时，通过使用ThreadPoolExecutor能够减少每个任务的调用开销，从而提高系统性能。
2. 提供了一种方法来约束和管理执行任务集合时消耗的资源（包括线程）。
3. 维护一些基本统计信息，例如已完成任务的数量。

To be useful across a wide range of contexts, this class provides many adjustable parameters and extensibility hooks. However, programmers are urged to use the more convenient [Executors](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html) factory methods [Executors.newCachedThreadPool()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newCachedThreadPool--) (unbounded thread pool, with automatic thread reclamation), [Executors.newFixedThreadPool(int)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool-int-) (fixed size thread pool) and [Executors.newSingleThreadExecutor()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newSingleThreadExecutor--) (single background thread), that preconfigure settings for the most common usage scenarios. Otherwise, use the following guide when manually configuring and tuning this class:

为了支持更多的使用场景，类中提供了很多可以调整的参数和可以拓展的钩子。但是鼓励程序员通过一种更简单的方式使用线程池，即通过Executors类提供的一些工厂方法，如：

[Executors.newCachedThreadPool()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newCachedThreadPool--) – 线程池大小无限制，自动回收线程。

[Executors.newFixedThreadPool(int)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool-int-) – 线程池大小固定。

[Executors.newSingleThreadExecutor()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newSingleThreadExecutor--) – 线程池中只有一个线程

这些方法针对一些常用配置做了预先配置。如果你打算自己手动配置和调整ThreadPoolExecutor类时，建议先阅读一下下面的注意事项：

## Core and maximum pool sizes

A ThreadPoolExecutor will automatically adjust the pool size (see [getPoolSize()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getPoolSize--)) according to the bounds set by corePoolSize (see [getCorePoolSize()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getCorePoolSize--)) and maximumPoolSize (see [getMaximumPoolSize()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getMaximumPoolSize--)). When a new task is submitted in method [execute(Runnable)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#execute-java.lang.Runnable-), and fewer than corePoolSize threads are running, a new thread is created to handle the request, even if other worker threads are idle. If there are more than corePoolSize but less than maximumPoolSize threads running, a new thread will be created only if the queue is full. By setting corePoolSize and maximumPoolSize the same, you create a fixed-size thread pool. By setting maximumPoolSize to an essentially unbounded value such as Integer.MAX\_VALUE, you allow the pool to accommodate an arbitrary number of concurrent tasks. Most typically, core and maximum pool sizes are set only upon construction, but they may also be changed dynamically using [setCorePoolSize(int)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setCorePoolSize-int-) and [setMaximumPoolSize(int)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setMaximumPoolSize-int-).

corePoolSize和maximumPoolSize ⭐

ThreadPoolExecutor将根据corePoolSize和maximumPoolSize自动调整线程池中线程的数量。当客户端通过execute方法向线程池提交了一个任务，此时线程池中的线程数如果少于corePoolSize，即使此时次线程池中存在空闲的线程，也将创建一个新的线程执行该任务；如果此时线程池中的线程数多于corePoolSize但不多于maximumPoolSize,只有队列满了才会创建新的线程。如果将corePoolSize和maximumPoolSize的值设置为相等，你相当于创建了一个大小会一直改变的线程池。通过将maximumPoolSize设置为一个本质上不受限制的值（例如Integer.MAX\_VALUE），可以允许池容纳任意数量的并发任务。一般核情况下，corePoolSize和maximumPoolSize仅在构造时设置，但是也可以使用setCorePoolSize（int）和setMaximumPoolSize（int）动态更改。

说明：

corePoolSize – 核心池大小，表示线程池中始终维持的线程数。

maximumPoolSize – 池的最大值，表示线程池中最多存在多少个线程。

## On-demand construction

By default, even core threads are initially created and started only when new tasks arrive, but this can be overridden dynamically using method [prestartCoreThread()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#prestartCoreThread--) or [prestartAllCoreThreads()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#prestartAllCoreThreads--). You probably want to prestart threads if you construct the pool with a non-empty queue.

**按需配置**

默认情况下，线程池中的核心线程只有当新任务提交才会创建和启动，但可以通过prestartCoreThread（）或prestartAllCoreThreads（）方法动态修改。如果要使用非空队列构造池，则可能要预启动线程。

## Creating new threads

New threads are created using a [ThreadFactory](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadFactory.html). If not otherwise specified, a [Executors.defaultThreadFactory()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#defaultThreadFactory--) is used, that creates threads to all be in the same [ThreadGroup](https://docs.oracle.com/javase/8/docs/api/java/lang/ThreadGroup.html) and with the same NORM\_PRIORITY priority and non-daemon status. By supplying a different ThreadFactory, you can alter the thread's name, thread group, priority, daemon status, etc. If a ThreadFactory fails to create a thread when asked by returning null from newThread, the executor will continue, but might not be able to execute any tasks. Threads should possess the "modifyThread" RuntimePermission. If worker threads or other threads using the pool do not possess this permission, service may be degraded: configuration changes may not take effect in a timely manner, and a shutdown pool may remain in a state in which termination is possible but not completed.

**创建新线程**

使用ThreadFactory创建新线程。如果未另行指定，则使用Executors.defaultThreadFactory（），该线程创建的线程全部位于同一ThreadGroup中，并且具有相同的NORM\_PRIORITY优先级和非守护程序状态。通过提供不同的ThreadFactory，可以更改线程的名称，线程组，优先级，守护程序状态等。如果ThreadFactory在通过从newThread返回null返回要求时未能创建线程，执行器将继续执行，但可能无法执行执行任何任务。线程应具有“ modifyThread” RuntimePermission。如果使用该池的工作线程或其他线程不具有此权限，则服务可能会降级：配置更改可能不会及时生效，并且关闭池可能保持在可能终止但未完成的状态。

## Keep-alive times

If the pool currently has more than corePoolSize threads, excess threads will be terminated if they have been idle for more than the keepAliveTime (see [getKeepAliveTime(TimeUnit)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getKeepAliveTime-java.util.concurrent.TimeUnit-)). This provides a means of reducing resource consumption when the pool is not being actively used. If the pool becomes more active later, new threads will be constructed. This parameter can also be changed dynamically using method [setKeepAliveTime(long, TimeUnit)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setKeepAliveTime-long-java.util.concurrent.TimeUnit-). Using a value of Long.MAX\_VALUE [TimeUnit.NANOSECONDS](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html#NANOSECONDS) effectively disables idle threads from ever terminating prior to shut down. By default, the keep-alive policy applies only when there are more than corePoolSize threads. But method [allowCoreThreadTimeOut(boolean)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#allowCoreThreadTimeOut-boolean-) can be used to apply this time-out policy to core threads as well, so long as the keepAliveTime value is non-zero.

**线程存活时间**

如果当前池中的线程数超过corePoolSize，则多余的线程将在空闲时间超过keepAliveTime时终止（请参阅getKeepAliveTime（TimeUnit））。当未积极使用池时，这提供了减少资源消耗的方法。如果池稍后变得更加频繁，则将构建新线程。还可以使用setKeepAliveTime（long，TimeUnit）方法动态更改其参数。使用Long.MAX\_VALUE TimeUnit.NANOSECONDS值可以有效地使空闲线程永远不会在关闭之前终止。默认情况下，仅当池中线程数超过corePoolSize时，保持活动策略才适用。但是，只要keepAliveTime值不为零，方法allowCoreThreadTimeOut（boolean）还可用于将此超时策略应用于核心线程。

## Queuing

Any [BlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html) may be used to transfer and hold submitted tasks. The use of this queue interacts with pool sizing:

* If fewer than corePoolSize threads are running, the Executor always prefers adding a new thread rather than queuing.
* If corePoolSize or more threads are running, the Executor always prefers queuing a request rather than adding a new thread.
* If a request cannot be queued, a new thread is created unless this would exceed maximumPoolSize, in which case, the task will be rejected.

**队列**

任何BlockingQueue（阻塞队列）均可用于传输和保留提交的任务。队列的使用与池的大小相关联：

* 如果运行的线程数少于corePoolSize，则执行程序总是喜欢添加新线程，而不是排队。
* 如果正在运行corePoolSize或更多线程，则执行程序总是更喜欢对请求进行排队，而不是添加新线程。
* 如果无法将请求排队，则将创建一个新线程，除非池中的线程数已经达到maximumPoolSize，在这种情况下，该任务将被拒绝。

There are three general strategies for queuing:

这有3种比较一般的排队策略：

*1、Direct handoffs.* A good default choice for a work queue is a [SynchronousQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html) that hands off tasks to threads without otherwise holding them. Here, an attempt to queue a task will fail if no threads are immediately available to run it, so a new thread will be constructed. This policy avoids lockups when handling sets of requests that might have internal dependencies. Direct handoffs generally require unbounded maximumPoolSizes to avoid rejection of new submitted tasks. This in turn admits the possibility of unbounded thread growth when commands continue to arrive on average faster than they can be processed.

直接执行。对于工作队列，一个很好的默认选择是SynchronousQueue，它可以直接将任务移交给线程，而不必另外保留它们。在这里，如果没有立即可用的线程来运行任务，则尝试将任务排队将会失败，因此将构造一个新线程。在处理可能具有内部依赖性的请求集时，此策略避免了锁。直接切换通常需要无限制的maximumPoolSizes，以避免拒绝新提交的任务。因此，如果任务执行的速度低于任务提交的速度，池中线程的数量将一直增长。

*2、Unbounded queues.* Using an unbounded queue (for example a [LinkedBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html) without a predefined capacity) will cause new tasks to wait in the queue when all corePoolSize threads are busy. Thus, no more than corePoolSize threads will ever be created. (And the value of the maximumPoolSize therefore doesn't have any effect.) This may be appropriate when each task is completely independent of others, so tasks cannot affect each others execution; for example, in a web page server. While this style of queuing can be useful in smoothing out transient bursts of requests, it admits the possibility of unbounded work queue growth when commands continue to arrive on average faster than they can be processed.

无限队列。采用此策略时，如果所有核心线程都处于忙碌状态，任务被保存到等待队列。因此，池中的线程数不会超过corePoolSize（因此，maximumPoolSize的值没有任何作用）。当每个任务完全独立于其他任务时，本策略时合适的，因为任务不会影响彼此的执行。例如，在网页服务器中。尽管这种排队方式对于消除短暂的请求突发很有用，但是一旦请求处理的速率低于请求提交的速率，工作队列将无限制地增长。

*3、Bounded queues.* A bounded queue (for example, an [ArrayBlockingQueue](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html)) helps prevent resource exhaustion when used with finite maximumPoolSizes, but can be more difficult to tune and control. Queue sizes and maximum pool sizes may be traded off for each other: Using large queues and small pools minimizes CPU usage, OS resources, and context-switching overhead, but can lead to artificially low throughput. If tasks frequently block (for example if they are I/O bound), a system may be able to schedule time for more threads than you otherwise allow. Use of small queues generally requires larger pool sizes, which keeps CPUs busier but may encounter unacceptable scheduling overhead, which also decreases throughput.

有界队列。使用有界队列能够防止资源被耗尽，但是也更加难以调整和控制。queueSize和maximumPoolSize相互权衡：使用大队列和小池可最大程度地减少CPU使用率、降低操作系统资源消耗以及降低上下文切换带来的开销，但可能导致吞吐量下降。如果任务频繁阻塞（例如，如果它们受I / O限制），则系统可能能够安排比您原先允许的线程更多的时间。使用小队列通常需要更大的池大小，这会将提高CPU使用率，但可能会带来不可接受的调度开销，这也会降低吞吐量。

## Rejected tasks

New tasks submitted in method [execute(Runnable)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#execute-java.lang.Runnable-) will be rejected when the Executor has been shut down, and also when the Executor uses finite bounds for both maximum threads and work queue capacity, and is saturated. In either case, the execute method invokes the [RejectedExecutionHandler.rejectedExecution(Runnable, ThreadPoolExecutor)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html#rejectedExecution-java.lang.Runnable-java.util.concurrent.ThreadPoolExecutor-)method of its [RejectedExecutionHandler](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html). Four predefined handler policies are provided:

被拒绝任务

在执行器被关闭或者是上文提到的某些情况下向执行器提交任务，任务会被拒绝执行。以下提供了四个预定义的处理策略：

1. In the default [ThreadPoolExecutor.AbortPolicy](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.AbortPolicy.html), the handler throws a runtime [RejectedExecutionException](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionException.html) upon rejection.

在默认的ThreadPoolExecutor.AbortPolicy（拒绝策略）中，处理程序在拒绝时会抛出RejectedExecutionException运行时异常。

1. In [ThreadPoolExecutor.CallerRunsPolicy](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.CallerRunsPolicy.html), the thread that invokes execute itself runs the task. This provides a simple feedback control mechanism that will slow down the rate that new tasks are submitted.

在ThreadPoolExecutor.CallerRunsPolicy中，客户端线程自己执行提交的任务。这提供了一种简单的反馈控制机制，将降低新任务提交的速度。

3、In [ThreadPoolExecutor.DiscardPolicy](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.DiscardPolicy.html), a task that cannot be executed is simply dropped.

在ThreadPoolExecutor.DiscardPolicy中，简单地删除了无法执行的任务。

4、In [ThreadPoolExecutor.DiscardOldestPolicy](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.DiscardOldestPolicy.html), if the executor is not shut down, the task at the head of the work queue is dropped, and then execution is retried (which can fail again, causing this to be repeated.)

在ThreadPoolExecutor.DiscardOldestPolicy中，如果未关闭执行程序，则将丢弃工作队列开头的任务（最老的一个任务），然后重试执行（该操作可能再次失败，导致重复执行此操作）。

It is possible to define and use other kinds of [RejectedExecutionHandler](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html) classes. Doing so requires some care especially when policies are designed to work only under particular capacity or queuing policies.

可以定义和使用其他种类的RejectedExecutionHandler类。这样做需要格外小心，尤其是在设计策略仅在特定容量或排队策略下工作时。

## Hook methods

This class provides protected overridable [beforeExecute(Thread, Runnable)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#beforeExecute-java.lang.Thread-java.lang.Runnable-) and [afterExecute(Runnable, Throwable)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#afterExecute-java.lang.Runnable-java.lang.Throwable-)methods that are called before and after execution of each task. These can be used to manipulate the execution environment; for example, reinitializing ThreadLocals, gathering statistics, or adding log entries. Additionally, method [terminated()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#terminated--) can be overridden to perform any special processing that needs to be done once the Executor has fully terminated.

If hook or callback methods throw exceptions, internal worker threads may in turn fail and abruptly terminate.

钩子方法

此类提供受保护的可重写的beforeExecute（Thread，Runnable）和afterExecute（Runnable，Throwable）方法，这些方法在执行每个任务之前和之后调用。这些可以用来操纵执行环境。例如，重新初始化ThreadLocals，收集统计信息或添加日志条目。此外，一旦执行程序完全终止，可以重写方法Terminate（）以执行需要执行的任何特殊处理。

如果钩子或回调方法引发异常，内部工作线程可能进而失败并突然终止。

## Finalization

A pool that is no longer referenced in a program AND has no remaining threads will be shutdown automatically. If you would like to ensure that unreferenced pools are reclaimed even if users forget to call [shutdown()](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#shutdown--), then you must arrange that unused threads eventually die, by setting appropriate keep-alive times, using a lower bound of zero core threads and/or setting [allowCoreThreadTimeOut(boolean)](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#allowCoreThreadTimeOut-boolean-).

终结

程序中不再引用且没有剩余线程的池将自动关闭。如果即使在用户忘记调用shutdown（）的情况下仍要确保收回未引用的池，则必须使用零核心线程的下限和/或通过设置适当的保持存活时间，安排未使用的线程最终死掉设置allowCoreThreadTimeOut（boolean）。

# 内部类说明

|  |  |
| --- | --- |
| **Modifier and Type** | **Class and Description** |
| static class | [**ThreadPoolExecutor.AbortPolicy**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.AbortPolicy.html)  A handler for rejected tasks that throws a RejectedExecutionException.  抛出RejectedExecutionException的被拒绝任务的处理程序。 |
| static class | [**ThreadPoolExecutor.CallerRunsPolicy**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.CallerRunsPolicy.html)  A handler for rejected tasks that runs the rejected task directly in the calling thread of the execute method, unless the executor has been shut down, in which case the task is discarded.  拒绝任务的处理程序，它直接在execute方法的调用线程中运行拒绝任务，除非执行器已关闭，在这种情况下，该任务将被丢弃。 |
| static class | [**ThreadPoolExecutor.DiscardOldestPolicy**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.DiscardOldestPolicy.html)  A handler for rejected tasks that discards the oldest unhandled request and then retries execute, unless the executor is shut down, in which case the task is discarded.  拒绝任务的处理程序将丢弃最早的未处理请求，然后重试执行，除非执行器被关闭，在这种情况下，该任务将被丢弃。 |
| static class | [**ThreadPoolExecutor.DiscardPolicy**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.DiscardPolicy.html)  A handler for rejected tasks that silently discards the rejected task.  拒绝任务的处理程序，静默丢弃被拒绝的任务。 |

# 方法说明

## 构造方法说明

|  |
| --- |
| **Constructor and Description** |
| [**ThreadPoolExecutor**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#ThreadPoolExecutor-int-int-long-java.util.concurrent.TimeUnit-java.util.concurrent.BlockingQueue-)(int corePoolSize,int maximumPoolSize, long keepAliveTime, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit,[**BlockingQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> workQueue)  Creates a new ThreadPoolExecutor with the given initial parameters and default thread factory and rejected execution handler.  使用给定的初始参数和默认线程工厂以及默认的拒绝执行处理程序创建一个新的ThreadPoolExecutor。 |
| [**ThreadPoolExecutor**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#ThreadPoolExecutor-int-int-long-java.util.concurrent.TimeUnit-java.util.concurrent.BlockingQueue-java.util.concurrent.RejectedExecutionHandler-)(int corePoolSize,int maximumPoolSize, long keepAliveTime, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit,[**BlockingQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> workQueue, [**RejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html) handler)  Creates a new ThreadPoolExecutor with the given initial parameters and default thread factory.  使用给定的初始参数和默认线程工厂创建一个新的ThreadPoolExecutor。 |
| [**ThreadPoolExecutor**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#ThreadPoolExecutor-int-int-long-java.util.concurrent.TimeUnit-java.util.concurrent.BlockingQueue-java.util.concurrent.ThreadFactory-)(int corePoolSize,int maximumPoolSize, long keepAliveTime, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit,[**BlockingQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> workQueue, [**ThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadFactory.html) threadFactory)  Creates a new ThreadPoolExecutor with the given initial parameters and default rejected execution handler.  使用给定的初始参数和默认的拒绝执行处理程序创建一个新的ThreadPoolExecutor。 |
| [**ThreadPoolExecutor**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#ThreadPoolExecutor-int-int-long-java.util.concurrent.TimeUnit-java.util.concurrent.BlockingQueue-java.util.concurrent.ThreadFactory-java.util.concurrent.RejectedExecutionHandler-)(int corePoolSize,int maximumPoolSize, long keepAliveTime, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit,[**BlockingQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> workQueue, [**ThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadFactory.html) threadFactory, [**RejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html) handler)  Creates a new ThreadPoolExecutor with the given initial parameters.  使用给定的初始参数创建一个新的ThreadPoolExecutor。 |

## 其他方法说明

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| protected void | [**afterExecute**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#afterExecute-java.lang.Runnable-java.lang.Throwable-)([**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) r, [**Throwable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html) t)  Method invoked upon completion of execution of the given Runnable.  给定Runnable执行完成时调用的方法。 |
| void | [**allowCoreThreadTimeOut**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#allowCoreThreadTimeOut-boolean-)(boolean value)  Sets the policy governing whether core threads may time out and terminate if no tasks arrive within the keep-alive time, being replaced if needed when new tasks arrive.  设置策略，以控制在保持活动时间内没有任务到达时核心线程是否可能超时并终止，并在新任务到达时根据需要替换。 |
| boolean | [**allowsCoreThreadTimeOut**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#allowsCoreThreadTimeOut--)()  Returns true if this pool allows core threads to time out and terminate if no tasks arrive within the keepAlive time, being replaced if needed when new tasks arrive.  如果此池允许核心线程超时，并且在keepAlive时间内没有任务到达时终止，则返回true；如果有新任务到达，则在需要时替换该线程。 |
| boolean | [**awaitTermination**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#awaitTermination-long-java.util.concurrent.TimeUnit-)(long timeout, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit)  Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.  阻塞直到关闭请求后所有任务完成执行，或者发生超时，或者当前线程被中断（以先发生的为准）。 |
| protected void | [**beforeExecute**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#beforeExecute-java.lang.Thread-java.lang.Runnable-)([**Thread**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html) t, [**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) r)  Method invoked prior to executing the given Runnable in the given thread.  在给定线程中执行给定Runnable之前调用的方法。 |
| void | [**execute**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#execute-java.lang.Runnable-)([**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) command)  Executes the given task sometime in the future.  在将来的某个时间执行给定的任务。 |
| protected void | [**finalize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#finalize--)()  Invokes shutdown when this executor is no longer referenced and it has no threads.  当该执行程序不再被引用且没有线程时，调用关闭。 |
| int | [**getActiveCount**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getActiveCount--)()  Returns the approximate number of threads that are actively executing tasks.  返回正在主动执行任务的线程的大概数量。 |
| long | [**getCompletedTaskCount**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getCompletedTaskCount--)()  Returns the approximate total number of tasks that have completed execution.  返回已完成执行的任务的总数。 |
| int | [**getCorePoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getCorePoolSize--)()  Returns the core number of threads.  返回线程的核心数量。 |
| long | [**getKeepAliveTime**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getKeepAliveTime-java.util.concurrent.TimeUnit-)([**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit)  Returns the thread keep-alive time, which is the amount of time that threads in excess of the core pool size may remain idle before being terminated.  返回线程保持活动时间，该时间是超过核心池大小的线程在被终止之前可能保持空闲的时间。 |
| int | [**getLargestPoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getLargestPoolSize--)()  Returns the largest number of threads that have ever simultaneously been in the pool.  返回池中曾经同时存在的最大线程数。 |
| int | [**getMaximumPoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getMaximumPoolSize--)()  Returns the maximum allowed number of threads. |
| int | [**getPoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getPoolSize--)()  Returns the current number of threads in the pool.  返回池中的当前线程数。 |
| [**BlockingQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> | [**getQueue**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getQueue--)()  Returns the task queue used by this executor.  返回此执行程序使用的任务队列。 |
| [**RejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html) | [**getRejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getRejectedExecutionHandler--)()  Returns the current handler for unexecutable tasks.  返回当前可执行程序的处理程序。 |
| long | [**getTaskCount**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getTaskCount--)()  Returns the approximate total number of tasks that have ever been scheduled for execution.  返回计划执行的任务总数。 |
| [**ThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadFactory.html) | [**getThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#getThreadFactory--)()  Returns the thread factory used to create new threads.  返回用于创建新线程的线程工厂。 |
| boolean | [**isShutdown**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#isShutdown--)()  Returns true if this executor has been shut down. |
| boolean | [**isTerminated**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#isTerminated--)()  Returns true if all tasks have completed following shut down.  如果所有任务在关闭后都已完成，则返回true。 |
| boolean | [**isTerminating**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#isTerminating--)()  Returns true if this executor is in the process of terminating after [**shutdown()**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#shutdown--) or [**shutdownNow()**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#shutdownNow--) but has not completely terminated.  启动所有核心线程，使它们空闲地等待工作。 |
| int | [**prestartAllCoreThreads**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#prestartAllCoreThreads--)()  Starts all core threads, causing them to idly wait for work.  启动所有核心线程，使它们空闲地等待工作。 |
| boolean | [**prestartCoreThread**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#prestartCoreThread--)()  Starts a core thread, causing it to idly wait for work.  启动一个核心线程，使其闲置地等待工作。 |
| void | [**purge**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#purge--)()  Tries to remove from the work queue all [**Future**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html) tasks that have been cancelled.  尝试从工作队列中删除所有已取消的未来任务。 |
| boolean | [**remove**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#remove-java.lang.Runnable-)([**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) task)  Removes this task from the executor's internal queue if it is present, thus causing it not to be run if it has not already started.  如果执行程序的内部队列中存在该任务，则将其删除，如果尚未启动，则导致该任务无法运行。 |
| void | [**setCorePoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setCorePoolSize-int-)(int corePoolSize)  Sets the core number of threads.  设置核心线程数。 |
| void | [**setKeepAliveTime**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setKeepAliveTime-long-java.util.concurrent.TimeUnit-)(long time, [**TimeUnit**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/TimeUnit.html) unit)  Sets the time limit for which threads may remain idle before being terminated. |
| void | [**setMaximumPoolSize**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setMaximumPoolSize-int-)(int maximumPoolSize)  Sets the maximum allowed number of threads. |
| void | [**setRejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setRejectedExecutionHandler-java.util.concurrent.RejectedExecutionHandler-)([**RejectedExecutionHandler**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RejectedExecutionHandler.html) handler)  Sets a new handler for unexecutable tasks. |
| void | [**setThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#setThreadFactory-java.util.concurrent.ThreadFactory-)([**ThreadFactory**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadFactory.html) threadFactory)  Sets the thread factory used to create new threads. |
| void | [**shutdown**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#shutdown--)()  Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.  启动有序关闭，在该关闭中执行先前提交的任务，但不接受任何新任务。 |
| [**List**](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)<[**Runnable**](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)> | [**shutdownNow**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#shutdownNow--)()  Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.  尝试停止所有正在执行的任务，暂停正在等待的任务的处理，并返回正在等待执行的任务的列表。 |
| protected void | [**terminated**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#terminated--)()  Method invoked when the Executor has terminated.  执行程序终止时调用的方法。 |
| [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) | [**toString**](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html#toString--)()  Returns a string identifying this pool, as well as its state, including indications of run state and estimated worker and task counts.  返回标识此池及其状态的字符串，包括运行状态的指示以及估计的工作人员和任务计数。 |

# 其他

## 线程池最多接收多少任务？

回想以下线程池的主要处理流程：



我们总结出线程池接收任务时主要有以下过程：

1. 池中线程数逐渐达到corePoolSize，此时workQueue为空。
2. WorkQueue逐渐装满，此时池中线程数为corePoolSize。
3. 线程池中的线程数逐渐添加到MaximumPoolSize，此时workQueue为装满状态。

所以线程池中最多接收的任务数为：maximumPoolSize+workQueue.length