在java.util.concurrent包中多数的执行器实现都使用了由工作线程组成的线程池，工作线程独立于所它所执行的Runnable任务和Callable任务，并且常用来执行多个任务。

使用工作线程可以使创建线程的开销最小化。在大规模并发应用中，创建大量的Thread对象会占用占用大量系统内存，分配和回收这些对象会产生很大的开销。

一种最常见的线程池是固定大小的线程池。这种线程池始终有一定数量的线程在运行，如果一个线程由于某种原因终止运行了，线程池会自动创建一个新的线程来代替它。需要执行的任务通过一个内部队列提交给线程，当没有更多的工作线程可以用来执行任务时，队列保存额外的任务。

使用固定大小的线程池一个很重要的好处是可以实现优雅退化。例如一个Web服务器，每一个HTTP请求都是由一个单独的线程来处理的，如果为每一个HTTP都创建一个新线程，那么当系统的开销超出其能力时，会突然地对所有请求都停止响应。如果限制Web服务器可以创建的线程数量，那么它就不必立即处理所有收到的请求，而是在有能力处理请求时才处理。

创建一个使用线程池的执行器最简单的方法是调用[java.util.concurrent.Executors](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html)的[newFixedThreadPool](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool-int-)方法。Executors类还提供了下列一下方法：

* [newCachedThreadPool](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html#newCachedThreadPool-int-)方法创建了一个可扩展的线程池。适合用来启动很多短任务的应用程序。
* [newSingleThreadExecutor](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html#newSingleThreadExecutor-int-)方法创建了每次执行一个任务的执行器。
* 还有一些创建ScheduledExecutorService执行器的方法。

如果上面的方法都不满足需要，可以尝试[java.util.concurrent.ThreadPoolExecutor](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html)或者[java.util.concurrent.ScheduledThreadPoolExecutor](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html)。