1、Lock接口

java.util.concurrent.locks.Lock接口用作线程同步机制，类似于同步块。新的锁定机制更灵活，提供比同步块更多的选项。 锁和同步块之间的主要区别如下：

- **序列的保证 - 同步块不提供对等待线程进行访问的序列的任何保证，但Lock接口处理它。**
- **无超时，如果未授予锁，则同步块没有超时选项。Lock接口提供了这样的选项。**
- **单一方法同步块必须完全包含在单个方法中，而Lock接口的方法lock()和unlock()可以以不同的方式调用。**

Lock类中的方法

以下是Lock类中可用的重要方法的列表。

编号	方法	描述说明
1	public void lock()	获得锁
2	public void lockInterruptibly()	获取锁定，除非当前线程中断
3	public Condition newCondition()	返回绑定到此Lock实例的新Condition实例
4	public boolean tryLock()	只有在调用时才可以获得锁
5	public boolean tryLock(long time, TimeUnit unit)	如果在给定的等待时间内自由，并且当前线程未被中断，则获取该锁。
6	public void unlock()	释放锁

2、ReadWriteLock接口

java.util.concurrent.locks.ReadWriteLock接口允许一次读取多个线程，但一次只能写入一个线程。

- **读锁 - 如果没有线程锁定ReadWriteLock进行写入，则多线程可以访问读锁。**
- **写锁 - 如果没有线程正在读或写，那么一个线程可以访问写锁。**

锁方法

以下是Lock类中可用的重要方法的列表。

编号	方法	描述
1	public Lock readLock()	返回用于读的锁。
2	public Lock writeLock()	返回用于写的锁。

3、Condition接口

java.util.concurrent.locks.Condition接口提供一个线程挂起执行的能力，直到给定的条件为真。 Condition对象必须绑定到Lock，并使用newCondition()方法获取对象。

Condition类的方法

以下是Condition类中可用的重要方法的列表。

序号	方法名称	描述
1	public void await()	使当前线程等待，直到发出信号或中断信号。
2	public boolean await(long time, TimeUnit unit)	使当前线程等待直到发出信号或中断，或指定的等待时间过去。
3	public long awaitNanos(long nanosTimeout)	使当前线程等待直到发出信号或中断，或指定的等待时间过去。
4	public long awaitUninterruptibly()	使当前线程等待直到发出信号。
5	public long awaitUntil()	使当前线程等待直到发出信号或中断，或者指定的最后期限过去。
6	public void signal()	唤醒一个等待线程。
7	public void signalAll()	唤醒所有等待线程。

4、Exceutor接口

java.util.concurrent.Executor接口是支持启动新任务的一个简单接口。

Executor接口中的方法

序号	方法	描述
1	void execute(Runnable command)	在将来的某个时间执行给定的命令。

5、ExecutorService接口

java.util.concurrent.ExecutorService接口是Executor接口的子接口，并添加了功能来管理生命周期，这两个单独的任务和执行器本身。

ExecutorService接口的方法

序号	方法	描述
1	boolean awaitTermination(long timeout, TimeUnit unit)	阻止所有任务在关闭请求完成后执行，或发生超时，或当前线程中断，以先到者为准。
2	<T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)	执行给定的任务，返回持有它们的状态和结果的所有完成的列表。
3	<T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)	执行给定的任务，返回在所有完成或超时到期时持有其状态和结果的列表，以先发生者为准。
4	<T> T invokeAny(Collection<? extends Callable<T>> tasks)	执行给定的任务，返回一个已经成功完成的结果(即不抛出异常)。
5	<T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)	执行给定的任务，返回一个已经成功完成的结果(即，不抛出异常)，如果有则在给定的超时过去之前。
6	boolean isShutdown()	如果执行程序已关闭，则返回true。

6	boolean isShutdown()	如果执行程序已关闭，则返回true。
7	boolean isTerminated()	如果所有任务在关闭后完成，则返回true。
8	void shutdown()	启动有序关闭，其中先前提交的任务将被执行，但不会接受任何新任务。
9	List<Runnable> shutdownNow()	尝试停止所有主动执行的任务，停止等待任务的处理，并返回正在等待执行的任务列表。
10	<T> Future<T> submit(Callable<T> task)	提交值返回任务以执行，并返回代表任务待处理结果。
11	Future<?> submit(Runnable task)	提交一个可运行的任务执行，并返回一个表示该任务的Future。
12	<T> Future<T> submit(Runnable task, T result)	提交一个可运行的任务执行，并返回一个表示该任务的Future。

6、ScheduledExecutorService接口

java.util.concurrent.ScheduledExecutorService接口是ExecutorService接口的子接口，并支持将来和/或定期执行任务。

ScheduledExecutorService接口的方法

序号	方法	描述
1	ScheduledFuture schedule(Callable callable, long delay, TimeUnit unit)	创建并执行在给定延迟后启用ScheduledFuture。
2	ScheduledFuture<?> schedule(Runnable command, long delay, TimeUnit unit)	创建并执行在给定延迟后启用的单次操作。
3	ScheduledFuture<?> scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)	创建并执行在给定的初始延迟之后，随后以给定的时间段首先启用的周期性动作; 那就是执行会在initialDelay之后开始，然后是initialDelay + period，然后是initialDelay + 2 * period，等等。
4	ScheduledFuture<?> scheduleWithFixedDelay(Runnable command, long initialDelay, long delay, TimeUnit unit)	创建并执行在给定的初始延迟之后首先启用的定期动作，随后在一个执行的终止和下一个执行的开始之间给定的延迟。