```java
public interface ExecutorService extends Executor {

    /**
     * 关闭执行器，主要有以下特点：
     * 1. 已经提交给该执行器的任务将会继续执行，但是不再接受新任务的提交；
     * 2. 如果执行器已经关闭了，则再次调用没有副作用.
     */
    void shutdown();
    /**
     * 立即关闭执行器，主要有以下特点：
     * 1. 尝试停止所有正在执行的任务，无法保证能够停止成功，但会尽力尝试(例如，通过 Thread.interrupt
     * 2. 暂停处理已经提交但未执行的任务；
     *
     * @return 返回已经提交但未执行的任务列表
     */
    List<Runnable> shutdownNow();

    /**
     * 如果该执行器已经关闭，则返回true.
     */
    boolean isShutdown();
```

1. 尝试停止所有正在执行的任务，无法保证能够停止成功，但会尽力尝试(例如，通过 Thread.interrupt 中断任务，但是不响应中断的任务可能无法终止)；

```java
    /**
     * 判断执行器是否已经【终止】.
     * <p>
     * 仅当执行器已关闭且所有任务都已经执行完成，才返回true.
     * 注意：除非首先调用 shutdown 或 shutdownNow，否则该方法永远返回false.
     */
    boolean isTerminated();

    /**
     * 阻塞调用线程，等待执行器到达【终止】状态.
     *
     * @return {@code true} 如果执行器最终到达终止状态，则返回true；否则返回false
     * @throws InterruptedException if interrupted while waiting
     */
    boolean awaitTermination(long timeout, TimeUnit unit) throws InterruptedException;
```

```java
    /**
     * 提交一个具有返回值的任务用于执行.
     * 注意：Future的get方法在成功完成时将会返回task的返回值.
     *
     * @param task 待提交的任务
     * @param <T>  任务的返回值类型
     * @return 返回该任务的Future对象
     * @throws RejectedExecutionException 如果任务无法安排执行
     * @throws NullPointerException       if the task is null
     */
    <T> Future<T> submit(Callable<T> task);

    /**
     * 提交一个 Runnable 任务用于执行.
     * 注意：Future的get方法在成功完成时将会返回给定的结果(入参时指定).
     *
     * @param task   待提交的任务
     * @param result 返回的结果
     * @param <T>    返回的结果类型
     * @return 返回该任务的Future对象
     * @throws RejectedExecutionException 如果任务无法安排执行
     * @throws NullPointerException       if the task is null
     */
    <T> Future<T> submit(Runnable task, T result);
```

```
    /**
     * 提交一个 Runnable 任务用于执行.
     * 注意: Future的get方法在成功完成时将会返回null.
     *
     * @param task 待提交的任务
     * @return 返回该任务的Future对象
     * @throws RejectedExecutionException 如果任务无法安排执行
     * @throws NullPointerException       if the task is null
     */
    Future<?> submit(Runnable task);

    /**
     * 执行给定集合中的所有任务，当所有任务都执行完成后，返回保持任务状态和结果的 Future 列表.
     * <p>
     * 注意: 该方法为同步方法. 返回列表中的所有元素的Future.isDone() 为 true.
     *
     * @param tasks 任务集合
     * @param <T>   任务的返回结果类型
     * @return 任务的Future对象列表，列表顺序与集合中的迭代器所生成的顺序相同,
     * @throws InterruptedException       如果等待时发生中断，会将所有未完成的任务取消.
     * @throws NullPointerException       任一任务为 null
     * @throws RejectedExecutionException 如果任一任务无法安排执行
     */
    <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws Interrup
```

执行给定集合中的所有任务，当所有任务都执行完成后或超时期满时（无论哪个首先发生），返回保持任务状态和结果的 Future 列表.

```
    /**
     * 执行给定集合中的所有任务，当所有任务都执行完成后或超时期满时（无论哪个首先发生），返回保持
     */
    <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks, long timeout, T

    /**
     * 执行给定集合中的任务，只有其中某个任务率先成功完成（未抛出异常），则返回其结果.
     * 一旦正常或异常返回后，则取消尚未完成的任务.
     */
    <T> T invokeAny(Collection<? extends Callable<T>> tasks) throws InterruptedException, E

    /**
     * 执行给定集合中的任务，如果在给定的超时期满前，某个任务已成功完成（未抛出异常），则返回其结
     * 一旦正常或异常返回后，则取消尚未完成的任务.
     */
    <T> T invokeAny(Collection<? extends Callable<T>> tasks, long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException, TimeoutException;
}
```