ExecutorService常用方法和newFixedThreadPool创建固定大小的线程池

1、ExecutorService：

是一个接口，继承了Executor：

```
public interface ExecutorService extends Executor {
}
```

2、Executor：

而Executor亦是一个接口，该接口只包含了一个方法：

```
void execute(Runnable command);
```

3、Executors：

该类是一个辅助类，此包中所定义的 Executor、ExecutorService、ScheduledExecutorService、ThreadFactory 和 Callable 类的工厂和实用方法。

此类支持以下各种方法：

• 创建并返回设置有常用配置字符串的 ExecutorService 的方法。 • 创建并返回设置有常用配置字符串的 ScheduledExecutorService 的方法。 • 创建并返回"包装的"ExecutorService 方法，它通过使特定于实现的方法不可访问来禁用重新配置。 • 创建并返回 ThreadFactory 的方法，它可将新创建的线程设置为已知的状态。 • 创建并返回非闭包形式的 Callable 的方法，这样可将其用于需要 Callable 的执行方法中。

4、创建ExecutorService的方法：

```
newFixedThreadPool()
```

创建一个可重用固定线程数的线程池，以共享的无界队列方式来运行这些线程。

5、ExecutorService的方法：

shutdown

void **shutdown**()

启动一次顺序关闭，执行以前提交的任务，但不接受新任务。如果已经关闭，则调用没有其他作用。

抛出：

SecurityException - 如果安全管理器存在并且关闭此 ExecutorService 可能操作某些不允许调用者修改的线程（因为它没有保持 RuntimePermission("modifyThread")），或者安全管理器的 checkAccess 方法拒绝访问。

启动一次顺序关闭，执行以前提交的任务，但不接受新任务。如果已经关闭，则调用没有其他作用。

awaitTermination

boolean **awaitTermination**(long timeout,
TimeUnit unit)
throws InterruptedException

请求关闭、发生超时或者当前线程中断，无论哪一个首先发生之后，都将导致阻塞，直到所有任务完成执行。

参数：

timeout - 最长等待时间

unit - timeout 参数的时间单位

返回：

如果此执行程序终止，则返回 true；如果终止前超时期满，则返回 false

抛出：

InterruptedException - 如果等待时发生中断

请求关闭、发生超时或者当前线程中断，无论哪一个首先发生之后，都将导致阻塞，直到所有任务完成执行。既是等待所有子线程执行结束。

execute

void **execute**(Runnable command)

在未来某个时间执行给定的命令。该命令可能在新的线程、已入池的线程或者正调用的线程中执行，这由 Executor实现决定。

参数：

command - 可运行的任务

抛出：

RejectedExecutionException - 如果不能接受执行此任务。

NullPointerException - 如果命令为 null

在未来某个时间执行给定的命令。该命令可能在新的线程、已入池的线程或者正调用的线程中执行，这由 Executor 实现决定。

submit

Future<?> **submit**(Runnable task)

提交一个 Runnable 任务用于执行，并返回一个表示该任务的 Future。该 Future 的 get 方法在成功 完成时将会返回 null。

参数：

task - 要提交的任务

返回：

表示任务等待完成的 Future

抛出：

RejectedExecutionException - 如果任务无法安排执行

NullPointerException - 如果该任务为 null

提交一个 Runnable 任务用于执行，并返回一个表示该任务的 Future。该 Future 的 get 方法在成功 完成时将会返回 null。

6、下面是相关的使用例子：

```
public class ExecutorServiceTest {

    public static void main(String[] args) throws IOException, InterruptedException {
        // 创建一个固定大小的线程池
        ExecutorService service = Executors.newFixedThreadPool(3);
        for (int i = 0; i < 10; i++) {
            System.out.println("创建线程" + i);
            Runnable run = new Runnable() {
                @Override
                public void run() {
                    System.out.println("启动线程");
                }
            };
            // 在未来某个时间执行给定的命令
            service.execute(run);
        }
        // 关闭启动线程
        service.shutdown();
        // 等待子线程结束，再继续执行下面的代码
        service.awaitTermination(Long.MAX_VALUE, TimeUnit.DAYS);
        System.out.println("all thread complete");
    }
}
```

可以发现线程被循环创建，但是启动线程却不是连续的，而是由ExecutorService决定的。