Executors-API

- **生成固定大小的线程池**

```
  // 生成固定大小的线程池
  public static ExecutorService newFixedThreadPool(int nThreads) {
    return new ThreadPoolExecutor(nThreads, nThreads, 0L,
TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());
  }

  // corePoolSize —— 最小线程数；如果allowCoreThreadTimeOut设置为true，线
程池最后会减少到0；
  public ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long
keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue) {
        this(corePoolSize, maximumPoolSize, keepAliveTime, unit,
workQueue,
             Executors.defaultThreadFactory(), defaultHandler);
    }
```

对于 `allowCoreThreadTimeOut`

> If false (default), `core threads` stay alive even when idle.
> If true, `core threads` use keepAliveTime to time out waiting for work.

- **CachedThreadPool**
  `newCachedThreadPool()` 生成一个初始大小为0，空闲时间为60s，没有上限的连接池：

```
public static ExecutorService newCachedThreadPool() {
      return new ThreadPoolExecutor(0, Integer.MAX_VALUE,
                                    60L, TimeUnit.SECONDS,
                                    new SynchronousQueue<Runnable>
());
  }
```

  `newCachedThreadPool(ThreadFactory threadFactory)` 同上；

- **ScheduledThreadPool**
  `newScheduledThreadPool(int corePoolSize)` 和
  `newScheduledThreadPool(int corePoolSize, ThreadFactory threadFactory)` 生成延迟或

定期执行任务的线程池；

- `ScheduledExecutorService#scheduleWithFixedDelay`和`ScheduledExecutorService#scheduleAtFixedRate`
- ScheduledExecutorService#`scheduleWithFixedDelay`：是`前一次执行结束`到`后一次执行开始`的间隔为`delay`；
- ScheduledExecutorService#`scheduleAtFixedRate`：执行时刻是`initialDelay`、`initialDelay + period`、`initialDelay + period * 2`、`initialDelay + period * 3`。。。等，如果任务执行时间大于`period`，那么下一次执行开始时刻就晚一点；
- **单线程**
  - `newSingleThreadExecutor()`生成一个单线程并执行任务队列；
  - 如果单线程因为执行任务而意外中断或关闭，那么会生成一个新的线程代替旧的线程继续执行接下来的任务；
  - `newSingleThreadExecutor()`和`newFixedThreadPool(1)`区别：

```
  // final ExecutorService single =
Executors.newSingleThreadExecutor();
  final ExecutorService fixed = Executors.newFixedThreadPool(1);
  ThreadPoolExecutor executor = (ThreadPoolExecutor) fixed;
  executor.setCorePoolSize(4);  // newFixedThreadPool(1)可以再设置大小
```

- **单线程 + 定时/延迟**
  `newSingleThreadScheduledExecutor`
- **newWorkStealingPool**
  作用：貌似是尽可能地利用所有处理器，生成一个线程池；