线程池之ThreadPoolExecutor状态控制

读懂ThreadPoolExecutor执行原理，需要先掌握其状态控制的方式，因为使用了大量位运算，读起来有点吃力，所以单独用一篇文章分析。

以下是ThreadPoolExecutor状态控制的主要变量和方法：

//原子状态控制数

private final AtomicInteger ctl = new AtomicInteger(ctlOf(RUNNING, 0));

//29比特位

private static final int COUNT\_BITS = Integer.SIZE - 3;

//实际容量 2^29-1

private static final int CAPACITY = (1 << COUNT\_BITS) - 1;

// runState is stored in the high-order bits

// runState存储在高位中

private static final int RUNNING = -1 << COUNT\_BITS;

private static final int SHUTDOWN = 0 << COUNT\_BITS;

private static final int STOP = 1 << COUNT\_BITS;

private static final int TIDYING = 2 << COUNT\_BITS;

private static final int TERMINATED = 3 << COUNT\_BITS;

// Packing and unpacking ctl 打包和解压ctl

// 解压runState

private static int runStateOf(int c) { return c & ~CAPACITY; }

// 解压workerCount

private static int workerCountOf(int c) { return c & CAPACITY; }

// 打包ctl

private static int ctlOf(int rs, int wc) { return rs | wc; }

线程池使用一个AtomicInteger的ctl变量将workerCount（工作线程数量）和runState（运行状态）两个字段压缩在一起，这种做法在Java源码里经常有出现，如在ReentrantReadWriteLock里就将一个int分成高16位和低16位，分别表示读锁状态和写锁状态。ThreadPoolExecutor里也使用了同样的思想，表现得更加复杂。

ThreadPoolExecutor用3个bit位表示runState，29个bit位表示workerCount。因此这里需要特别说明的是：

**确切的说，当最大线程数量配置为Integer.MXA\_VAULE时，ThreadPoolExecutor的线程最大数量依然是2^29-1**。

目前来看这是完全够用的，但随着计算机的不断发展，真的到了不够用的时候可以改变为AtomicLong。这如同32位系统时间戳会在2038年01月19日03时14分07秒耗尽一样，当以后我们的系统线程能够超过2^29-1时，这些代码就需要调整了。对于未来，无限可能。

运行状态解释：

|  |  |
| --- | --- |
| 状态 | 解释 |
| RUNNING | 运行态，可处理新任务并执行队列中的任务 |
| SHUTDOWN | 关闭态，不接受新任务，但处理队列中的任务 |
| STOP | 停止态，不接受新任务，不处理队列中的任务，且打断运行中的任务 |
| TIDYING | 整理态，所有任务已经结束，workerCount=0，将执行terminated()方法 |
| TERMINATED | 结束态，terminated()方法已完成 |

整个ctl的状态，会在线程池的不同运行阶段进行CAS转换。