链滴

线程池源码阅读

作者：614756773

原文链接：https://ld246.com/article/1575367949202

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20190526.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h3 id="写在前面的话">写在前面的话</h3>

本次源码阅读的主要类

AbstractExecutorService
ThreadPoolExecutor

需要提前了解的相关知识

线程池参数，参考我之前的文章<a href="https://ld246.com/forward?goto=http%3A%2F%2
www.qz-hotpot.xyz%2Farticles%2F2019%2F09%2F26%2F1569484316929.html" target="_blan
" rel="nofollow ugc">关于线程池参数的理解
位运算 <code>ThreadPoolExecutor使用位运算来做状态标志</code>
Unsafe 类的 API
BlockQueue 类的 API <code>存取任务都会用到</code>
Runnable，Future

<blockquote>
以上的类都在 <code>java.util.concurrent</code> 包中
</blockquote>
<h3 id="AbstractExecutorService">AbstractExecutorService</h3>
<table>
<thead>
<tr>
<th>方法名称</th>
<th>功能</th>
</tr>
</thead>
<tbody>
<tr>
<td>invokeAny</td>
<td>执行队列中的所有任务，
当某一个方法完成时则会立即返回，同时取消其余未完成的任务<
td>
</tr>
<tr>
<td>invokeAll</td>
<td>执行所有任务，知道全部任务都完成时才返回</td>
</tr>
</tbody>
</table>
<h4 id="invokeAny----doInvokeAny">invokeAny -> doInvokeAny</h4>
<pre><code class="highlight-chroma"><span class="highlight
cl">   private <T> T doInvokeAny(Collection<? extends Callable<T>> tasks,

boolean timed, long nanos)
        throws Interr
uptedException, ExecutionException, TimeoutException {
        if (tasks ==
ull)

```
</span></span><span class="highlight-line"><span class="highlight-cl">            throw new
NullPointerException();
</span></span><span class="highlight-line"><span class="highlight-cl">        int ntasks = t
sks.size();
</span></span><span class="highlight-line"><span class="highlight-cl">        if (ntasks ==
)
</span></span><span class="highlight-line"><span class="highlight-cl">            throw new
llegalArgumentException();
</span></span><span class="highlight-line"><span class="highlight-cl">        ArrayList&lt;
uture&lt;T&gt;&gt; futures = new ArrayList&lt;Future&lt;T&gt;&gt;(ntasks);
</span></span><span class="highlight-line"><span class="highlight-cl">        ExecutorCo
pletionService&lt;T&gt; ecs =
</span></span><span class="highlight-line"><span class="highlight-cl">            new Execu
orCompletionService&lt;T&gt;(this);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        try {
</span></span><span class="highlight-line"><span class="highlight-cl">            ExecutionE
ception ee = null;
</span></span><span class="highlight-line"><span class="highlight-cl">            final long
eadline = timed ? System.nanoTime() + nanos : 0L;
</span></span><span class="highlight-line"><span class="highlight-cl">            Iterator&lt
? extends Callable&lt;T&gt;&gt; it = tasks.iterator();
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">            futures.ad
(ecs.submit(it.next()));
</span></span><span class="highlight-line"><span class="highlight-cl">            --ntasks;
</span></span><span class="highlight-line"><span class="highlight-cl">            int active =
1;
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">            for (;;) {
</span></span><span class="highlight-line"><span class="highlight-cl">                Future&
t;T&gt; f = ecs.poll();
</span></span><span class="highlight-line"><span class="highlight-cl">                // #1
</span></span><span class="highlight-line"><span class="highlight-cl">                if (f ==
ull) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    // #1.

</span></span><span class="highlight-line"><span class="highlight-cl">                    if (nta
ks &gt; 0) {
</span></span><span class="highlight-line"><span class="highlight-cl">                        --n
asks;
</span></span><span class="highlight-line"><span class="highlight-cl">                        fut
res.add(ecs.submit(it.next()));
</span></span><span class="highlight-line"><span class="highlight-cl">                        ++
ctive;
</span></span><span class="highlight-line"><span class="highlight-cl">                    }
</span></span><span class="highlight-line"><span class="highlight-cl">                    // #1.

</span></span><span class="highlight-line"><span class="highlight-cl">                    else if
(active == 0)
</span></span><span class="highlight-line"><span class="highlight-cl">                        bre
k;
</span></span><span class="highlight-line"><span class="highlight-cl">                    // #1
3
```

```
                else if (timed) {
                    f = ecs.poll(nanos, TimeUnit.NANOSECONDS);
                    if (f == null)
                        throw new TimeoutException();
                    nanos = deadline - System.nanoTime();
                }
                // #1.

                else
                    f = ecs.take();
            }
            // #2
            if (f != null) {
                --active;
                try {
                    return f.get();
                } catch (ExecutionException eex) {
                    ee = eex;
                } catch (RuntimeException rex) {
                    ee = new ExecutionException(rex);
                }
            }

            if (ee == null)
                ee = new ExecutionException();
            // #3
            throw ee;

        } finally {
            for (int i = 0, size = futures.size(); i < size; i++)
                futures.get(i).cancel(true);
        }
    }
```


理想情况下的代码流程

执行#2
<code>即直接获取到了任务的结果，然后立即返回。</code>
<code>在返后之前会执行finally里的语句块，取消其余未完成的任务</code>

不能立即得到任务结果

首先执行#1

若存在待执行的任务，则执行#1.1
<code>添加一个新的任务去执行。意思就是说之前的个任务没执行完，我们可以认为它工作的比较慢，再上一个新任务试试，指不定这个新任务还会先完</code>
若也没有在执行的任务，则执行#1.2，然后执行#3 抛出异常。
<code>很明显这种情形下出现了某些不可预知的问题，</code>
<code>毕竟即获取不到结果，当前又没有任务在执行还没有可用的任务，那妥妥的任务在执行的时候出现了异常</code>
若设置了超时等待，则执行#1.3
<code>设置了超时等待的话，就用poll在指定的时间范围去获取任务结果，</code>
<code>如果时间到了还没获取到，那么按照调用者的意图就应该出异常，告诉他了</code>
若没有设置超时等待，则执行#1.4
<code>阻塞的等待任务的返回结果</code>
<cod>ecs肯定使用了生产者消费者模式，只要有一个任务完成了，肯定就会把数据存到ecs中</code></l>

然后执行#2

<h4 id="invokeAll">invokeAll</h4>
<pre><code class="highlight-chroma">   public <T> List<Future<T>> invokeAll(Collection<? extends Callable&t;T>> tasks)
        throws Interr ptedException {
        if (tasks == ull)
            throw new NullPointerException();
        ArrayList< uture<T>> futures = new ArrayList<Future<T>>(tasks.size());
        boolean don = false;
        try {
            // #1
            for (Callab e<T> t : tasks) {
                RunnablFuture<T> f = newTaskFor(t);
                futures. dd(f);
                execute( );
            }
            // #2
            for (int i =

```
, size = futures.size(); i &lt; size; i++) {
</span></span><span class="highlight-line"><span class="highlight-cl">                Future&
t;T&gt; f = futures.get(i);
</span></span><span class="highlight-line"><span class="highlight-cl">                if (!f.isD
ne()) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    try {
</span></span><span class="highlight-line"><span class="highlight-cl">                        f.ge
();
</span></span><span class="highlight-line"><span class="highlight-cl">                    } catc
 (CancellationException ignore) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    } catc
 (ExecutionException ignore) {
</span></span><span class="highlight-line"><span class="highlight-cl">                    }
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">            // #3
</span></span><span class="highlight-line"><span class="highlight-cl">            done = tru
;
</span></span><span class="highlight-line"><span class="highlight-cl">            return futu
es;
</span></span><span class="highlight-line"><span class="highlight-cl">        } finally {
</span></span><span class="highlight-line"><span class="highlight-cl">            if (!done)
</span></span><span class="highlight-line"><span class="highlight-cl">                for (int i
= 0, size = futures.size(); i &lt; size; i++)
</span></span><span class="highlight-line"><span class="highlight-cl">                    future
.get(i).cancel(true);
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<p>invokeAll 方法就简单些了，就是单纯的阻塞式的去获取每一个任务的结果然后返回。</p>
<ul>
<li>理想情况下的代码流程
<ul>
<li>执行#1 <code>提交并执行所有的任务</code></li>
<li>执行#2 <code>阻塞式的去获取所有任务的结果</code></li>
<li>执行#3 <code>标志所有的任务已完成</code></li>
</ul>
</li>
<li>异常情况下的代码流程
<ul>
<li>执行#1</li>
<li>执行#2</li>
<li>抛出了非 <code>CancellationException</code> 和 <code>ExecutionException</code>
异常</li>
<li>不会执行#3！！！</li>
<li>执行 finally，将所有的任务取消掉</li>
</ul>
</li>
</ul>
<hr>
<h3 id="ThreadPoolExecutor">ThreadPoolExecutor</h3>
<h4 id="主要方法">主要方法</h4>
<table>
<thead>
```

<tr>
<th>名称</th>
<th>作用</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute</td>
<td>提交任务</td>
</tr>
<tr>
<td>submit</td>
<td>提交任务，同时能够获取执行结果。
该方法在父类 <code>AbstractExecutorService</code> 中</td>
</tr>
<tr>
<td>addWorker</td>
<td>启动一个新线程，并且执行任务。
当线程数量大于线程参数的 <code>coreSize</code> 时就不会再执行了</td>
</tr>
</tbody>
</table>
<h4 id="主要属性">主要属性</h4>
<table>
<thead>
<tr>
<th>变量名</th>
<th>作用</th>
<th>二进制码</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctl</td>
<td>该变量是一个原子类的 Integer，二进制码一共 32 位，
高 3 位用来标志线程池状态，剩的 29 为用来记录线程数量</td>
<td></td>
</tr>
<tr>
<td>RUNNING</td>
<td><code>运行中</code> 的二进制标志</td>
<td>高 3 位为 <code>111</code></td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td><code>优雅关闭</code> 的二进制标志</td>
<td>高 3 位为 <code>000</code></td>
</tr>
<tr>
<td>STOP</td>
<td><code>暴力关闭</code> 的二进制标志</td>
<td>高 3 位为 <code>001</code></td>
</tr>
<tr>

<td>TIDYING</td>
<td><code>即将完全关闭</code> 的二进制标志</td>
<td>高 3 位为 <code>010</code></td>
</tr>
<tr>
<td>TERMINATED</td>
<td><code>已完全关闭</code> 的二进制标志</td>
<td>高 3 位为 <code>011</code></td>
</tr>
<tr>
<td>上方表格中，后面的 5 个变量都是用来标志线程池的状态。他们是有顺序的，越往后说明线程的活跃程度越低。
并且只有当值为负数时线程池才是 <code>运行中</code> 的状态</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
复习一下各个状态的含义：

RUNNING

线程池处于运行中，可以接受新的任务

SHUTDOWN

线程池开始关闭，不会接受新的任务，但是已经提交却还处于执行中的任务会让它执行完成

STOP

立即关闭线程池，不接受新任务，同时打断还在执行中的任务

TIDYING

所有任务已关闭，<code>ctl</code> 中记录的线程数也为 0 了，然后会调用 <code>terminated()</code> 函数，
<code>terminated()</code> 函数在 ThreadPoolExecutor 中并没有现，我们可以自己去重写它用于完成一些自定义的收尾工作

TERMINATED

线程池彻底停止

<h4 id="辅助方法">辅助方法</h4>
<table>
<thead>
<tr>
<th>名称</th>
<th>作用</th>
</tr>

</thead>
<tbody>
<tr>
<td>runStateOf</td>
<td>取得 ctl 的高 3 位，即线程池的运行状态</td>
</tr>
<tr>
<td>workerCountOf</td>
<td>取得 ctl 的低 29 位，即运行的线程数量</td>
</tr>
<tr>
<td>ctlOf</td>
<td>将运行状态和运行线程数量存放在一个变量中</td>
</tr>
</tbody>
</table>
<h4 id="execute方法解析">execute 方法解析</h4>
<pre><code class="highlight-chroma">   public void execute(Runnable command) {
        if (command == null)
            throw new NullPointerException();
        /*
         * 1. 如果线池中的运行线程数小于corePoolSize那么就启动
         * 一个新的程去执行任务，如果成功那么就退出该方法，不
     * 功的原因是：于并发的环境下，在判断的时候线程数小于
     * corePoolSiz，但此时其他地方启动了一个新线程刚好导致
     * 运行线程数达corePoolSize，调用addWorker就会失败
     *
         * 2. 如果任能存放进队列，那么仍然需要再次检查线程池
         * 状态，因可能在判断的时候线程池还是运行状态，但是
     * 进入方法体之线程池就被关闭了，所以需要再检查一下，
     * 并且在必要时除任务或者启动新的线程
         *
         * 3. 如果我不能将任务放进队列中，说明队列已经满了，
     * 我们就会尝试启动一个线程（此时启动的线程就是由
     * corePoolSiz增加到maxPoolSize的过程）。如果启动失败
     * 则拒绝该任务启动失败的原因可能是线程数已经大于
     * maxPoolSize

```
</span></span><span class="highlight-line"><span class="highlight-cl">         */
</span></span><span class="highlight-line"><span class="highlight-cl">        int c = ctl.get
);
</span></span><span class="highlight-line"><span class="highlight-cl">    // #1
</span></span><span class="highlight-line"><span class="highlight-cl">        if (workerCo
ntOf(c) &lt; corePoolSize) {
</span></span><span class="highlight-line"><span class="highlight-cl">            if (addWor
er(command, true))
</span></span><span class="highlight-line"><span class="highlight-cl">                return;
</span></span><span class="highlight-line"><span class="highlight-cl">            c = ctl.get()

</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    // #2
</span></span><span class="highlight-line"><span class="highlight-cl">        if (isRunning(
) &amp;amp;&amp;amp; workQueue.offer(command)) {
</span></span><span class="highlight-line"><span class="highlight-cl">            int recheck
= ctl.get();
</span></span><span class="highlight-line"><span class="highlight-cl">            if (! isRunn
ng(recheck) &amp;amp;&amp;amp; remove(command))
</span></span><span class="highlight-line"><span class="highlight-cl">                reject(c
mmand);
</span></span><span class="highlight-line"><span class="highlight-cl">            else if (wo
kerCountOf(recheck) == 0)
</span></span><span class="highlight-line"><span class="highlight-cl">                addWor
er(null, false);
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    // #3
</span></span><span class="highlight-line"><span class="highlight-cl">        else if (!add
orker(command, false))
</span></span><span class="highlight-line"><span class="highlight-cl">            reject(co
mand);
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<p>简单来说 <code>execute</code> 方法主要判断是 <code>启动线程执行任务</code> 还是
code>将任务放进待执行队列</code>，<br>任务执行的调用主要还是靠 <code>addWorker</co
e> 方法来完成</p>
<h4 id="addWorker方法解析">addWorker 方法解析</h4>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">    private boolean addWorker(Runnable firstTask, boolean core) {
</span></span><span class="highlight-line"><span class="highlight-cl">        retry:
</span></span><span class="highlight-line"><span class="highlight-cl">        for (;;) {
</span></span><span class="highlight-line"><span class="highlight-cl">            int c = ctl.
et();
</span></span><span class="highlight-line"><span class="highlight-cl">            int rs = ru
StateOf(c);
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">            // #1
</span></span><span class="highlight-line"><span class="highlight-cl">            if (rs &gt;=
SHUTDOWN &amp;amp;&amp;
</span></span><span class="highlight-line"><span class="highlight-cl">                ! (rs ==
HUTDOWN &amp;amp;&amp;
</span></span><span class="highlight-line"><span class="highlight-cl">                    firstTa
k == null &amp;amp;&amp;
</span></span><span class="highlight-line"><span class="highlight-cl">                    ! wor
```

```
Queue.isEmpty()))
                return false;
            // #2
            for (;;) {
                int wc = workerCountOf(c);
                if (wc &gt;= CAPACITY ||
                    wc &gt;= (core ? corePoolSize : maximumPoolSize))
                    return false;
                if (compareAndIncrementWorkerCount(c))
                    break retry;
                c = ctl.get();  // Re-read ctl
                if (runStateOf(c) != rs)
                    continue retry;
            }
        }

        boolean workerStarted = false;
        boolean workerAdded = false;
        Worker w = null;
    // #3
        try {
        // #3.1
            w = new Worker(firstTask);
            final Thread t = w.thread;
            if (t != null) {
                final ReentrantLock mainLock = this.mainLock;
                mainLock.lock();
                try {
                    int rs = runStateOf(ctl.get());
                    // #3.

                    if (rs &lt; SHUTDOWN ||
                        (rs == SHUTDOWN &amp;&amp; firstTask == null)) {
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">                    if (t.
sAlive()) // precheck that t is startable
</span></span><span class="highlight-line"><span class="highlight-cl">                        t
row new IllegalThreadStateException();
</span></span><span class="highlight-line"><span class="highlight-cl">                    wo
kers.add(w);
</span></span><span class="highlight-line"><span class="highlight-cl">                    int
 = workers.size();
</span></span><span class="highlight-line"><span class="highlight-cl">                    if (s
&gt; largestPoolSize)
</span></span><span class="highlight-line"><span class="highlight-cl">                        l
rgestPoolSize = s;
</span></span><span class="highlight-line"><span class="highlight-cl">                    wo
kerAdded = true;
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">            } finally {
</span></span><span class="highlight-line"><span class="highlight-cl">                main
ock.unlock();
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        // #3.3
</span></span><span class="highlight-line"><span class="highlight-cl">            if (worke
Added) {
</span></span><span class="highlight-line"><span class="highlight-cl">                t.start
);
</span></span><span class="highlight-line"><span class="highlight-cl">                worke
Started = true;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        } finally {
</span></span><span class="highlight-line"><span class="highlight-cl">            if (! worker
tarted)
</span></span><span class="highlight-line"><span class="highlight-cl">                addWor
erFailed(w);
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        return worke
Started;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<p>该方法是用来创建，运行，清理线程的。其两个参数的作用：</p>
<ul>
<li>firstTask
<ul>
<li>该参数用来表示创建线程时首先要执行这个任务，这样可以避免放入到队列中进行排队</li>
<li>只有在创建 coreThread 时以及创建 maxThread 时才会传递该参数，其余时候都是传递的 null<
li>
</ul>
<blockquote>
<p>举个例子，如果我们在创建 coreThread 时没有传递 firstTask，那么这个新的线程就会先空闲
，然后等待任务队列里有数据时，从任务队列里取一个任务出来再执行。这样这个线程刚创建那会儿
会闲着，浪费资源 <code>我们的目标就是榨干CPU，不能浪费</code></p>
</blockquote>
</li>
<li>core
<ul>
```

该参数在代码块里只用了一次，就是来判断是创建 coreThread 还是临时的 maxThread，
后根据不同的类型，来判断他们的数量是否达到上限了

执行流程

首先执行#1

检查一下线程池的状态，如果是 STOP，TIDYING，TERMINATED 状态的话，则直接返回 false 表明启动线程失败。
如果现在状态是 SHUTDOWN，但是 firstTask 不为空或者 workQueue 为空的话，那么也直接回 false。<code>因为SHUTDOWN状态允许还在运行中的任务继续执行，但是若还想启动线程并携带一个任务那就不允许了</code>

接着执行#2

检查线程数量是否太多了，如果过多则直接返回 false
如果线程数量还允许继续增加，那么使用 CAS 添加线程数，添加成功则跳出大循环去执行#3

添加失败了，那就在判断一下线程池的状态和之前是否相同，不同的话说明出现了一点点小问题那么就从头再来，继续执行#1

兜兜转转一圈终于申请到了可以添加线程的权限，接下来执行#3，进行真正的创建线程

#3.1 创建一个 worker <code>worker内部创建了线程</code>
#3.2 检查线程池状态，只有当以下两种情况时才可能说明创建线程真正的成功

情况 1：线程池是 RUNNING
情况 2：线程池是 SHUTDOWN 但是没有携带任务

#3.3 上面的 2 个步骤都通过后接着启动线程！<code>终于启动了</code>

<blockquote>
主流程大体就是这样，还需要去看一看 Worker 的实现才行
</blockquote>