参考Dubbo线程池模型实现快速消费线程池

如何解决 JDK 线程池中不超过最大线程数下即时快速消费任务，而不是在队列中堆积。

因为最近业务落地改造中需要线程池，又去看了一遍源码，防止线上埋雷，也再次回顾了这个问题。

然后发现网上也有这种问题提问，虽然是不同的提问，但是核心思想是一致的。

How to get the ThreadPoolExecutor to increase threads to max ...

I’ ve been frustrated for some time with the default behavior of ThreadPoolExecutor ...

Stack Overflow

业务是多变的，而 JDK 中的线程池消费流程却是固定的，所以 **基于阻塞队列、线程池扩展改变了原有流程。**

线程池参数

我们这里讲解以 **ThreadPoolExecutor#execute(Runnable runnable)** 举例，这里先说下线程池的一些参数。

本篇只是说明上述问题，不会对线程池做详细讲解。

Java | 复制代码

```java
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler) {...}
```

- **corePoolSize**：线程池中的核心线程数量，如果没有全局设置池内线程的过期时间，池内会维持此数量线程。
- **maximumPoolSize**：线程池中的最大线程数量，当核心线程都在运行任务，并且阻塞队列中任务数量已满，此时会创建非核心线程。
- **keepAliveTime & unit**：线程池中线程过期时间以及时间单位。

- **workQueue**：存放线程池内任务的阻塞队列，如 ArrayBlockingQueue、LinkedBlockingQueue...
- **threadFactory**：创建线程池中线程的线程工厂，可以在创建线程时初始化优先级、名称、守护状态...
- **handler**：当线程池中全部线程都在运行，阻塞队列也满的时候，会将添加的任务执行拒绝策略，JDK 线程池中实现了四种拒绝策略，默认 **AbortPolicy**，抛出异常。

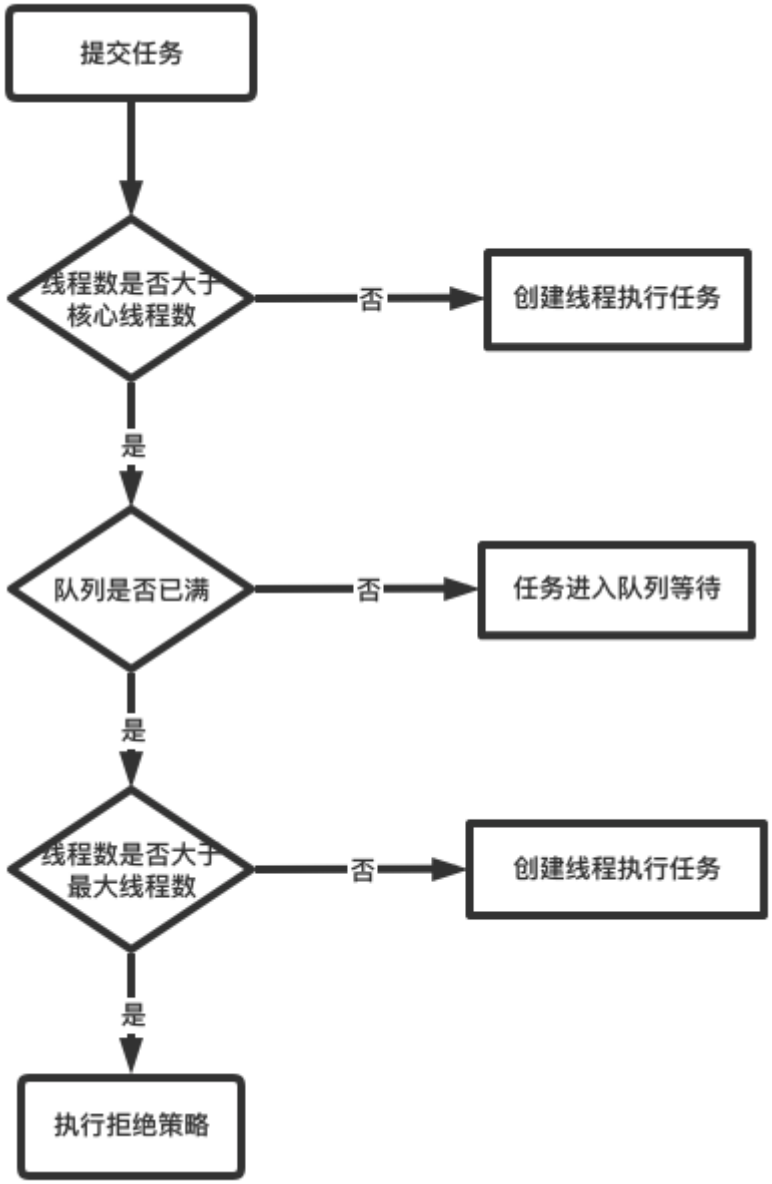
线程池任务添加流程

相信大家在网上看到过许多类似的线程池执行流程图哈，这里还是简要赘述下，源码如下:

Java | 复制代码

```java
public void execute(Runnable command) {
    ...
    int c = ctl.get();
    if (workerCountOf(c) < corePoolSize) {
        if (addWorker(command, true))
            return;
        c = ctl.get();
    }
    if (isRunning(c) && workQueue.offer(command)) {
        int recheck = ctl.get();
        if (!isRunning(recheck) && remove(command))
            reject(command);
        else if (workerCountOf(recheck) == 0)
            addWorker(null, false);
    } else if (!addWorker(command, false))
        reject(command);
}
```

1. 线程池提交任务首先判断当前线程数是否大于核心线程数，否则创建核心线程执行任务；
2. 如果当前线程超过了核心线程数，判断阻塞队列是否已满，否则将任务添加到队列中；
3. 如果阻塞队列已满，判断当前线程是否大于最大线程数，否则创建非核心线程执行任务；
4. 如果当前线程大于或等于最大线程数，执行拒绝策略。



这道问题的意图就是要将第二步就行改写。

如果当前线程大于核心线程数，不将任务放入阻塞队列，而是创建非核心线程执行任务。

举例说明一下:

```java
public static void main(String[] args) {
    ThreadPoolExecutor threadPoolExecutor =
            new ThreadPoolExecutor(1, 3, 60,
                    TimeUnit.SECONDS,
                    new ArrayBlockingQueue(10));

    for (int i = 0; i < 7; i++) {
        threadPoolExecutor.execute(() -> {
            System.out.println(Thread.currentThread().getName() + "-执行任务
            LockSupport.park();
        });
    }
    threadPoolExecutor.shutdown();
    /**
     * pool-1-thread-1执行任务
     */
}
```

看到这段代码，正常情况下只会有一个任务会被执行，其余任务会被放置阻塞队列中。

而我们需要做的就是，**发现池内线程大于核心线程数，不放入阻塞队列，而是创建非核心线程进行消费任务。**

本地代码实现参考 Dubbo 源码中 **EagerThreadPoolExecutor**，确实能实现对应效果，这里就不演示了，一起看一下 Dubbo 如何做的。

Dubbo 中实现的快速消费

Dubbo 中涉及到的类有两个，EagerThreadPoolExecutor <https://github.com/apache/dubbo/blob/ae514c0f8a726268b9a42fe1903e1f59d6d24c3f/dubbo-common/src/main/java/org/apache/dubbo/common/threadpool/support/eager/EagerThreadPoolExecutor.java#L30> 、TaskQueue <https://github.com/apache/dubbo/blob/master/dubbo-common/src/main/java/org/apache/dubbo/common/threadpool/support/eager/TaskQueue.java>

这里贴一下重点代码。

1）TaskQueue 自定义阻塞队列。

Java 复制代码

```java
public class TaskQueue<R extends Runnable> extends LinkedBlockingQueue<Runn
        ...
    // 队列中持有线程池的引用
    private EagerThreadPoolExecutor executor;

    public TaskQueue(int capacity) {
        super(capacity);
    }

    public void setExecutor(EagerThreadPoolExecutor exec) {
        executor = exec;
    }

    @Override
    public boolean offer(Runnable runnable) {
                ...
        // 获取线程池中线程数
        int currentPoolThreadSize = executor.getPoolSize();
        // 如果有核心线程正在空闲，将任务加入阻塞队列，由核心线程进行处理任务
        if (executor.getSubmittedTaskCount() < currentPoolThreadSize) {
            return super.offer(runnable);
        }

        /**
         *【重点】当前线程池线程数量小于最大线程数
         * 返回false，根据线程池源码，会创建非核心线程
         */
        if (currentPoolThreadSize < executor.getMaximumPoolSize()) {
            return false;
        }

        // 如果当前线程池数量大于最大线程数，任务加入阻塞队列
        return super.offer(runnable);
    }
}
```

存在一个疑点，**getSubmittedTaskCount()** 是如何获取提交任务数量的?

这里就需要看一下 **EagerThreadPoolExecutor** 实现了，也比较简单，只是 **重写了线程池的两个方法: afterExecute()、execute()。**

2）EagerThreadPoolExecutor 封装快速消费线程池。

Java | 复制代码

```java
public class EagerThreadPoolExecutor extends ThreadPoolExecutor {

    /**
     * task count
     */
    private final AtomicInteger submittedTaskCount = new AtomicInteger(0);

    /**
     * @return current tasks which are executed
     */
    public int getSubmittedTaskCount() {
        return submittedTaskCount.get();
    }

    @Override
    protected void afterExecute(Runnable r, Throwable t) {
        submittedTaskCount.decrementAndGet();
    }

    @Override
    public void execute(Runnable command) {
        if (command == null) {
            throw new NullPointerException();
        }
        // do not increment in method beforeExecute!
        submittedTaskCount.incrementAndGet();
        try {
            super.execute(command);
        } catch (RejectedExecutionException rx) {
            // retry to offer the task into queue.
            final TaskQueue queue = (TaskQueue) super.getQueue();
            try {
                if (!queue.retryOffer(command, 0, TimeUnit.MILLISECONDS)) {
                    submittedTaskCount.decrementAndGet();
                    throw new RejectedExecutionException("Queue capacity is
                }
            } catch (InterruptedException x) {
                submittedTaskCount.decrementAndGet();
                throw new RejectedExecutionException(x);
            }
        } catch (Throwable t) {
            // decrease any way
            submittedTaskCount.decrementAndGet();
            throw t;
        }
    }
}
```

EagerThreadPoolExecutor 继承了 ThreadPoolExecutor，在 execute() 上做了个性化设计。

并在线程池内新增了一个任务数量的字段，是一个原子类，添加任务时自增，任务异常及结束时递减。

这样就能保证 **TaskQueue#offer(Runnable runnable)** 做出逻辑处理。