Java 线程池的学习

作者：gitsilence
原文链接：https://ld246.com/article/1622557526714
来源网站：链滴

Java线程池七个参数详解

参考地址：

https://blog.csdn.net/ye17186/article/details/89467919

https://blog.csdn.net/u010648555/article/details/106137206

corePoolSize 线程池核心线程大小

线程池中会维护一个最小的线程数量

即使这些线程处于空闲状态，他们也不会销毁

除非设置了allowCoreThreadTimeOut

maximumPoolSize 线程池最大线程数量

一个被提交到线程池以后，首先会找有没有空闲存活线程，如果有则直接将
任务交给空闲线程执行；如果没有则会缓存到工作队列中，如果队列满了，
才会创建一个新线程，这个新线程会处理**最新提交的任务。**

线程不会限制的去创建线程数量的限制，这个数量即由maximumPoolSize指定

keepAliveTime空闲线程存活时间

一个线程如果处于空闲状态，并且当前的线程数量大于corePoolSize，
那么在指定时间后，这个空闲线程会被销毁，这里的指定时间由keepAliveTime设定

unit 空闲线程 存活时间单位

keepAliveTime的计量单位，TimeUnit.SECONDS

workQueue 工作队列

新任务被提交后，会先进入到此工作队列中，任务调度时再从队列中取任务

jdk中提供了四种工作队列：

ArrayBlockingQueue

基于数组的有界阻塞队列，按FIFO（先入先出）排序。新任务进来，会放到该队列的队尾。

有界的数组可以防止资源耗尽问题。

当线程池中的线程数量达到corePoolSize后，再有新任务进来，则会将任务放入改队列的队尾，等待调度。

如果队列已经是满的，则创建一个新线程，如果线程数量已经达到maximumPoolSize，则会执行拒策略。

LinkedBlockingQueue

基于链表的无界阻塞队列（其实最大容量为Integer.MAX = 2147483647），按照FIFO排序。

由于该队列的近似无界性，当线程池中线程达到corePoolSize后，再有新任务进来，

会一直存入该队列，而不会去创建线程知道maximumPoolSize，因此使用该工作队列时，

参数maximumPoolSize其实是不起作用的。

经测试，当我们给 new LinkedBlockingQueue(3) 初始容量时，

maximumPoolSize还是生效的。

SynchronousQueue

一个不缓存任务的阻塞队列，生产者放入一个任务必须等到消费者取出这个任务。

也就是说新任务进来时，不会缓存，而是直接被调度执行该任务，如果没有可用线程，

则创建新线程，如果线程数量达到maximumPoolSize，则执行拒绝策略。

PriorityBlockingQueue

具有优先级的无界阻塞队列，优先级通过参数Comparator实现

threadFactory 线程工厂

创建一个新线程时使用的工厂，可以用来设定线程名、是否为daemon线程

三种方式创建ThreadFactory设置线程名称

1、Spring框架提供的CustomizableThreadFactory

2、Google guava工具类提供的ThreadFactoryBuilder，使用链式方式创建

3、Apache common-lang3提供的 BasicThreadFactory

handler 拒绝策略

当工作队列中的任务已达到最大限制，并且线程池中的线程量也达到最大限制，

这是如果有新任务提交进来，该如何处理呢。这里的拒绝策略，就是解决这个问题的。

场景：

- 核心线程数：2
- 最大线程数：3
- 队列容量数：2

假如一共提交了6个任务（FIFO），当前是否存活线程，如果没有线程，会创建2个线程来处理任务，

然后将后面的2个任务放到队列中。

当第5个任务提交时，没有存活的空闲线程、队列容量也满了，就会去创建新线程，此时的新创建线会执行最新提交的任务，也就是第5个任务会先被执行。 但是线程总数要保证 <= 最大线程数。

当第6个任务提交时，没有存活的空闲线程、队列容量也满了，线程总数达到了最大线程数，此时就执行拒绝策略。

jdk中提供了4种拒绝策略。

1、CallerRunsPolicy

该策略下，在调用者线程中直接执行被拒绝任务的run方法，除非线程池已经shutdown，则直接抛弃务。

2、AbortPolicy

该策略下，直接丢弃任务，并抛出RejectedExecutionException异常。

3、DisabledPolicy

该策略下，直接丢弃任务，什么都不做。

4、DiscardOldestPolicy

该策略下，抛弃进入队列最早的那个任务，然后尝试把这次拒绝的任务放入队列。

代码实操

ThreadTask

```
package cn.lacknb.test.threadPools;


import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 12:10 下午
```

```
 **/
public class ThreadTask implements Runnable {

    private String task;

    public ThreadTask (String task) {
        this.task = task;
    }

    @Override
    public void run() {
        System.out.println(task + " ===> 当前执行线程为：" + Thread.currentThread().getName())

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

ThreadTask02

```
package cn.lacknb.test.threadPools;


import java.util.concurrent.Callable;
import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 12:10 下午
 **/
public class ThreadTask02 implements Callable<String> {

    private String task;

    public ThreadTask02(String task) {
        this.task = task;
    }

    @Override
    public String call() {
        System.out.println(task + " ===> 当前执行线程为：" + Thread.currentThread().getName())

        try {
            TimeUnit.SECONDS.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return task + " ------>>> 线程执行完毕";
    }
```

```
}
```

ArrayBlockingQueue

```
package cn.lacknb.test.threadPools;

import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 5:14 下午
 **/
public class ThreadPoolArrayBlockingQueue {

    public static void main(String[] args) {
        // 这里ArrayBlockingQueue是有界的，所以要给初始容量
        ThreadPoolExecutor pool = new ThreadPoolExecutor(2, 3, 60L,
                TimeUnit.SECONDS, new ArrayBlockingQueue<>(3));
        for (int i = 0; i < 6; i++) {
            pool.execute(new ThreadTask("任务-0" + (i + 1)));
        }
        // 关闭线程池
        pool.shutdown();
    }

}
```

LinkedBlockingQueue

```
package cn.lacknb.test.threadPools;

import java.util.concurrent.*;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 12:11 下午
 **/
public class ThreadPoolLinkedBlockingQueue {

    /**
     * corePoolSize: 核心线程大小
     * maximumPoolSize: 线程池最大线程数量
     * keepAliveTime: 空闲线程存活时间
     * unit：空闲线程存活时间单位
     * workQueue：工作队列
     * threadFactory：线程工厂
     * handler：拒绝策略
     *
```

```
     * @param args
     */
    public static void main(String[] args) {
        ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(2, 3,
                60L, TimeUnit.SECONDS, new LinkedBlockingQueue<>(3));
        /*
         * 核心线程：2
         * 最大线程数：3
         * 阻塞队列：3
         * 现在提交6个任务，当任务数达到核心线程数时，会将剩余的任务放到阻塞队列中
         * 第5个任务放到阻塞队列后，阻塞队列满了。
         * 当第6个任务来了，由于阻塞队列满了，就会去创建新的线程执行 第6个任务
         * 但是创建的线程数 <= 最大线程数 - 核心线程数
         * 当任务执行完成后，当创建的线程一直处于空闲状态，时间达到 keepAliveTime
         * 这些线程将会被销毁。
         * */
        for (int i = 0; i < 7; i++) {
            // execute 方法，提交线程任务，不会阻塞，没有返回值
            threadPoolExecutor.execute(new ThreadTask("任务-0" + (i + 1)));
        }

//        for (int i = 0; i < 6; i++) {
//            // submit 方法，提交线程任务，会阻塞 直到有返回值。
//            Future<String> submit = threadPoolExecutor.submit(new ThreadTask02("任务-0" + (i
+ 1)));
//            try {
//                System.out.println(submit.get());
//            } catch (InterruptedException e) {
//                e.printStackTrace();
//            } catch (ExecutionException e) {
//                e.printStackTrace();
//            }
//        }
        // 关闭线程池状态，如果有线程正在执行，则等待线程执行完后关闭
        threadPoolExecutor.shutdown();
        // 关闭线程池状态，如果有线程正在执行，停止执行
//        threadPoolExecutor.shutdownNow();
    }

}
```

PriorityBlockingQueue

```
package cn.lacknb.test.threadPools;

import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 5:25 下午
```

```
 **/
public class ThreadPoolPriorityBlockingQueue {

    public static void main(String[] args) {
        // PriorityBlockingQueue 具有优先级的无界队列，通过参数Comparator实现
        ThreadPoolExecutor pool = new ThreadPoolExecutor(2, 3, 60L,
                TimeUnit.SECONDS, new PriorityBlockingQueue<>());
        for (int i = 0; i < 6; i++) {
            pool.execute(new ThreadTask("任务-0" + (i + 1)));
        }
        pool.shutdown();
    }

}
```

SynchronousQueue

```
package cn.lacknb.test.threadPools;

import java.util.concurrent.SynchronousQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/5/30 5:18 下午
 **/
public class ThreadPoolSynchronousQueue {

    public static void main(String[] args) {
        // SynchronousQueue 不缓存任务，生产者放入一个任务必须等到消费者取出任务
        // 也就是说新任务进来时，不会缓存，而是直接被调度执行该任务，如果没有可用线程则创建
程
        // 如果线程数量达到maximumPoolSize，则执行拒绝策略
        ThreadPoolExecutor pool = new ThreadPoolExecutor(2, 3, 60L,
                TimeUnit.SECONDS, new SynchronousQueue<>());
        // 所以当这里的 提交任务 大于3的时候，就会执行拒绝策略。
        for (int i = 0; i < 3; i++) {
            pool.execute(new ThreadTask("任务-0" + (i + 1)));
        }
        pool.shutdown();
    }

}
```

ThreadPoolFactory

```
package cn.lacknb.test.threadPools;

import org.apache.commons.lang3.concurrent.BasicThreadFactory;
import java.util.concurrent.ArrayBlockingQueue;
```

```
import java.util.concurrent.ThreadFactory;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

/**
 * @author: gitsilence
 * @description:
 * @date: 2021/6/1 9:48 下午
 **/
public class ThreadPoolFactory {
    public static void main(String[] args) {

        // 原始
//        ThreadFactory factory = runnable -> {
//            Thread thread = new Thread(runnable);
//            thread.setName("thread-pool");
//            return thread;
//        };
        // Spring 框架提供的，这里名字为前缀
//        CustomizableThreadFactory factory = new CustomizableThreadFactory("spring-thread
pool-");

        // Google guava工具类提供的，这里名字 为完整的名字
//        ThreadFactory factory = new ThreadFactoryBuilder().setNameFormat("guava-thread-p
ol").build();

        // Apache commons-lang3 提供的，这里名字 为完整的名字
        ThreadFactory factory = new BasicThreadFactory.Builder().namingPattern("basicThreadFa
tory-pool").build();

        ThreadPoolExecutor pool = new ThreadPoolExecutor(2, 3, 60L,
                TimeUnit.SECONDS, new ArrayBlockingQueue<>(2), factory);
        for (int i = 0; i < 5; i++) {
            pool.execute(new ThreadTask("任务-0" + (i + 1)));
        }
        pool.shutdown();
    }
}


        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>23.0</version>
        </dependency>
```