在J2SE(TM)5.0 中，Doug Lea 编写了一个优秀的并发实用程序开放源码库 util.concurrent，它包括互斥、信号量、诸如在并发访问下执行得很好的队列和散列表之类集合类以及几个工作队列实现。该包中的 PooledExecutor 类是一种有效的、广泛使用的以工作队列为基础的线程池的正确实现。Util.concurrent 定义一个 Executor 接口，以异步地执行 Runnable，另外还定义了 Executor 的几个实现，它们具有不同的调度特征。将一个任务排入 executor 的队列非常简单：

Executor executor = new QueuedExecutor();

…

Runnable runnable = … ;

executor.execute(runnable);

PooledExecutor 是一个复杂的线程池实现，它不但提供工作线程（worker thread）池中任务的调度，而且还可灵活地调整池的大小，同时还提供了线程生命周期管理，这个实现可以限制工作队列中任务的数目，以防止队列中的任务耗尽所有可用内存，另外还提供了多种可用的关闭和饱和度策略（阻塞、废弃、抛出、废弃最老的、在调用者中运行等）。所有的 Executor 实现为您管理线程的创建和销毁，包括当关闭 executor 时，关闭所有线程，

二、线程池的使用

线程池类为 java.util.concurrent.ThreadPoolExecutor，常用构造方法为：

ThreadPoolExecutor(int corePoolSize,

int maximumPoolSize,

long keepAliveTime, TimeUnit unit,

BlockingQueue<Runnable> workQueue,

RejectedExecutionHandler handler)

• corePoolSize

线程池维护线程的最少数量

• maximumPoolSiz

线程池维护线程的最大数量

• keepAliveTime

线程池维护线程所允许的空闲时间

• unit

线程池维护线程所允许的空闲时间的单位

• workQueue

线程池所使用的缓冲队列

• handler

线程池对拒绝任务的处理策略

一个任务通过 execute(Runnable)方法被添加到线程池，任务就是一个 Runnable类型的对象，任务的执行方法就是 Runnable类型对象的run()方法。

当一个任务通过execute(Runnable)方法欲添加到线程池时：

* 如果此时线程池中的数量小于corePoolSize，即使线程池中的线程都处于空闲状态，也要创建新的线程来处理被添加的任务。
* 如果此时线程池中的数量等于 corePoolSize，但是缓冲队列 workQueue未满，那么任务被放入缓冲队列。
* 如果此时线程池中的数量大于corePoolSize，缓冲队列workQueue满，并且线程池中的数量小于maximumPoolSize，建新的线程来处理被添加的任务。
* 如果此时线程池中的数量大于corePoolSize，缓冲队列workQueue满，并且线程池中的数量等于maximumPoolSize，那么通过 handler所指定的策略来处理此任务。

也就是：处理任务的优先级为：

核心线程corePoolSize、任务队列workQueue、最大线程maximumPoolSize，如果三者都满了，使用handler处理被拒绝的任务。

当线程池中的线程数量大于 corePoolSize时，如果某线程空闲时间超过keepAliveTime，线程将被终止。这样，线程池可以动态的调整池中的线程数。

unit可选的参数为java.util.concurrent.TimeUnit中的几个静态属性：

NANOSECONDS、MICROSECONDS、MILLISECONDS、SECONDS。

workQueue我常用的是：java.util.concurrent.ArrayBlockingQueue

handler有四个选择：

• ThreadPoolExecutor.AbortPolicy()

抛出java.util.concurrent.RejectedExecutionException异常

• ThreadPoolExecutor.CallerRunsPolicy()

重试添加当前的任务，他会自动重复调用execute()方法

• ThreadPoolExecutor.DiscardOldestPolicy()

抛弃旧的任务

• ThreadPoolExecutor.DiscardPolicy()

抛弃当前的任务

用法举例

package cn.simplelife.exercise;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.ThreadPoolExecutor;

import java.util.concurrent.TimeUnit;

public class TestThreadPool {

private static int produceTaskSleepTime = 2;

public static void main(String[] args) {

//构造一个线程池

ThreadPoolExecutor producerPool = new ThreadPoolExecutor(1, 1, 0,

TimeUnit.SECONDS, new ArrayBlockingQueue(3),

new ThreadPoolExecutor.DiscardOldestPolicy());

//每隔produceTaskSleepTime的时间向线程池派送一个任务。

int i=1;

while(true){

try {

Thread.sleep(produceTaskSleepTime);

String task = “task@ ” + i;

System.out.println(”put ” + task);

producerPool.execute(new ThreadPoolTask(task));

i++;

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

package cn.simplelife.exercise;

import java.io.Serializable;

public class ThreadPoolTask implements Runnable,Serializable{

//JDK1.5中，每个实现Serializable接口的类都推荐声明这样的一个ID

private static final long serialVersionUID = 0;

private static int consumeTaskSleepTime = 2000;

private Object threadPoolTaskData;

ThreadPoolTask(Object tasks){

this.threadPoolTaskData = tasks;

}

//每个任务的执行过程，现在是什么都没做，除了print和sleep，:)

public void run(){

System.out.println(”start ..”+threadPoolTaskData);

try {

//便于观察现象，等待一段时间

Thread.sleep(consumeTaskSleepTime);

} catch (Exception e) {

e.printStackTrace();

}

threadPoolTaskData = null;

}

}

对这两段程序的说明：

1. 在这段程序中，一个任务就是一个Runnable类型的对象，也就是一个ThreadPoolTask类型的对象。

2. 一般来说任务除了处理方式外，还需要处理的数据，处理的数据通过构造方法传给任务。

3. 在这段程序中，main()方法相当于一个残忍的领导，他派发出许多任务，丢给一个叫 threadPool的任劳任怨的小组来做。

o 这个小组里面队员至少有两个，如果他们两个忙不过来， 任务就被放到任务列表里面。

o 如果积压的任务过多，多到任务列表都装不下(超过3个)的时候，就雇佣新的队员来帮忙。但是基于成本的考虑，不能雇佣太多的队员，至多只能雇佣 4个。

o 如果四个队员都在忙时，再有新的任务， 这个小组就处理不了了，任务就会被通过一种策略来处理，我们的处理方式是不停的派发，直到接受这个任务为止(更残忍！呵呵)。

o 因为队员工作是需要成本的，如果工作很闲，闲到 3SECONDS都没有新的任务了，那么有的队员就会被解雇了，但是，为了小组的正常运转，即使工作再闲，小组的队员也不能少于两个。

4. 通过调整 produceTaskSleepTime和 consumeTaskSleepTime的大小来实现对派发任务和处理任务的速度的控制，改变这两个值就可以观察不同速率下程序的工作情况。

5. 通过调整4中所指的数据，再加上调整任务丢弃策略， 换上其他三种策略，就可以看出不同策略下的不同处理方式。

6. 对于其他的使用方法，参看jdk的帮助，很容易理解和使用。