# java中四种线程池的使用

java通过Executors提供四种线程池，分别为：  
newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。  
newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

## newCachedThreadPool

创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。示例代码如下：

package com.anjz;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

/\*\*

\* @author shuai.ding

\* @date 2017年7月19日下午3:27:20

\*/

public class Test {

public static void main(String[] args) {

ExecutorService cachedThreadPool = Executors.newCachedThreadPool();

for(int i=0;i<10;i++){

final int index=i;

// try {

// Thread.sleep(index \* 1000);

// } catch (InterruptedException e) {

// e.printStackTrace();

// }

cachedThreadPool.execute(new Runnable() {

@Override

public void run() {

System.out.println(Thread.currentThread().getName()+":"+index);

}

});

}

}

}

执行结果为：

pool-1-thread-1:0

pool-1-thread-2:1

pool-1-thread-6:5

pool-1-thread-4:3

pool-1-thread-3:2

pool-1-thread-7:6

pool-1-thread-10:9

pool-1-thread-8:7

pool-1-thread-5:4

pool-1-thread-9:8

创建了10个线程。

去掉代码中的注释：执行结果为：

pool-1-thread-1:0

pool-1-thread-1:1

pool-1-thread-1:2

pool-1-thread-1:3

pool-1-thread-1:4

pool-1-thread-1:5

pool-1-thread-1:6

pool-1-thread-1:7

pool-1-thread-1:8

pool-1-thread-1:9

创建了1个线程。

总结：线程池可以无限大，当执行第二个任务时，第一个任务已经完成，会复用执行第一个任务的线程，而不用每次新建线程。线程可以无上限，同时又可以复用空闲线程，资源利用率比较高。

## **newFixedThreadPool**

创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。示例代码如下：

package com.anjz;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

/\*\*

\* @author shuai.ding

\* @date 2017年7月19日下午3:27:20

\*/

public class Test {

public static void main(String[] args) {

ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);

for(int i=0;i<10;i++){

final int index=i;

fixedThreadPool.execute(new Runnable() {

public void run() {

try {

System.out.println(Thread.currentThread().getName()+":"+index);

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

}

fixedThreadPool.shutdown();

}

}

执行结果：

pool-1-thread-1:0

pool-1-thread-2:1

pool-1-thread-3:2

pool-1-thread-2:3

pool-1-thread-1:4

pool-1-thread-3:5

pool-1-thread-1:6

pool-1-thread-2:7

pool-1-thread-3:8

pool-1-thread-1:9

因为线程池大小为3，每个任务输出index后sleep2秒，所以每两秒打印三个数字。

定长线程池的大小最好根据系统资源进行设置。如Runtime.getRuntime().availableProcessors()

## newScheduledThreadPool

创建一个定长线程池，支持定时及周期性任务执行。延迟执行示例代码如下：

package com.anjz;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.TimeUnit;

/\*\*

\* @author shuai.ding

\* @date 2017年7月19日下午3:27:20

\*/

public class Test {

public static void main(String[] args) {

ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool(5);

scheduledThreadPool.schedule(new Runnable() {

public void run() {

System.out.println("delay 3 seconds");

}

}, 3, TimeUnit.SECONDS);

scheduledThreadPool.shutdown();

}

}

表示延迟3秒执行。

定期执行示例代码如下：

package com.anjz;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.TimeUnit;

/\*\*

\* @author shuai.ding

\* @date 2017年7月19日下午3:27:20

\*/

public class Test {

public static void main(String[] args) {

ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool(5);

scheduledThreadPool.scheduleAtFixedRate(new Runnable() {

public void run() {

System.out.println(Thread.currentThread().getName()+":"+"delay 1 seconds, and excute every 3 seconds");

}

}, 1,3, TimeUnit.SECONDS);

}

}

表示延迟1秒后每3秒执行一次。

## **newSingleThreadExecutor**

创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。示例代码如下:

package com.anjz;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

/\*\*

\* @author shuai.ding

\* @date 2017年7月19日下午3:27:20

\*/

public class Test {

public static void main(String[] args) {

ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();

for(int i=0;i<10;i++){

final int index=i;

singleThreadExecutor.execute(new Runnable() {

public void run() {

try {

System.out.println(Thread.currentThread().getName()+":"+index);

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

}

}

}

结果依次输出，相当于顺序执行各个任务。

转载地址：http://cuisuqiang.iteye.com/blog/2019372