链滴

Java 线程池笔记

作者：helly
原文链接：https://ld246.com/article/1510799294606
来源网站：链滴

线程池

架构：

executor接口　　executors类

ExecutorService接口

AbstractExecutorService抽象类 ScheduledExecutorService接口

ThreadPoolExecutor类 ScheduledThreadPoolExecutor类

Executor接口

将任务的提交与任务的执行分离

void execute(Runnable command)

ExecutorService接口（继承Executor接口）

为executor接口服务

submit

invoke

shutdown

AbstractExecutorService抽象类（实现ExecutorService接口）

继承ExecutorService接口，实现该接口的方法

ThreadPoolExecutor类（继承ThreadPoolExecutor类）

```
//-----------------------------------------------------------------

private final BlockingQueue\<Runnable> workQueue; // 阻塞队列
private final ReentrantLock mainLock = new ReentrantLock(); // 互斥锁，锁住的是线程池
private final HashSet workers = new HashSet(); // 线程集合
private final Condition termination = mainLock.newCondition(); // “终止条件”，与“mainLoc
”绑定

//-----------------------------------------------------------------

private volatile int corePoolSize;
private volatile int maximumPoolSize;
private volatile boolean allowCoreThreadTimeOut; // 是否允许线程在空闲状态时，仍然能够存活
private volatile long keepAliveTime;
private volatile ThreadFactory threadFactory;
private volatile RejectedExecutionHandler handler; // handler是RejectedExecutionHandler类
。它是"线程池拒绝策略"的句柄，也就是说"当某任务添加到线程池中，而线程池拒绝该任务时，线程
会通过handler进行相应的处理
// ThreadPoolExecutor.DiscardPolicy：无法执行的任务将被删除。
// ThreadPoolExecutor.AbortPolicy：默认策略，任务遭到拒绝，直接抛出异常RejectedExecutionE
ception。
// ThreadPoolExecutor.CallerRunsPolicy：让调用者所在的线程来运行该任务。
// ThreadPoolExecutor.DiscardOldestPolicy：将位于阻塞队列头部的任务删除，然后尝试重新执
任务（失败的话重复此过程）。

//-----------------------------------------------------------------
```

```
private int largestPoolSize; // 线程池中线程数量曾经达到过的最大值
private long completedTaskCount; // 已完成任务数量

//-----------------------------------------------------------------

// 线程池的生命周期
RUNNING->SHUTDOWN（调用shutdown()方法后）、STOP（调用shutdwonNow()方法后->TIG
ING（所有任务已终止）->TERMINATED（调用terminated()方法后）

//-----------------------------------------------------------------

// 构造器
ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit un
t, BlockingQueue\<Runnable> workQueue, ThreadFactory threadFactory, RejectedExecutionH
ndler handler) {
  // threadFactory来自Executors.defaultThreadFactory()
}

//-----------------------------------------------------------------

// 添加任务到线程池
public void execute(Runnable command) {
  // 情况1：
  // 情况2：
  // 情况3：
}

// 添加任务到线程池，实际上还是通过调用execute方法
public Future submit(Runnable task) {

}

// 添加任务到线程池，并创建一个线程启动该任务
// core为true的话，则以corePoolSize为界限，若"线程池中已有任务数量>=corePoolSize"，则返回
alse；core为false的话，则以maximumPoolSize为界限，若"线程池中已有任务数量>=maximumP
olSize"，则返回false。
private boolean addWorker(Runnable firstTask, boolean core) {

}

//-----------------------------------------------------------------

// 关闭线程池
public void shutdown() {

}

public void shutdownNow() {

}
```

ScheduledExecutorService接口（继承Executor接口）

相当于提供了"延时"和"周期执行"功能的ExecutorService

ScheduledThreadPoolExecutor类（实现ScheduledExecutorService接口）

ScheduledThreadPoolExecutor

Executors类

是个静态工厂类。它通过静态工厂方法返回ExecutorService、ScheduledExecutorService、Thread actory 和 Callable 等类的对象。

```
public static ExecutorService newFixedThreadPool(int nThreads) {
  return new ThreadPoolExecutor(nThreads, nThreads, 0L, TimeUnit.MILLISECONDS, new Link
dBlockingQueue\<Runnable>());
}

public static ThreadFactory defaultThreadFactory() {
  return new DefaultThreadFactory();
}
```

当我们需要获取线程的执行结果时，就需要用到Callable和Future。Callable用于产生结果（将Callab e的实现类的对象交给线程池去执行），Future用于获取结果。

Callable接口：

```
V call();
```

Future接口：

```
V get() throws InterruptedException, ExecutionException;
V get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, Timeout
xception;
....
```

RunnableFuture（继承了Future接口和Runnable接口）

```
public interface RunnableFuture\<V> extends Runnable, Future\<V> {

}
```

FutureTask类（简介实现RunnableFuture接口）

参考：

java.util.concurrrent包源码

http://www.cnblogs.com/skywang12345/p/java_threads_category.html