目录

[什么是线程池？为什么要用线程池？ 1](#_Toc514704688)

[实现一个我们自己的线程池 1](#_Toc514704689)

[JDK中的线程池和工作机制 1](#_Toc514704690)

[线程池的创建 2](#_Toc514704691)

[各个参数含义 2](#_Toc514704692)

[提交任务 2](#_Toc514704693)

[关闭线程池 2](#_Toc514704694)

[合理配置线程池 2](#_Toc514704695)

[预定义的线程池 2](#_Toc514704696)

[FixedThreadPool 2](#_Toc514704697)

[SingleThreadExecutor 2](#_Toc514704698)

[CachedThreadPool 2](#_Toc514704699)

[WorkStealingPool（JDK7以后） 3](#_Toc514704700)

[ScheduledThreadPoolExecutor 3](#_Toc514704701)

[Executor框架 3](#_Toc514704702)

[了解CompletionService 3](#_Toc514704703)

**线程池**

# 什么是线程池？为什么要用线程池？

1. 降低资源的消耗。降低线程创建和销毁的资源消耗；
2. 提高响应速度：线程的创建时间为T1，执行时间T2,销毁时间T3，免去T1和T3的时间
3. 提高线程的可管理性。

# 实现一个我们自己的线程池

1. 线程必须在池子已经创建好了，并且可以保持住，要有容器保存多个线程；

2、线程还要能够接受外部的任务，运行这个任务。容器保持这个来不及运行的任务.

# JDK中的线程池和工作机制

## 线程池的创建

ThreadPoolExecutor，jdk所有线程池实现的父类

### 各个参数含义

**int** corePoolSize ：线程池中核心线程数，< corePoolSize ，就会创建新线程，= corePoolSize ，这个任务就会保存到BlockingQueue，如果调用prestartAllCoreThreads（）方法就会一次性的启动corePoolSize 个数的线程。

**int** maximumPoolSize, 允许的最大线程数，BlockingQueue也满了，< maximumPoolSize时候就会再次创建新的线程

**long** keepAliveTime, 线程空闲下来后，存活的时间，这个参数只在> corePoolSize才有用

TimeUnit unit, 存活时间的单位值

BlockingQueue<Runnable> workQueue, 保存任务的阻塞队列

ThreadFactory threadFactory, 创建线程的工厂，给新建的线程赋予名字

RejectedExecutionHandler handler ：饱和策略

AbortPolicy ：直接抛出异常，默认；

CallerRunsPolicy：用调用者所在的线程来执行任务

DiscardOldestPolicy：丢弃阻塞队列里最老的任务，队列里最靠前的任务

DiscardPolicy ：当前任务直接丢弃

实现自己的饱和策略，实现RejectedExecutionHandler接口即可

## 提交任务

execute(Runnable command) 不需要返回

Future<T> submit(Callable<T> task) 需要返回

## 关闭线程池

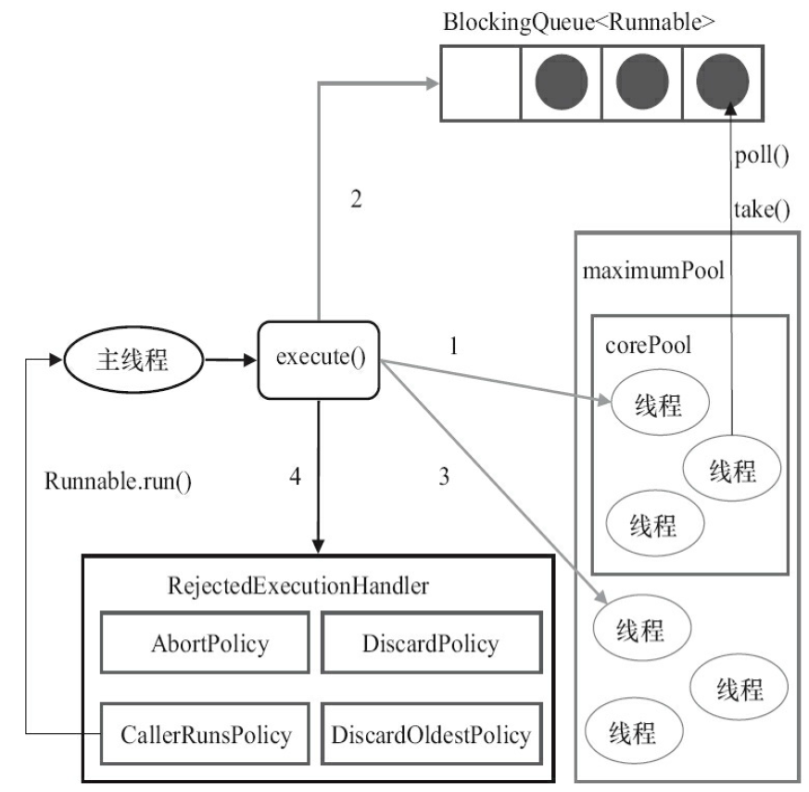
shutdown(),shutdownNow();

shutdownNow():设置线程池的状态，还会尝试停止正在运行或者暂停任务的线程

shutdown()设置线程池的状态，只会中断所有没有执行任务的线程

## 工作机制





# 合理配置线程池

根据任务的性质来：计算密集型（CPU），IO密集型，混合型

计算密集型：加密，大数分解，正则…….， 线程数适当小一点，最大推荐：机器的Cpu核心数+1，为什么+1，防止页缺失，(机器的Cpu核心=Runtime.*getRuntime*().availableProcessors();)

IO密集型：读取文件，数据库连接，网络通讯, 线程数适当大一点，机器的Cpu核心数\*2,

混合型：尽量拆分，IO密集型>>计算密集型，拆分意义不大，IO密集型~计算密集型

队列的选择上，应该使用有界，无界队列可能会导致内存溢出，OOM

# 预定义的线程池

## FixedThreadPool

创建固定线程数量的，适用于负载较重的服务器，使用了无界队列

## SingleThreadExecutor

创建单个线程，需要顺序保证执行任务，不会有多个线程活动，使用了无界队列

## CachedThreadPool

会根据需要来创建新线程的，执行很多短期异步任务的程序，使用了SynchronousQueue

## WorkStealingPool（JDK7以后）

基于ForkJoinPool实现

## ScheduledThreadPoolExecutor

需要定期执行周期任务，Timer不建议使用了。

newSingleThreadScheduledExecutor：只包含一个线程，只需要单个线程执行周期任务，保证顺序的执行各个任务

newScheduledThreadPool 可以包含多个线程的，线程执行周期任务，适度控制后台线程数量的时候

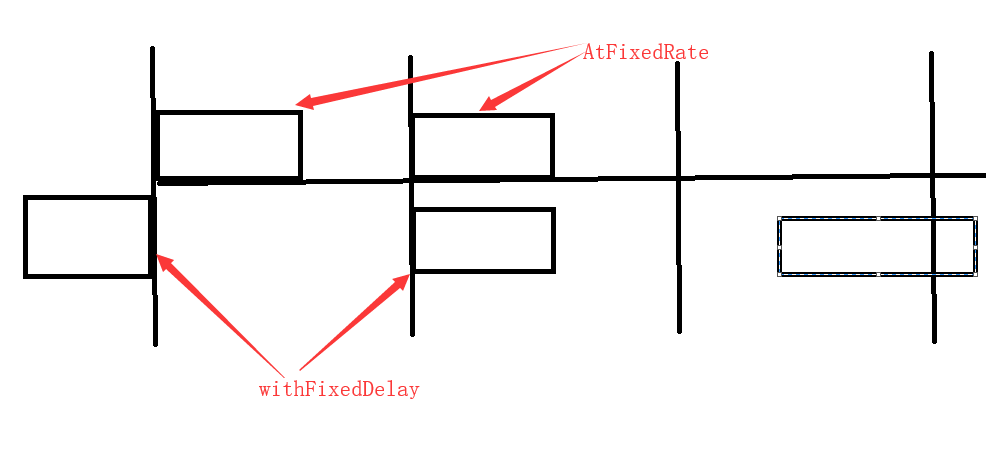
方法说明：

schedule：只执行一次，任务还可以延时执行

scheduleAtFixedRate：提交固定时间间隔的任务

scheduleWithFixedDelay：提交固定延时间隔执行的任务

两者的区别：



scheduleAtFixedRate任务超时：

规定60s执行一次，有任务执行了80S，下个任务马上开始执行

第一个任务 时长 80s，第二个任务20s，第三个任务 50s

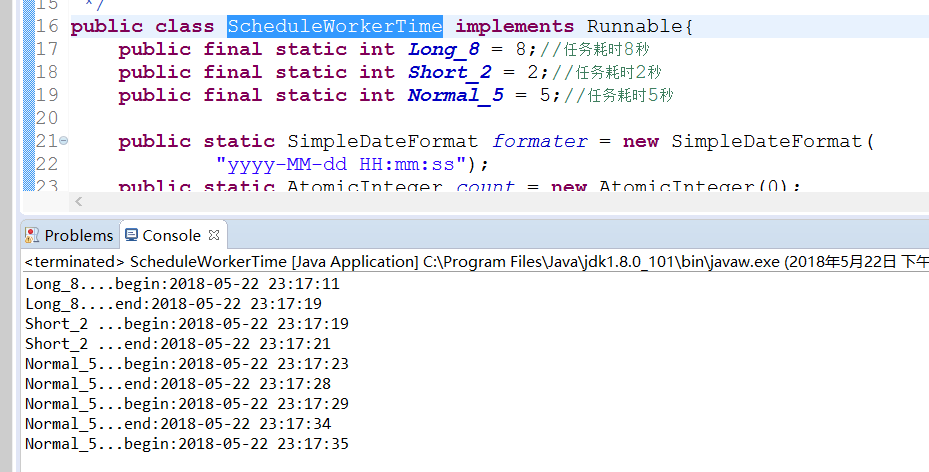
第一个任务第0秒开始，第80S结束；

第二个任务第80s开始，在第100秒结束；

第三个任务第120s秒开始，170秒结束

第四个任务从180s开始

参加代码：ScheduleWorkerTime类，执行效果如图：



建议在提交给ScheduledThreadPoolExecutor的任务要住catch异常。

# Executor框架

# 了解CompletionService