# 线程池原理

首先定义指定数量的核心线程来执行任务

当核心线程到达最大时，将任务放到工作队列中等待执行

进行添加任务，超过了工作队列的最大数量时

创建非核心线程来执行任务

非核心线程到达最大数量时，使用拒绝策略来应对继续添加的任务

核心线程不会被回收

非核心线程会定时被回收

构造函数参数

int corePoolSize,核心线程数量，任务执行完成后也会存在，除非设置了线程消失时间

int maximumPoolSize,非核心线程数，线程池中最大允许的线程数量 ，线程会定期被回收

long keepAliveTime,线程空闲时等待的时间，如果等待超过这个时间线程就会被终止

TimeUnit unit,时间单位

BlockingQueue<Runnable> workQueue：任务队列（队列是阻塞的，能够保证取出和添加的线程安全性）

ThreadFactory threadFactory,线程工厂，创建工作线程；DefaultThreadFactory默认线程工厂

RejectedExecutionHandler handler：拒绝策略（多余的任务处理器）

# 4中线程池，应用场景

## Executors.newFixedThreadPool

创建固定数量的工作线程ThreadPoolExecutor线程池，就是执行ThreadPoolExecutor的构造函数

CorePoolSize,maximumPoolSize的数量是一样的，

Keepalivetime：空闲线程等待新任务的时间，超过这个时间，线程就会被终止，设置为0，多余的空闲线程会立刻被终止

任务队列是LinkedBlockingQueue，无界队列；

无界工作队列的影响

1. 当线程数量达到corePoolSize后，新任务就会添加到无界队列中，线程池中的线程数量不会超过corePoolSize

2. maximumPoolSize是一个无效的参数，因为任务队列不会满

3. keepAliveTime是一个无效的参数

4. 使用了无界队列，运行中的拒绝策略不会发送

## Executors.newCachedThreadPool

大小无界的线程池，可以进行线程的自动回收。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；

CorePoolSize为0，

maximumPoolSize值为Integer的最大值，表示可以创建Integer.MAX\_VALUE个线程

keepAliveTime:线程空闲等待时间为60秒，超过60秒线程就会终止，时间单位为秒

SynchronousQueue：内部只包含一个元素的队列

## Executors.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。

corePoolSize，maximumPoolSize的值都为1，那么就只会创建一个线程来执行任务，也就是说任务是按顺序一个一个的执行

## Executors.newScheduledThreadPool

创建一个定长线程池，支持定时及周期性任务执行，适用于后台线程执行周期任务，

同时为了满足后台资源管理的需求而限制后台线程的数量的应用场景

# 线程池好处

1、 降低资源消耗：重复利用创建的线程降低线程创建和销毁带来的性能消耗

2、 提高响应速度：当任务到达时，任务不需要等待创建线程就能被执行

3、 提高线程的可管理性：线程是稀缺资源，如果无限制的创建线程不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一分配，调优，监控

# 拒绝策略

RejectedExecutionHandler:拒绝策略，当线程池和队列都满了的时候，不能在创建线程或者往任务队列中添加任务，

那么就采取策略来处理新提交的任务，默认是AbortPolicy，表示无法处理新任务并抛出异常

DiscardOldestPolicy s;// 丢弃队列中的最近一个任务，并执行当前任务

DiscardPolicy p;// 不处理，不丢弃