# 线程池的作用

首先通过线程池中线程的重用，减少创建和销毁线程的性能开销。

其次，能控制线程池中的并发数，否则会因为大量的线程争夺CPU资源造成阻塞。

最后，线程池能够对线程进行管理，比如使用ScheduledThreadPool来设置延迟N秒后执行任务，并且每隔M秒循环执行一次。

线程池中的真正实现者——ThreadPoolExecutor

# 原理

Executor作为一个接口，它的具体实现就是ThreadPoolExecutor。

Android中的线程池都是直接或间接通过配置ThreadPoolExecutor来实现不同特性的线程池。

# ThreadPoolExecutor执行任务时的心路历程

（1）当currentSize<corePoolSize时，没什么好说的，直接启动一个核心线程并执行任务。

（2）当currentSize>=corePoolSize、并且workQueue未满时，添加进来的任务会被安排到workQueue中等待执行。

（3）当workQueue已满，但是currentSize<maximumPoolSize时，会立即开启一个非核心线程来执行任务。

（4）当currentSize>=corePoolSize、workQueue已满、并且currentSize>maximumPoolSize时，调用handler默认抛出RejectExecutionExpection异常。

# Android中的四类线程池

**1 FixThreadPool（一堆人排队上公厕）**

Android中最常见的四类具有不同特性的线程池分别为FixThreadPool、CachedThreadPool、ScheduleThreadPool以及SingleThreadExecutor。

public static ExecutorService newFixThreadPool(int nThreads){

return new ThreadPoolExecutor(nThreads, nThreads, 0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());

}

//使用

Executors.newFixThreadPool(5).execute(r);

1. 从配置参数来看，FixThreadPool只有核心线程，并且数量固定的，也不会被回收，所有线程都活动时，因为队列没有限制大小，新任务会等待执行。
2. 由于线程不会回收，FixThreadPool会更快地响应外界请求，这也很容易理解，就好像有人突然想上厕所，公厕不是现用现建的

**2 SingleThreadPool（公厕里只有一个坑位）**

public static ExecutorService newSingleThreadPool (int nThreads){

return new FinalizableDelegatedExecutorService ( new ThreadPoolExecutor (1, 1, 0, TimeUnit. MILLISECONDS, new LinkedBlockingQueue<Runnable>()) );

}

//使用

Executors.newSingleThreadPool ().execute(r);

（1）从配置参数可以看出，SingleThreadPool只有一个核心线程，确保所有任务都在同一线程中按顺序完成。因此不需要处理线程同步的问题。

**3 CachedThreadPool（一堆人去一家很大的咖啡馆喝咖啡）**

public static ExecutorService newCachedThreadPool(int nThreads){

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE, 60L, TimeUnit. SECONDS, new SynchronousQueue<Runnable>());

}

//使用

Executors.newCachedThreadPool().execute(r);

（1）CachedThreadPool只有非核心线程，最大线程数非常大，所有线程都活动时，会为新任务创建新线程，否则利用空闲线程（60s空闲时间，过了就会被回收，所以线程池中有0个线程的可能）处理任务。

（2）任务队列SynchronousQueue相当于一个空集合，导致任何任务都会被立即执行。

（4）比较适合执行大量的耗时较少的任务。喝咖啡人挺多的，喝的时间也不长。

**4 ScheduledThreadPool（4个里面唯一一个有延迟执行和周期重复执行的线程池）**

public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize){

return new ScheduledThreadPoolExecutor(corePoolSize);

}

public ScheduledThreadPoolExecutor(int corePoolSize){

super(corePoolSize, Integer.MAX\_VALUE, 0, NANOSECONDS, new DelayedQueue ());

}

//使用，延迟1秒执行，每隔2秒执行一次Runnable r

Executors. newScheduledThreadPool (5).scheduleAtFixedRate(r, 1000, 2000, TimeUnit.MILLISECONDS);

（1）核心线程数固定，非核心线程（闲着没活干会被立即回收）数没有限制。

（2）从上面代码也可以看出，ScheduledThreadPool主要用于执行定时任务以及有固定周期的重复任务。

# **线程池的submit和execute区别**

execute开启线程执行池中的任务

Submit提交线程执行池中的任务并且返回Future对象

1、接收的参数不一样

2、submit有返回值，而execute没有

3、submit方便Exception处理

Future.get()在执行成功后返回的值是null

Submit的例子：

ExecutorService pool = Executors.newFixedThreadPool(2);

Future future = pool.submit(new RunnableTest("Task2"));

try {

if(future.get()==null){//如果Future's get返回null，任务完成

System.out.println("任务完成");

}

} catch (InterruptedException e) {

} catch (ExecutionException e) {

//否则我们可以看看任务失败的原因是什么

System.out.println(e.getCause().getMessage());

}

# 线程池的关闭

常见：

应用程序结束时：当整个应用程序即将结束时，可以在最后关闭线程池，以确保所有任务都已经完成并且没有新的任务提交。

在调用shutdown()之后，线程池将不再接受新的任务，但会等待已经提交的任务完成执行，直至任务队列为空。

Tips：

如果需要立即关闭线程池，可以使用线程池的shutdownNow()方法。它将尝试立即停止所有的线程，并返回尚未执行的任务列表。

在关闭线程池之前，可以调用线程池的awaitTermination()方法等待一段时间，以确保所有任务都已经完成。这个方法可以等待一定的时间，直到线程池中的所有任务执行完毕，或者超时。