**Java(Android)线程池**

**1、new Thread的弊端**

执行一个异步任务你还只是如下new Thread吗？

new Thread(new Runnable() {

@Override

public void run() {

// TODO Auto-generated method stub

}

}).start();

那你就out太多了，new Thread的弊端如下：

a. 每次new Thread新建对象性能差。

b. 线程缺乏统一管理，可能无限制新建线程，相互之间竞争，及可能占用过多系统资源导致死机或oom。

c. 缺乏更多功能，如定时执行、定期执行、线程中断。

相比new Thread，Java提供的四种线程池的好处在于：

a. 重用存在的线程，减少对象创建、消亡的开销，性能佳。

b. 可有效控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。

c. 提供定时执行、定期执行、单线程、并发数控制等功能。

**2、Java 线程池**

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。

**(1). 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：

Java

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(index);

}

});

}

线程池为无限大，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。

**(2). 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：

ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);

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

final int index = i;

fixedThreadPool.execute(new Runnable() {

@Override

public void run() {

try {

System.out.println(index);

Thread.sleep(2000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

}

因为线程池大小为3，每个任务输出index后sleep 2秒，所以每两秒打印3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()。可参考[PreloadDataCache](http://www.trinea.cn/android/preloaddatacache%e6%94%af%e6%8c%81%e9%a2%84%e5%8f%96%e7%9a%84%e6%95%b0%e6%8d%ae%e7%bc%93%e5%ad%98%ef%bc%8c%e4%bd%bf%e7%94%a8%e7%ae%80%e5%8d%95%ef%bc%8c%e6%94%af%e6%8c%81%e5%a4%9a%e7%a7%8d%e7%bc%93/)。

**(3) 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：

Crayon Syntax Highlighter v2.7.1

Java

ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool(5);

scheduledThreadPool.schedule(new Runnable() {

@Override

public void run() {

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

}

}, 3, TimeUnit.SECONDS);

表示延迟3秒执行。

定期执行示例代码如下：

Crayon Syntax Highlighter v2.7.1

Java

scheduledThreadPool.scheduleAtFixedRate(new Runnable() {

@Override

public void run() {

System.out.println("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。

ScheduledExecutorService比Timer更安全，功能更强大，后面会有一篇单独进行对比。

**(4)、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：

Crayon Syntax Highlighter v2.7.1

Java

ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();

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

final int index = i;

singleThreadExecutor.execute(new Runnable() {

@Override

public void run() {

try {

System.out.println(index);

Thread.sleep(2000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

}

结果依次输出，相当于顺序执行各个任务。

现行大多数GUI程序都是单线程的。Android中单线程可用于[数据库操作](http://www.trinea.cn/android/database-performance/)，文件操作，应用批量安装，应用批量删除等不适合并发但可能IO阻塞性及影响UI线程响应的操作。