



51CTO 技术论坛

论坛首页 版主墙 最有价值午饭 最新热帖 推荐帖子 资料下载 学习路线图 视频学院 经典版首页

论坛首页 移动平台 Android开发论坛 Android AsyncTask两种线程池分析和总结

我的帖子 个人中心 设置

返回列表

高级回复

新帖

查看:13496 | 回复: 16

Android AsyncTask两种线程池分析和总结 [复制链接]

rongwei84n

发表于 2014-7-2 19:46 | 来自 51CTO网页

[只看他] 楼主



版主

帖子 4997

精华 2

无忧币 10097



个人空间 发短消息
家园好友 他的博客
他的资源
他的课程中心

Android AsyncTask两种线程池分析和总结

(一) 前言

在android AsyncTask里面有两种线程池供我们调用

1. THREAD_POOL_EXECUTOR, 异步线程池
2. SERIAL_EXECUTOR, 同步线程池

正如上面名称描述的那样，一个是异步线程池，多个任务在线程池中并发执行；还有一个是同步执行的。

默认的话，直接调用execute的话，是使用SERIAL_EXECUTOR

下面的话，会用源代码的方式来原因说明这两种线程池的作用和注意事项。

(二) THREAD_POOL_EXECUTOR用法举例

1. 代码

```
01 private static int produceTaskMaxNumber = 500;
02 public void dotask(){
03     for (int i = 1; i <= produceTaskMaxNumber; i++){
04         // 产生一个任务，并将其加入到线程池
05         String task = "task@ " + i;
06         Log.d("Sandy", "put " + task);
07         MyAsyncTask asyncnt = new MyAsyncTask(task);
08         asyncnt.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, 0);
09     }
10 }
11
12 static class MyAsyncTask extends AsyncTask<Integer, Integer, Integer>{
13     private static int consumeTaskSleepTime = 2000;
14     // 保存任务所需要的数据
15     private Object threadPoolTaskData;
16     public MyAsyncTask(String s){
17         threadPoolTaskData = s;
18     }
19     <a href="http://home.51cto.com/index.php?s=/space/5017954" target="_blank">
20     protected Integer doInBackground(Integer... arg0) {
21         Log.d("Sandy", "start .." + threadPoolTaskData
22             + " thread id: " + Thread.currentThread().getId()
23             + " thread name: " + Thread.currentThread().getName());
24         try {
25             // 便于观察，等待一段时间
26             Thread.sleep(consumeTaskSleepTime);
27         }
28         catch (Exception e) {
29             Log.d("Sandy", "", e);
30         }
31         threadPoolTaskData = null;
32         return 0;
33     }
34 }
```

2. 使用方法比较简单，首先创建一个继承自AsyncTask的MyAsyncTask类，然后调用

```
1 MyAsyncTask asyncnt = new MyAsyncTask(task);
2 asyncnt.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, 0);
```

就可以了。

3. 上面代码执行的时候会出错，导致程序异常终止，如下图

很抱歉，“AndroidTest”已停止运行。

确定

确定

原因是：

```
77 AndroidRuntime(125555) FATAL EXCEPTION: main
78 java.lang.RuntimeException: Unable to instantiate class com.example.android.os.AsyncTask: java.lang.OutOfMemoryError: Java heap space
79     at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPoolExecutor.java:303)
80     at java.util.concurrent.ThreadPoolExecutor.execute(ThreadPoolExecutor.java:339)
81     at android.os.AsyncTask.execute(AsyncTask.java:59)
82     at com.example.android.MainActivity$1.run(MainActivity.java:82)
83     at android.os.Handler.dispatchMessage(Handler.java:105)
84     at android.os.Looper.loop(Looper.java:136)
85     at android.app.ActivityThread.main(ActivityThread.java:6595)
86     at java.lang.reflect.Method.invoke(Method.java:130)
87     at com.android.internal.os.ZygoteInit$MethodAndArgsWrapper.run(ZygoteInit.java:812)
88     at android.os.Handler.dispatchMessage(Handler.java:105)
89     at android.os.Looper.loop(Looper.java:136)
90     at android.app.ActivityThread.main(ActivityThread.java:6595)
91     at java.lang.reflect.Method.invoke(Method.java:130)
92     at com.android.internal.os.ZygoteInit$MethodAndArgsWrapper.run(ZygoteInit.java:812)
93     at android.os.Handler.dispatchMessage(Handler.java:105)
94     at android.os.Looper.loop(Looper.java:136)
95     at android.app.ActivityThread.main(ActivityThread.java:6595)
96     at java.lang.reflect.Method.invoke(Method.java:130)
97     at com.android.internal.os.ZygoteInit$MethodAndArgsWrapper.run(ZygoteInit.java:812)
98     at android.os.Handler.dispatchMessage(Handler.java:105)
99     at android.os.Looper.loop(Looper.java:136)
100    at android.app.ActivityThread.main(ActivityThread.java:6595)
101    at java.lang.reflect.Method.invoke(Method.java:130)
102    at com.android.internal.os.ZygoteInit$MethodAndArgsWrapper.run(ZygoteInit.java:812)
```

就是因为我们尝试添加500个task到AsyncTask.THREAD_POOL_EXECUTOR线程池中，但是它的核心线程是5，队列容量是128，最大线程数是9。

所以，抛出了这个异常。

那么，接下来的话，我们会去分析这个异常怎么出来的。

（三）THREAD_POOL_EXECUTOR代码分析

从AsyncTask.THREAD_POOL_EXECUTOR的定义开始分析

1. 代码路径

frameworks\base\core\java\android\os\AsyncTask.java

代码：

```
01 private static final int CPU_COUNT = Runtime.getRuntime().availableProcessors();
02 private static final int CORE_POOL_SIZE = CPU_COUNT + 1;
03 private static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
04 private static final int KEEP_ALIVE = 1;
05
06 ...
07 /**
08  * An {@link Executor} that can be used to execute tasks in parallel.
09  */
10 public static final Executor THREAD_POOL_EXECUTOR
11     = new ThreadPoolExecutor(CORE_POOL_SIZE, MAXIMUM_POOL_SIZE, KEEP_ALIVE,
12         TimeUnit.SECONDS, sPoolWorkQueue, sThreadFactory);
```

它的几个参数CORE_POOL_SIZE, MAXIMUM_POOL_SIZE, 都是根据当前手机的处理器的数量进行动态定义的。

那么，继续往下面看，看这几个参数传进去后是什么意思。

2. 代码路径

\libcore\luni\src\main\java\java\util\concurrent\ThreadPoolExecutor.java

代码：

```
01 public ThreadPoolExecutor(int corePoolSize,
02                             int maximumPoolSize,
03                             long keepAliveTime,
04                             TimeUnit unit,
05                             BlockingQueue<Runnable> workQueue,
06                             ThreadFactory threadFactory,
07                             RejectedExecutionHandler handler) {
08     if (corePoolSize < 0 ||
09         maximumPoolSize <= 0 ||
10         maximumPoolSize < corePoolSize ||
11         keepAliveTime < 0)
12         throw new IllegalArgumentException();
13     if (workQueue == null || threadFactory == null || handler == null)
14         throw new NullPointerException();
15     this.corePoolSize = corePoolSize;
16     this.maximumPoolSize = maximumPoolSize;
17     this.workQueue = workQueue;
18     this.keepAliveTime = unit.toNanos(keepAliveTime);
19     this.threadFactory = threadFactory;
20     this.handler = handler;
21 }
22
23 /**
24  * The default rejected execution handler
25  */
26 private static final RejectedExecutionHandler defaultHandler =
27     new AbortPolicy();
```

这是ThreadPoolExecutor的构造函数，首先需要明白的是这几个参数的含义

- A. corePoolSize: 线程池维护线程的最少数量
- B. maximumPoolSize: 线程池维护线程的最大数量
- C. keepAliveTime: 线程池维护线程所允许的空闲时间
- D. unit: 线程池维护线程所允许的空闲时间的单位
- E. workQueue: 线程池所使用的缓冲队列

F. handler: 线程池对拒绝任务的处理策略

当一个任务通过`asynct.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, 0)`方法欲添加到线程池时:
如果此时线程池中的数量小于`corePoolSize`, 即使线程池中的线程都处于空闲状态, 也要创建新的线程来处理被添加的任务。
如果此时线程池中的数量等于 `corePoolSize`, 但是缓冲队列 `workQueue`未满, 那么任务被放入缓冲队列。
如果此时线程池中的数量大于`corePoolSize`, 缓冲队列`workQueue`满, 并且线程池中的数量小于`maximumPoolSize`, 建新的线程来处理被添加的任务。
如果此时线程池中的数量大于`corePoolSize`, 缓冲队列`workQueue`满, 并且线程池中的数量等于`maximumPoolSize`, 那么通过 `handler`所指定的策略来处理此任务。

也就是: 处理任务的优先级为:

核心线程`corePoolSize`、任务队列`workQueue`、最大线程`maximumPoolSize`, 如果三者都满了, 使用`handler`处理被拒绝的任务。

当线程池中的线程数量大于 `corePoolSize`时, 如果某线程空闲时间超过`keepAliveTime`, 线程将被终止。这样, 线程池可以动态的调整池中的线程数。

`unit`可选的参数为`java.util.concurrent.TimeUnit`中的几个静态属性:
`NANOSECONDS`、`MICROSECONDS`、`MILLISECONDS`、`SECONDS`。

`workQueue`是`BlockQueue`的子类, `ArrayBlockingQueue`, `DelayQueue`

`handler`有四个选择(这不是`android.Handler`):

`ThreadPoolExecutor.AbortPolicy()` - 这个也是`AsyncTask.THREAD_POOL_EXECUTOR`使用的

抛出`java.util.concurrent.RejectedExecutionException`异常

`ThreadPoolExecutor.CallerRunsPolicy()`

重试添加当前的任务, 他会自动重复调用`execute()`方法

`ThreadPoolExecutor.DiscardOldestPolicy()`

抛弃旧的任务

`ThreadPoolExecutor.DiscardPolicy()`

抛弃当前的任务

所以, 正是我们的`AsyncTask.THREAD_POOL_EXECUTOR`使用了`AbortPolicy()`类型的`handler`, 所以才会抛出异常..

那么, 在把任务添加到`AsyncTask.THREAD_POOL_EXECUTOR`之后, 下面的工作就是由这个线程池来调度线程执行任务了。

(四) `AsyncTask.SERIAL_EXECUTOR`

1. 使用方法

`AsyncTask.SERIAL_EXECUTOR`的使用方法和`Async.THREAD_POOL_EXECUTOR`差不多。不过正如前面所说, 它是默认的`Executor`, 所以可以直接调用, 所以可以有两种调用方法。

```
1 | a.    asynct.executeOnExecutor(AsyncTask.SERIAL_EXECUTOR, 0);  
2 | b.    asynct.execute(0);
```

效果是一样的

2. 执行流程

代码路径:

`frameworks\base\core\java\android\os\AsyncTask.java`

代码:

```
01 | public final AsyncTask<Params, Progress, Result> executeOnExecutor(Execut  
02 |     Params... params) {  
03 |     ...  
04 |     exec.execute(mFuture);  
05 |     ....  
06 | }  
07 |  
08 | private static class SerialExecutor implements Executor {  
09 |     final ArrayDeque<Runnable> mTasks = new ArrayDeque<Runnable>();  
10 |     Runnable mActive;  
11 |     public synchronized void execute(final Runnable r) {  
12 |         mTasks.offer(new Runnable() {  
13 |             public void run() {  
14 |                 try {  
15 |                     r.run();  
16 |                 } finally {  
17 |                     scheduleNext();  
18 |                 }  
19 |             }  
20 |         });  
21 |         if (mActive == null) {  
22 |             mActive = r;  
23 |         }  
24 |     }  
25 | }
```

```
18         }
19     }
20     });
21     if (mActive == null) {
22         scheduleNext();
23     }
24 }
25
26 protected synchronized void scheduleNext() {
27     if ((mActive = mTasks.poll()) != null) {
28         THREAD_POOL_EXECUTOR.execute(mActive);
29     }
30 }
31 }
```

嗯，它会调用到SerialExecutor.execute(Runnable r)方法
在这个方法里面，它首先把任务放到mTasks这个集合里面；然后判断mActive是否为空,再调用scheduleNext ()方法。
mActive为null的意思是当前没有任务在执行，如果mActive!=null，那么说明当前有任务正在执行，那么只要把任务添加到mTasks里面即可。
因为任务执行完毕后，会再次调用scheduleNext ()方法的，就是
finally {
 scheduleNext();
}
这样就形成了一种链状调用结构，只要mTasks里面还有任务，就会不断逐一调用，如果后面有任务进来，就只要添加到mTasks里面即可。
同时，不知道大家注意到没有，这两个方法都是synchronized的，这样，就保证了多线程之间调度问题。
否则肯定会出现问题的，至于什么问题，大家想想就能明白。

4. 继续分析scheduleNext () 方法

这个方法首先把mTasks里面的数据取一个出来，然后调用
THREAD_POOL_EXECUTOR.execute(mActive);
我晕，这不就是上面一直在分析的AsyncTask.THREAD_POOL_EXECUTOR么？
好吧，原来AsyncTask.THREAD_POOL_EXECUTOR和AsyncTask.SERIAL_EXECUTOR的区别就是SERIAL_EXECUTOR在THREAD_POOL_EXECUTOR的基础上添加了一个mTasks的集合来保证任务顺序执行而已...

(五) 总结

说了这么多，总结下

1. AsyncTask里面有THREAD_POOL_EXECUTOR和SERIAL_EXECUTOR两种方式异步执行任务；THREAD_POOL_EXECUTOR是异步的，而SERIAL_EXECUTOR任务是顺序执行的。
2. THREAD_POOL_EXECUTOR如果添加的任务过多，没有及时处理的话，会导致程序崩溃，它的队列size是128；它的调度规则是核心池大小，队列大小，以及最大线程数和异常处理Handler来决定的。
3. SERIAL_EXECUTOR本质是在THREAD_POOL_EXECUTOR的基础上添加一个mTasks的集合来保证任务的顺序执行。

参考网址

http://blog.sina.com.cn/s/blog_8417aea80100t483.html

本帖最后由 rongwei84n 于 2014-7-2 19:47 编辑

分享至:

2

收藏 +

楼主关注

Android程序启动速度优化小结
三星s4上4.2.2。 ，调用系统自带相机拍照后，
android:multiprocess="true"作用
又遇到瓶颈了，怎么从非activity把handler的消息传递
遍历sd卡文件想查询一个指定文件时遇到的问题
Android-小小设置永久解决程序因为未捕获异常而异常终

版主推荐

Android ListView实现不同item的方法和原理分析
Android LocalBroadcastManager使用方法和代码流程分
Android第二期 - 动画数字三元归一
安卓智能聊天机器人的实现及源码分享
imageview如何显示多张图片
安卓与苹果系统的区别？

51CTO社区：活动汇总【9个活动正在进行中】 | #论坛扒点档#，随你造！