

郭霖的专栏

每当你在感叹，如果有这样一个东西就好了的时候，请注意，其实这是你的机会

☰ 目录视图

☰ 摘要视图

RSS 订阅

个人资料



guolin

关注 发私信

异步赠书：10月Python畅销书升级 【线路图】人工智能到底学什么?！ 程序员9月书讯 节后荐书：Python、PyQt5、Kotlin（评论送书）

Android AsyncTask完全解析，带你从源码的角度彻底理解

标签： Android AsyncTask 异步 源码 线程池 2013-10-11 08:35 91889人阅读 评论(90) 收藏 举报

☰ 分类：
Android疑难解析（41） ▼

版权声明：本文出自郭霖的博客，转载必须注明出处。

目录(?) [+]



访问： 7302941次
积分： 40883
等级： BLOG > 8
排名： 第96名
原创： 98篇 转载： 0篇
译文： 6篇 评论： 13695条

我的新书

《第二行代码》已出版
新书基于Android 7.0系统全面升级
更加入了许多振奋人心的新技术



查看详情

纸质书购买：
京东 天猫
当当 亚马逊

电子书购买：
PDF Kindle
豆瓣版 多看版

转载请注明出处：http://blog.csdn.net/guolin_blog/article/details/11711405

我们都知道，Android UI是线程不安全的，如果想要在子线程里进行UI操作，就需要一个异步消息处理机制。之前我也写过了一篇文章从源码层面分析了Android的异步消息处理机制，感兴趣的朋友可以参考 [Android Handler、Message完全解析，带你从源码的角度彻底理解](#)。

不过为了更加方便我们在子线程中更新UI元素，Android从1.5版本就引入了一个AsyncTask类，使用它就可以非常灵活方便地从子线程切换到UI线程，我们本篇文章的主角也就正是它了。

AsyncTask很早就出现在Android的API里了，所以我相信大多数朋友对它的用法都已经非常熟悉。不过今天我还是准备从AsyncTask的基本用法开始讲起，然后我们再来一起分析下AsyncTask源码，看看它是如何实现的，最后我会介绍一些关于AsyncTask你所不知道的秘密。

AsyncTask的基本用法

首先来看一下AsyncTask的基本用法，由于AsyncTask是一个抽象类，所以如果我们想使用它，就必须创建一个子类去继承它。在继承时我们可以为AsyncTask类指定三个泛型参数，这三个参数的用途如下：

1. Params

在执行AsyncTask时需要传入的参数，可用于在后台任务中使用。

2. Progress

微信公众号推荐

关注我的技术公众号，每天都有优质技术文章推送。



关注我的娱乐公众号，工作、学习累了的时候放松一下自己。



微信扫一扫上方二维码即可关注

共同进步

感兴趣的朋友可以加入我的QQ群，一起讨论学习，共同进步。

群号：256344794

博客专栏



Glide最全解析

文章：6篇
阅读：141522



Android数据库高手秘籍

文章：9篇
阅读：518306

友情链接

鸿洋的博客

夏安明的博客

stormzhang的博客

极客导航

金山云-以渔论坛

文章分类

后台任务执行时，如果需要在界面上显示当前的进度，则使用这里指定的泛型作为进度单位。

3. Result

当任务执行完毕后，如果需要对结果进行返回，则使用这里指定的泛型作为返回值类型。

因此，一个最简单的自定义AsyncTask就可以写成如下方式：

```
[java]
01. class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
02.     .....
03. }
```

这里我们把AsyncTask的第一个泛型参数指定为Void，表示在执行AsyncTask的时候不需要传入参数给后台任务。第二个泛型参数指定为Integer，表示使用整型数据来作为进度显示单位。第三个泛型参数指定为Boolean，则表示使用布尔型数据来反馈执行结果。

当然，目前我们自定义的DownloadTask还是一个空任务，并不能进行任何实际的操作去重写AsyncTask中的几个方法才能完成对任务的定制。经常需要去重写的方法有以

1. onPreExecute()

这个方法会在后台任务开始执行之间调用，用于进行一些界面上的初始化操作，比如进度条对话框等。

2. doInBackground(Params...)

这个方法中的所有代码都会在线程中运行，我们应该在这里去处理所有的耗时任务。任务一旦完成就可以通过return语句来将任务的执行结果进行返回，如果AsyncTask的第三个泛型参数指定的是Void，就可以不返回任务执行结果。注意，在这个方法中是不可以进行UI操作的，如果需要更新UI元素，比如说反馈当前任务的执行进度，可以调用publishProgress(Progress...)方法来完成。

3. onProgressUpdate(Progress...)

当在后台任务中调用了publishProgress(Progress...)方法后，这个方法就很快会被调用，方法中携带的参数就是在后台任务中传递过来的。在这个方法中可以对UI进行操作，利用参数中的数值就可以对界面元素进行相应的更新。

4. onPostExecute(Result)

当后台任务执行完毕并通过return语句进行返回时，这个方法就很快会被调用。返回的数据会作为参数传递到此方法中，可以利用返回的数据来进行一些UI操作，比如说提醒任务执行的结果，以及关闭进度条对话框等。

因此，一个比较完整的自定义AsyncTask就可以写成如下方式：

```
[java]
01. class DownloadTask extends AsyncTask<Void, Integer, Boolean> {
02.
03.     @Override
```

- Android精华教程 (21)
- Android疑难解析 (42)
- Android第一行代码 (4)
- Android数据库高手秘籍 (8)
- Java设计模式透析 (5)
- Ruby设计模式透析 (5)
- 随笔 (7)

文章存档

- 2017年10月 (1)
- 2017年08月 (1)
- 2017年06月 (1)
- 2017年05月 (1)
- 2017年04月 (1)

展开

阅读排行

- Android事件分发机制完... (302593)
- Android Vollev完全解析(... (280247)
- Android Fragment完全解... (257021)
- Android LavoutInflater原... (234769)
- Android Service完全解析... (209673)
- Android ActionBar完全解... (177632)
- Android高效加载大图、... (173151)
- Android视图绘制流程完... (172274)
- Android Vollev完全解析(... (140656)
- Android自定义View的实... (130726)

评论排行

- 历时一年、我的著作《第... (723)
- 历久而新、我的新书《第... (632)
- Android LavoutInflater原... (357)
- Android事件分发机制完... (322)
- Android照片墙完整版、... (318)
- Android数据库高手秘籍(... (273)
- Android下拉刷新完全解... (271)
- Android双向滑动菜单完... (267)
- 2015年终总结、忙碌和无... (253)
- Android Fragment应用实... (252)

最新评论

- Android Vollev完全解析(一)、初识V... qq_34070678 : @qq_37596793:呵呵，你真是个好葩呀
- Android新特性介绍、ConstraintLav...

```
04.     protected void onPreExecute() {
05.         progressDialog.show();
06.     }
07.
08.     @Override
09.     protected Boolean doInBackground(Void... params) {
10.         try {
11.             while (true) {
12.                 int downloadPercent = doDownload();
13.                 publishProgress(downloadPercent);
14.                 if (downloadPercent >= 100) {
15.                     break;
16.                 }
17.             }
18.         } catch (Exception e) {
19.             return false;
20.         }
21.         return true;
22.     }
23.
24.     @Override
25.     protected void onProgressUpdate(Integer... values) {
26.         progressDialog.setMessage("当前下载进度: " + values[0] +
27.
28.
29.     @Override
30.     protected void onPostExecute(Boolean result) {
31.         progressDialog.dismiss();
32.         if (result) {
33.             Toast.makeText(context, "下载成功", Toast.LENGTH_SHORT).show();
34.         } else {
35.             Toast.makeText(context, "下载失败", Toast.LENGTH_SHORT).show();
36.         }
37.     }
38. }
```

这里我们模拟了一个下载任务，在doInBackground()方法中去执行具体的下载逻辑，在onProgressUpdate()方法中显示当前的下载进度，在onPostExecute()方法中来提示任务的执行结果。如果想要启动这个任务，只需要简单地调用以下代码即可：

```
[java]
01.     new DownloadTask().execute();
```

以上就是AsyncTask的基本用法，怎么样，是不是感觉在子线程和UI线程之间进行切换变得灵活了很多？我们并不需求去考虑什么异步消息处理机制，也不需要专门使用一个Handler来发送和接收消息，只需要调用一下publishProgress()方法就可以轻松地子线程切换到UI线程了。

分析AsyncTask的源码

虽然AsyncTask这么简单好用，但你知道它是怎样实现的吗？那么接下来，我们就来分析一下AsyncTask的源码，对它的实现原理一探究竟。注意这里我选用的是Android 4.0的源码，如果你查看的是其它版本的源码，可能会有一些出入。

hello18767163272 : ConstraintLayout和CoordinatorLayout同时使用,能做到吗,郭大神??

Android照片墙应用实现、再多的图... DawnT_Young : @MrSun110:这个图片都是谷歌提供的, 应该需要翻墙才能访问

Android图片加载框架最全解析 (六... 天一方蓝 : 赞

Android数据库高手秘籍(四)——伸... T_AlانDream : 关系建立好了, 但是怎么样保存他们表1的id怎么传给表2呢

Android系统联系人全特效实现(下)... wei_ting1 : 膜拜楼主

Android图片加载框架最全解析 (四... csdn小瓯 : 篇幅过长了, 学习效果很差

Android状态栏微技巧, 带你真正理... beita08 : 郭神, 你好! 我的页面使用的是博客中的“透明状态栏效果”, 在输入框弹出软键盘时adjustResize...

Android图片加载框架最全解析 (六... Hitomis丶 : 哎。。。还是没讲如何查询某张图片是否已经缓存过

Android高级图片滚动控件、编写3... weixin_40377464 : final Handler handle r=new Handler(); Runnab...

文章搜索

统计

站长统计

从之前DownloadTask的代码就可以看出, 在启动某一个任务之前, 要先new出它的实例, 因此, 我们就先来看一下AsyncTask构造函数中的源码, 如下所示:

```
[java]
01. public AsyncTask() {
02.     mWorker = new WorkerRunnable<Params, Result>() {
03.         public Result call() throws Exception {
04.             mTaskInvoked.set(true);
05.             Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
06.             return postResult(doInBackground(mParams));
07.         }
08.     };
09.     mFuture = new FutureTask<Result>(mWorker) {
10.         @Override
11.         protected void done() {
12.             try {
13.                 final Result result = get();
14.                 postResultIfNotInvoked(result);
15.             } catch (InterruptedException e) {
16.                 android.util.Log.w(LOG_TAG, e);
17.             } catch (ExecutionException e) {
18.                 throw new RuntimeException("An error occurred wh
19.                     e.getCause());
20.             } catch (CancellationException e) {
21.                 postResultIfNotInvoked(null);
22.             } catch (Throwable t) {
23.                 throw new RuntimeException("An error occurred while executing "
24.                     + "doInBackground()", t);
25.             }
26.         }
27.     };
28. }
```

这段代码虽然看起来有点长, 但实际上并没有任何具体的逻辑会得到执行, 只是初始化了两个变量, mWorker和mFuture, 并在初始化mFuture的时候将mWorker作为参数传入。mWorker是一个Callable对象, mFuture是一个FutureTask对象, 这两个变量会暂时保存在内存中, 稍后才会用到它们。

接着如果想要启动某一个任务, 就需要调用该任务的execute()方法, 因此现在我们来查看execute()方法的源码, 如下所示:

```
[java]
01. public final AsyncTask<Params, Progress, Result> execute(Params... params) {
02.     return executeOnExecutor(sDefaultExecutor, params);
03. }
```

简单的有点过分了, 只有一行代码, 仅是调用了executeOnExecutor()方法, 那么具体的逻辑就应该写在这个方法里了, 快跟进去瞧一瞧:

```
[java]
01. public final AsyncTask<Params, Progress, Result> executeOnExecutor(Executor e
02.     Params... params) {
03.     if (mStatus != Status.PENDING) {
```

```

04.         switch (mStatus) {
05.             case RUNNING:
06.                 throw new IllegalStateException("Cannot execute task:"
07.                     + " the task is already running.");
08.             case FINISHED:
09.                 throw new IllegalStateException("Cannot execute task:"
10.                     + " the task has already been executed "
11.                     + "(a task can be executed only once)");
12.         }
13.     }
14.     mStatus = Status.RUNNING;
15.     onPreExecute();
16.     mWorker.mParams = params;
17.     exec.execute(mFuture);
18.     return this;
19. }

```

果然，这里的代码看上去才正常点。可以看到，在第15行调用了onPreExecute()方法，因此证明了onPreExecute()方法会第一个得到执行。可是接下来的代码就看不明白了，怎么没见到哪里有调用doInBackground()方法呢？别着急，慢慢找总会找到的，我们看到，在第17行调用了execute()方法，并将前面初始化的mFuture对象传了进去，那么这个Executor对象又是哪里来的呢？我们继续往下看，在第16行调用了mWorker.mParams = params;，这里的params是在哪里定义的，源码如下所示：

[java]

```

01. public static final Executor SERIAL_EXECUTOR = new SerialExecutor();
02. ....
03. private static volatile Executor sDefaultExecutor = SERIAL_EXECUTOR;

```

可以看到，这里先new出了一个SERIAL_EXECUTOR常量，然后将sDefaultExecutor的值赋值为这个常量，也就是说，刚才在executeOnExecutor()方法中调用的execute()方法，其实也就是调用的SerialExecutor类中的execute()方法。那么我们自然要去看看SerialExecutor的源码了，如下所示：

[java]

```

01. private static class SerialExecutor implements Executor {
02.     final ArrayDeque<Runnable> mTasks = new ArrayDeque<Runnable>();
03.     Runnable mActive;
04.
05.     public synchronized void execute(final Runnable r) {
06.         mTasks.offer(new Runnable() {
07.             public void run() {
08.                 try {
09.                     r.run();
10.                 } finally {
11.                     scheduleNext();
12.                 }
13.             }
14.         });
15.         if (mActive == null) {
16.             scheduleNext();
17.         }
18.     }
19.
20.     protected synchronized void scheduleNext() {
21.         if ((mActive = mTasks.poll()) != null) {

```

```

22.         THREAD_POOL_EXECUTOR.execute(mActive);
23.     }
24. }
25. }

```

SerialExecutor类中也有一个execute()方法，这个方法里的所有逻辑就是在子线程中执行的了，注意这个方法有一个Runnable参数，那么目前这个参数的值是什么呢？当然就是mFuture对象了，也就是说在第9行我们要调用的是FutureTask类的run()方法，而在这个方法里又会去调用Sync内部类的innerRun()方法，因此我们直接来看innerRun()方法的源码：

```

[java]
01. void innerRun() {
02.     if (!compareAndSetState(READY, RUNNING))
03.         return;
04.     runner = Thread.currentThread();
05.     if (getState() == RUNNING) { // recheck after setting thread
06.         V result;
07.         try {
08.             result = callable.call();
09.         } catch (Throwable ex) {
10.             setException(ex);
11.             return;
12.         }
13.         set(result);
14.     } else {
15.         releaseShared(0); // cancel
16.     }
17. }

```

可以看到，在第8行调用了callable的call()方法，那么这个callable对象是什么呢？其实就是在初始化mFuture对象时传入的mWorker对象了，此时调用的call()方法，也就是一开始在AsyncTask的构造函数中指定的，我们把它单独拿出来看一下，代码如下所示：

```

[java]
01. public Result call() throws Exception {
02.     mTaskInvoked.set(true);
03.     Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
04.     return postResult(doInBackground(mParams));
05. }

```

在postResult()方法的参数里面，我们终于找到了doInBackground()方法的调用处，虽然经过了很多周转，但目前的代码仍然是运行在子线程当中的，所以这也就是为什么我们可以在doInBackground()方法中去处理耗时的逻辑。接着将doInBackground()方法返回的结果传递给了postResult()方法，这个方法的源码如下所示：

```

[java]
01. private Result postResult(Result result) {
02.     Message message = sHandler.obtainMessage(MESSAGE_POST_RESULT,
03.         new AsyncTaskResult<Result>(this, result));
04.     message.sendToTarget();
05.     return result;
06. }

```


如果你已经熟悉了异步消息处理机制，这段代码对你来说一定非常简单吧。这里使用sHandler对象发出了一条消息，消息中携带了MESSAGE_POST_RESULT常量和一个表示任务执行结果的AsyncTaskResult对象。这个sHandler对象是InternalHandler类的一个实例，那么稍后这条消息肯定会在InternalHandler的handleMessage()方法中被处理。InternalHandler的源码如下所示：

```
[java]
01. private static class InternalHandler extends Handler {
02.     @SuppressWarnings({"unchecked", "RawUseOfParameterizedType"})
03.     @Override
04.     public void handleMessage(Message msg) {
05.         AsyncTaskResult result = (AsyncTaskResult) msg.obj;
06.         switch (msg.what) {
07.             case MESSAGE_POST_RESULT:
08.                 // There is only one result
09.                 result.mTask.finish(result.mData[0]);
10.                 break;
11.             case MESSAGE_POST_PROGRESS:
12.                 result.mTask.onProgressUpdate(result.mData);
13.                 break;
14.         }
15.     }
16. }
```

这里对消息的类型进行了判断，如果这是一条MESSAGE_POST_RESULT消息，就会调用finish()方法，如果这是一条MESSAGE_POST_PROGRESS消息，就会去执行onProgressUpdate()方法。InternalHandler的finish()方法的源码如下所示：

```
[java]
01. private void finish(Result result) {
02.     if (isCancelled()) {
03.         onCancelled(result);
04.     } else {
05.         onPostExecute(result);
06.     }
07.     mStatus = Status.FINISHED;
08. }
```

可以看到，如果当前任务被取消掉了，就会调用onCancelled()方法，如果没有被取消，则调用onPostExecute()方法，这样当前任务的执行就全部结束了。

我们注意到，在刚才InternalHandler的handleMessage()方法里，还有一种MESSAGE_POST_PROGRESS的消息类型，这种消息是用于当前进度的，调用的正是onProgressUpdate()方法，那么什么时候才会发出这样一条消息呢？相信你已经猜到了，查看publishProgress()方法的源码，如下所示：

```
[java]
01. protected final void publishProgress(Progress... values) {
02.     if (!isCancelled()) {
03.         sHandler.obtainMessage(MESSAGE_POST_PROGRESS,
04.             new AsyncTaskResult<Progress>
05.                 (this, values)).sendToTarget();
06.     }
```

```
05.     }  
06. }
```

非常清晰了吧！正因如此，在doInBackground()方法中调用publishProgress()方法才可以从子线程切换到UI线程，从而完成对UI元素的更新操作。其实也没有什么神秘的，因为说到底，AsyncTask也是使用的异步消息处理机制，只是做了非常好的封装而已。

读到这里，相信你对AsyncTask中的每个回调方法的作用、原理、以及何时会被调用都已经搞明白了吧。

关于AsyncTask你所不知道的秘密

不得不说，刚才我们在分析SerialExecutor的时候，其实并没有分析的很仔细，仅仅只是关注了它会调用mFuture中的run()方法，但是至于什么时候会调用我们并没有进一步地研究。其实是AsyncTask在3.0版本以后做了最主要的修改的地方，它在AsyncTask中是以常量的，因此在整个应用程序中的所有AsyncTask实例都会共用同一个SerialExecutor。下面我进行更加详细的分析，为了方便阅读，我把它的代码再贴出来一遍：

```
[java]  
01. private static class SerialExecutor implements Executor {  
02.     final ArrayDeque<Runnable> mTasks = new ArrayDeque<Runnable>();  
03.     Runnable mActive;  
04.  
05.     public synchronized void execute(final Runnable r) {  
06.         mTasks.offer(new Runnable() {  
07.             public void run() {  
08.                 try {  
09.                     r.run();  
10.                 } finally {  
11.                     scheduleNext();  
12.                 }  
13.             }  
14.         });  
15.         if (mActive == null) {  
16.             scheduleNext();  
17.         }  
18.     }  
19.  
20.     protected synchronized void scheduleNext() {  
21.         if ((mActive = mTasks.poll()) != null) {  
22.             THREAD_POOL_EXECUTOR.execute(mActive);  
23.         }  
24.     }  
25. }
```

可以看到，SerialExecutor是使用ArrayDeque这个队列来管理Runnable对象的，如果我们一次性启动了很多个任务，首先在第一次运行execute()方法的时候，会调用ArrayDeque的offer()方法将传入的Runnable对象添加到队列的尾部，然后判断mActive对象是不是等于null，第一次运行当然是等于null了，于是会调用scheduleNext()方法。在这个方法中会从队列的头部取值，并赋值给mActive对象，然

后调用THREAD_POOL_EXECUTOR去执行取出的Runnable对象。之后如何又有新的任务被执行，同样还会调用offer()方法将传入的Runnable添加到队列的尾部，但是再去给mActive对象做非空检查的时候就会发现mActive对象已经不再是null了，于是就不会再调用scheduleNext()方法。

那么后面添加的任务岂不是永远得不到处理了？当然不是，看一看offer()方法里传入的Runnable匿名类，这里使用了一个try finally代码块，并在finally中调用了scheduleNext()方法，保证无论发生什么情况，这个方法都会被调用。也就是说，每次当一个任务执行完毕后，下一个任务才会得到执行，SerialExecutor模仿的是单一线程池的效果，如果我们快速地启动了很多任务，同一时刻只会有一个线程正在执行，其余的均处于等待状态。[Android照片墙应用实现，再多的图片也不怕崩溃](#) 这篇文章中例子的运行结果也证实了这个结论。

不过你可能还不知道，在Android 3.0之前是并没有SerialExecutor这个类的，那个时候是直接AsyncTask中构建了一个sExecutor常量，并对线程池总大小，同一时刻能够运行的线程数代码如下所示：

```
[java]
01. private static final int CORE_POOL_SIZE = 5;
02. private static final int MAXIMUM_POOL_SIZE = 128;
03. private static final int KEEP_ALIVE = 10;
04. ....
05. private static final ThreadPoolExecutor sExecutor = new ThreadPoolExecutor(CORE_POOL_SIZE,
06.                                     MAXIMUM_POOL_SIZE, KEEP_ALIVE, TimeUnit.SECONDS, sWorkQueue, null);
```

可以看到，这里规定同一时刻能够运行的线程数为5个，线程池总大小为128。也就是说当我们启动了10个任务时，只有5个任务能够立刻执行，另外的5个任务则需要等待，当有一个任务执行完毕后，第6个任务才会启动，以此类推。而线程池中最大能存放的线程数是128个，当我们尝试去添加第129个任务时，程序就会崩溃。

因此在3.0版本中AsyncTask的改动还是挺大的，在3.0之前的AsyncTask可以同时有5个任务在执行，而3.0之后的AsyncTask同时只能有1个任务在执行。为什么升级之后可以同时执行的任务数反而变少了呢？这是因为更新后的AsyncTask已变得更加灵活，如果不想使用默认的线程池，还可以自由地进行配置。比如使用如下的代码来启动任务：

```
[java]
01. Executor exec = new ThreadPoolExecutor(15, 200, 10,
02.                                     TimeUnit.SECONDS, new LinkedBlockingQueue<Runnable>());
03. new DownloadTask().executeOnExecutor(exec);
```

这样就可以使用我们自定义的一个Executor来执行任务，而不是使用SerialExecutor。上述代码的效果允许在同一时刻有15个任务正在执行，并且最多能够存储200个任务。

好了，到这里我们就已经把关于AsyncTask的所有重要内容深入浅出地理解了一遍，相信在将来使用它的时候能够更加得心应手。

关注我的技术公众号，每天都有优质技术文章推送。关注我的娱乐公众号，工作、学习累了的时候放松一下自己。

微信扫一扫下方二维码即可关注：



顶

134

踩

12

- 上一篇 Android 3D滑动菜单完全解析，实现推拉门式的立体特效
- 下一篇 Android数据库安全解决方案，使用SQLCipher进行加解密

相关文章推荐

- Android照片墙应用实现，再多的图片也不...
 - Python全栈工程师入门指南
 - Android AsyncTask 源码解析（张鸿洋版）
 - 自然语言处理在“天猫精灵”的实践应用--姜...
 - Android AsyncTask完全解析，带你从源码...
 - Vue2.x基本特性解析
 - 历久而新，我的新书《第二行代码》已出...
 - 程序员都应该掌握的Git和Github实用教程
- Android AsyncTask 源码解析
 - 深度学习项目实战-人脸检测
 - Android异步消息处理机制完全解析，带你...
 - Shell脚本编程
 - AsyncTask源码解析
 - Android 异步消息处理机制 让你深入理解 L...
 - Android AsyncTask完全解析，带你从源码...
 - Android AsyncTask完全解析，带你从源码...

查看评论



Amandu1995

58楼 2017-07-05 15:36发表

看懂了前面基本用法，后面源码有些不太了解。感谢分享！



code_wl

57楼 2017-01-07 19:19发表

郭神，一直看你的博客，我改造了一下asyncTask，使用方式类似于Okhttp了，并行的方式只需要修改线程池就可以了，忘评价下代码，本人目前还是大四学生 <https://github.com/wangli0/SuperTask.git>



易水南风

56楼 2016-12-07 12:15发表

感谢楼主分享。想问下，“3.0之后的AsyncTask同时只能有1个任务在执行”这句话是指同一个AsyncTask子类来说的还是一个应用中所有AsyncTask子类呢？



JeromeLiee

Re: 2017-09-22 10:48发表

回复sinat_23092639：是不同的Task，因为同一个Task如果正在执行，你再调用一次它的execute方法，会抛异常IllegalStateException("Can not execute task: the task is already running.<div data-bbox="330 382 370 413" data-label="Image">

刘韦声

55楼 2016-09-15

+1



Kerwin555

54楼 2016-09-04 16:07发表

不得不说，郭神是最会讲述的人，对比了几篇AsyncTask博客，按神的顺序来说，理解起来更容易



2013_android_study

53楼 2016-05-18 15:35发表

我也刚看这个源码，SerialExecutor是用来让所有任务串行取出。所有任务都会添加到ArrayDeque这个队列中来等待排队执行，真正执行是THREAD_POOL_EXECUTOR来执行任务。一个是任务排队用的，另一个是来执行任务的。并不是说THREAD_POOL_EXECUTOR 不能并发，只是任务没有并发提交到它。



2013_android_study

52楼 2016-05-18 15:34发表

我也刚看这个源码，SerialExecutor是用来让所有任务串行取出。所有任务都会添加到ArrayDeque这个队列中来等待排队执行，真正执行是THREAD_POOL_EXECUTOR来执行任务。一个是任务排队用的，另一个是来执行任务的。并不是说THREAD_POOL_EXECUTOR 不能并发，只是任务没有并发提交到它。



viki34

Re: 2017-03-16 20:26发表

回复u010802293：+1
郭哥 这个说错了

chengkun_123

Re: 2016-11-26 11:32发表



回复u010802293: 对的



Kerwin555

Re: 2016-09-04 10:45发表

回复u010802293: +1



JunDooong

51楼 2016-04-19 10:14发表

郭大神媳妇不错啊



Sausure

50楼 2016-03-02 11:27发表

博主，我想指出你的一个疏忽，在SerialExecutor类中，如果此时mActive非空，同时mTasks只剩下一个Runnable，当该Runnable对象执行时调用finally的scheduleNext()，就会调用到mActive = mTasks.poll()，由于此时mTask已经没有Runnable了所以返回null即mActive会重置为null，那么if (mActive == null) {scheduleNext();}还是有机会再次被调用的，不仅仅是第一次



罗小辉

49楼 2016-02-21 2'

第二次来看这个帖子，理解深了好多！



SnowDragon2015

48楼 2016-02-17 15:13发表

不错支持一个



Richard_tan0113

47楼 2016-01-14 11:06发表

每个大神都会分析AsyncTask，每个大神的思路略有不同。都看看，非常好。



杭州山不高

46楼 2015-11-11 17:25发表

写得好！为你点赞！



你相信命运吗

45楼 2015-09-27 21:03发表

用线程池和SerialExecutor有什么用啊，我们一般不是一个AsyncTask值execute一次吗，不会有多个任务给它啊



你相信命运吗

Re: 2015-09-27 21:05发表

回复yuanzhongcheng: 哦，我知道了，SerialExecutor是static的

qinpengtaiyuan

44楼 2015-09-12 20:22发表

秦

最后一部分的内容在使用的时候有所感觉，但是没有去深究，这下明白了。谢谢作者，作者辛苦了！



squery

43楼 2015-08-22 15:48发表

异步任务是基础啊，郭神能不能讲一讲里面的WorkRunnable和FutureTask，以前根本没接触过Java中也没提到过，看起来比较吃力，郭神你都是一带而过的，里面具体细节木有讲啊。



小贝费摩斯

42楼 2015-08-14 14:28发表

好久没用异步任务了，现在是各种框架



_Zhijun

41楼 2015-06-16 09:10发表

doDownload 方法在哪里



zeroones-

Re: 2015-07-29

回复u010211650：那是象征性的写的一个方法，类似于伪代码



simpsonst

40楼 2015-05-14 15:15发表

笔者写得真的很好，当我自己实现了一个类似这样功能的东西以后，再回过头来看这篇文章就能更加深入理解AsyncTask了



baidu_26773051

39楼 2015-03-22 10:26发表

大神，关于asyncTask，我有两个问题没弄懂。

- 1.异步任务的实例必须在UI线程中创建
- 2.execute(Params... params)方法必须在UI线程中调用

这是为什么呢？



废墟的树

Re: 2015-04-22 09:15发表

回复baidu_26773051：AsyncTask里面有个 handler message消息处理机制-----特点就是谁发送消息就谁处理消息，我们一般用AsyncTask都是更新UI操作，更新UI就必须在UI线程中创建 Handler+Message消息机制，来让UI线程----也就是主线程的handler来发送和处理消息。



micheal_yejinglin

38楼 2015-03-18 15:53发表

为什么要有下面这句话？
Process.setThreadPriority(Process.THREAD_PRIORITY_

BACKGROUND);

为什么AsyncTask执行时进程都处于background优先级？



Wbaokang

37楼 2015-03-16 21:18发表

思路清晰顺畅，赞一个！



lipeiwei2015

36楼 2015-02-16 20:04发表

赞，郭大哥的博文写的很赞，我会一篇一篇地读下去的



谁来自江湖

35楼 2015-02-12 16:02发表

mark一下



Jux-L

34楼 2014-11-14

很不错，值得学习



ssbg2

33楼 2014-11-05 11:20发表

博主，我买了一本你的书，特别好，没有堆砌代码，讲解深度也是恰到好处。
希望能再写一本面向有一定经验的开发者的书啊。



IT_Transformers

32楼 2014-11-04 15:20发表

ThreadPoolExecutor 关于这个的参数解析好像有问题



ywdfly

31楼 2014-10-31 10:10发表

大神，讲的很清晰、透彻啊



李小四

30楼 2014-08-21 16:27发表

我在一个Activity中使用了内部类继承AsyncTask，进行下载，然后，我退出了该activity,进入了其他的activity，但是为什么下载没有停止？



Kerwin555

Re: 2016-09-04 10:42发表

回复son__of__sun：这个得手动cancel吧



qiuqingpo

29楼 2014-08-12 18:07发表

分析的真心觉得好好好



萌小杰

28楼 2014-08-01 17:11发表

博主，想问个问题，就是SerialExecutor 如你所说，要等前面的runable执行完了，才会取出下一个去执行，那里面的那个ThreadPoolExecutor是不是就没有办法同时执行多个任务了？因为他要执行完一个任务，才能触发下一次。但是本身ThreadPoolExecutor如果任务数小于corePoolSize不是可以立马建线程同时去跑那些任务吗，这样的话是不是意味着asyncTask没办法多个同时执行？ThreadPoolExecutor纯粹变得用来跑任务而已了呢？



兰亭风雨

Re: 2014-08-02 20:23发表

回复cjjmeng：是SerialExecutor使任务串行执行的，每个任务还是提交到THREAD_POOL_EXECUTOR 线程池中执行，但只能等上一个执行完，才能提交下一个，因此线程池中永远只有一个任务在执行，当然，可能上一个任务执行完会，线程空闲时间还没超出1秒，这样就不会被移除线程池，在线程数量小于corePoolSize时，会再次创建一个线程放入线程池来执行新的任务，即使有空闲线程，但是却永远只能有一个线程处于工作状态。

另外，要实现并行，可以用自定义的ThreadPoolExecutor，也可以用源码中提供的THREAD_POOL_EXECUTOR，它的corePoolSize是CPU的数目+1，maximumPoolSize是2*cpu数目+1，keepAlive时间是1秒，阻塞队列采用BlockingQueue，长度为128



萌小杰

Re: 2014-08-05 11:32发表

回复mmc_maodun：对了 博主 经过验证 貌似3.1之后才有SerialExecutor 不是3.0



兰亭风雨

Re: 2014-08-05 13:40发表

回复cjjmeng：你去看下官方的培训教程吧！博主也贴了原文，这个SDK文档的解释中也有说明，最好自己去看看！



萌小杰

Re: 2014-08-04 17:41发表

回复mmc_maodun：所以意思是说，如果用默认的SerialExecutor，线程池中，同一时刻就只有一条线程在跑？是这个意思吗？



2013_android_study

Re: 2016-05-18 15:32发表

回复cjjmeng：我也刚看这个源码，SerialExecutor是用来让所有任务串行取出。所有任务都会添加到ArrayDeque这个队列中来等待排队执行，真正执行是THREAD_POOL_EXECUTOR来执行任务。一个是任务排队用的，另一个是来执行任务的。并不是说THREAD_POOL_EXECUTOR 不能并发，只是任务没有并发提交到它。



newmandirl

27楼 2014-07-31 00:06发表

还不错，顶一个



兰亭风雨

26楼 2014-07-18 10:25发表

补充一下：如果有多个AsyncTask的话，除了公用同一个ThreadPool，也公用一个static的Handler。
一个AsyncTask对应于一个Thread



兰亭风雨

25楼 2014-07-18 10:13发表

nice！高并发下还是用Handler+Thread比较合适



安辉就是我

24楼 2014-07-17 15:13发表

看了楼主博客，学习了很多知识！~~~~~多谢！



安辉就是我

23楼 2014-07-17

看了楼主博客，学习了很多知识！~~~~~多谢！



Jason_kxs

22楼 2014-07-11 22:52发表

想请问下楼主这个Executor exec = new ThreadPoolExecutor(15, 200, 10, TimeUnit.SECONDS, new LinkedBlockingQueue<Runnable>());

10的参数有什么含义吗？



Jason_kxs

21楼 2014-07-11 22:52发表

看了楼主博客，学习了很多知识！~~~~~多谢！



帝丹11

20楼 2014-06-11 10:31发表

写的好！lz我是你的脑残粉！



蔡Sir

19楼 2014-04-03 15:43发表

顶



新靖界

18楼 2014-03-25 14:24发表

我创建了两个类都继承AsyncTask类，两个类功能参数都一样。主程序执行第一个类的execute()方法，doInBackground

d() 方法会执行，doInBackground这个方法里面有一个方法会永远运行下去的方法（如监控）；然后主程序执行第二个类的execute()方法，为什么第二个类里面的doInBackground方法不执行？



萌小杰

Re: 2014-08-01 15:43发表

回复lilidejing：个人理解是，AsyncTask里共享一个SerialExecutor，根据博主的说法和代码，但你第一个类被执行的时候，包着doInBackground的Runnable对象被添加到SerialExecutor维护的队列中，然后被取出并执行，然后就一直在执行，因为SerialExecutor的execute方法里是

```
try {  
    r.run();  
} finally {  
    scheduleNext();  
}
```

你那个方法就一直在那里r.run那里运行，无法执行下一句scheduleNext();也就无法取出下一Runnable，所以但你另一个AsyncTask执行是，往里面添加Runnable也不会被取出执行，一直呆在队列中，直到上一个方法执行完



yang7162082

Re: 2014-05-11

回复lilidejing：因为底层是共享一个static的thread pool,个人理解。



u014190303

17楼 2014-03-23 20:00发表

不错，学习一下



堺雅人联通

16楼 2014-03-18 23:00发表

不错的东西，很值得学习



yang7162082

15楼 2014-03-10 14:12发表

学习了



zhaoyu88312

14楼 2014-01-05 21:05发表

引用“zhaoyu88312”的评论：

```
mTasks.offer(new Runnable() {  
    public...
```

原来的 r.run就相当于调用普通类的方法一样了



zhaoyu88312

13楼 2014-01-05 20:59发表

```
mTasks.offer(new Runnable() {  
    public void run() {
```

```
try {  
    r.run();  
} finally {  
    scheduleNext();  
}  
}  
});  
竟然是 r.run(),把 传入过来的任务代码原封不动拿过来了。  
是吗？
```



Jairus_Tse

12楼 2013-12-22 14:11发表

这篇文章写得太好了，谢谢分享



山水之晨

11楼 2013-12-18 17:26发表

楼主很强大，赞一个，这篇文章是我见过关于AsyncTask讲的最好的，最透彻的。



bad-guy

10楼 2013-12-07

您好，我想问一下能不能对单个任务进行中断取消啊，是不是只能运用分片上传与下载？



低调小一

Re: 2014-08-13 16

回复ziyinghui123：可以啊，保持asynctask的对象，判断状态，为running状态cancel即可



fdsgsfdgsd

9楼 2013-10-19 00:00发表

不错，正在使用AsyncTask，更深入理解一下！



zhkx123

8楼 2013-10-14 10:35发表

mark



leehong2005

7楼 2013-10-13 03:30发表

多个asynctask运行的情况下，因为底层是共享一个static的thread pool，在这样的情况下，会不会存在问题？



guolin

Re: 2013-10-13 08:47发表

回复leehong2005：就相当于在使用线程池来管理线程的，不会出问题的。



leehong2005

Re: 2013-10-13 18:12发表

回复sinyu890807：你确定不会有问题？你的应用中的不同模块都有asynctask来执行时，有如果前面有很多后台task在执行，正时当前界面如果要执行一个asynctask的话，就会出问题吧。它会存在排队现象。所以，asynctask有他的局限性。



singsong

Re: 2014-07-09 12:53发表

回复leehong2005：这种情况如何是好？



guolin

Re: 2013-10-13 18:42发表

回复leehong2005：额，你既然这些问题都考虑到了，说明对AsyncTask了解的挺深了，我可没说AsyncTask是万能的，根据合适的场景灵活运行就好。



liuh6

6楼 2013-10-12 16:39发表

很干的干货。。支持斑竹



zhuhf_blog

5楼 2013-10-12 10:22发表

很好



xinyujay

Re: 2014-02-14

小朱不要调皮



itchenlin

4楼 2013-10-12 10:17发表

大神，您太给力了!!! 能不能建一个Q群，让更多人和您一起交流啊!!! 哈哈!



guolin

Re: 2013-10-18 11

回复itchenlin：已经建了，看侧面公告栏。



lost2x

3楼 2013-10-11 09:54发表

感谢楼主分享。想请教一下，比如要下载3张图片，并绑定到3个imageview，由于每次onPostExecute绑定的imageview不同，不得不创建3个不同的AsyncTask子类吗？怎样能写得通用一点？



guolin

Re: 2013-10-11 10:32发表

回复lost2x：传个参数到asynctask里面不就好了，用这个参数标识一下当前作用的是哪个imageview。



snwrking

2楼 2013-10-11 09:49发表

[java]

```
01. Executor exec = new ThreadPoolExecutor
    (15, 200, 10,
02.     TimeUnit.SECONDS, new LinkedBlockingQueue<Runnable>());
03. new DownloadTask().executeOnExecutor(exec);
```

这代码很好，这下可以同时运行多个AsyncTask了。可惜就



低调小一

Re: 2014-08-13 16:48发表

回复snwrking: 不理解为什么要这么干, Java的API官方文档里明确的建议使用Executors的工厂方式进行配置。Executors.newFixedThreadPool(5); 更简单也更好理解一些。



guolin

Re: 2013-10-11 10:30发表

回复snwrking: 不是哦, 4.0以上的系统已经占将近70%的份额了, 以后还会不断增长, 看下数据统计吧
<http://developer.android.com/about/dashboards/index.html>



gordon1986

1楼 2013-10-11 09:37发表

不错, 支持一个

您还没有登录,请[登录](#)或[注册](#)

* 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司

| 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 