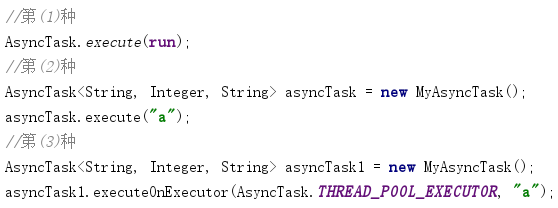
**Android异步任务**

<http://blog.csdn.net/luoshengyang/article/details/6905587>

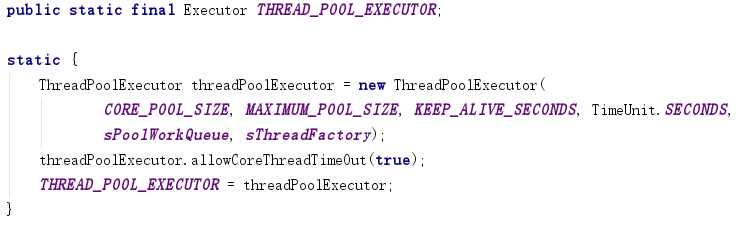
* **AsyncTask**

AsyncTask是一个轻量级异步任务类，它可以在线程池中执行后台任务，然后把执行的进度和最终的结果传递给主线程并在主线程中更新UI。与其他异步任务相比，它的优点在于异步执行任务的同时，能一直与UI线程打交道。

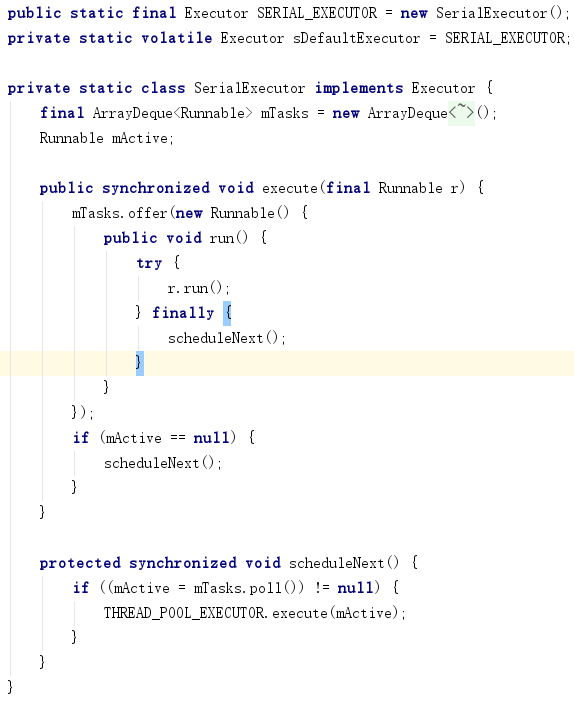
AsyncTask的结构比较特殊，其实AsyncTask内部是一个串行的线程池在执行异步任务，并且还有一个Handler来和UI线程交互。它有三种形式来执行异步任务：



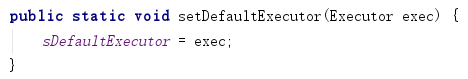
AsyncTask是一个异步抽象类，要想执行异步线程，要么使用继承它的子类并且实现doInBackground()方法；要么直接使用AsyncTask的静态方法execute()来执行异步任务。 不管哪种方式都是依靠内部的线程池来执行的。这个线程池是当AsyncTask类被第一次加载的时候就被创建，即在AsyncTask的静态代码块中被初始化：



生成一个公共的静态的线程池THREAD\_POOL\_EXECUTOR，这个线程池并不是一个串行的线程池，AsyncTask串行的执行任务是通过它的内部类SerialExecutor来操作的：



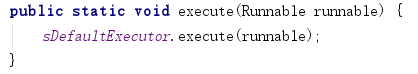
当AsyncTask类被加载的时候也会初始化另一个线程池SERIAL\_EXECUTOR，并设置给sDefaultExecutor。SerialExecutor也是一个线程池，这个线程池主要是将添加进来的任务存放到mTasks队列尾端，然后调用静态线程池THREAD\_POOL\_EXECUTOR执行队列头部的任务，当队列第一个任务执行完毕继续执行队列的下一个任务，通过SerialExecutor这个线程池就实现了AsyncTask串行的执行任务。如果想并发的执行任务可以通过下面方法来设置成AsyncTask.THREAD\_POOL\_EXECUTOR：



* **AsyncTask使用和源码分析**

通过AsyncTask的三种异步任务执行方式分析它的源码并且总结它的使用方式。

* **AsyncTask.execute(run);**



这种方式最简单，直接将需要执行的任务添加进串行线程池并调用静态的线程池THREAD\_POOL\_EXECUTOR行任务。

* **实现抽象类AsyncTask的异步任务**

**具体使用**

这种属于第二、三种执行方式。通过实现抽象类AsyncTask来执行异步任务，可以在UI线程获取任务的执行进度和执行结果。先来看看它们的使用方式然后分析源码的执行：

**class** MyAsyncTask **extends** AsyncTask<String, Integer, String> {  
  
 */\*\*  
 \** 任务执行调用此方法，在UI线程中执行，主要做一些准备工作。  
 *\*/* @Override  
 **protected void** onPreExecute() {  
 **super**.onPreExecute();  
 }  
  
 */\*\*  
 \** 在线程池中执行，用于执行异步任务，并返回任务的执行结果。  
 \* 在此方法中调用publishProgress方法并将计算好的任务进度传递进去，  
 \* publishProgress通过Handler发送携带着任务进度的消息通知UI线程  
 \* 调用AsyncTask的onProgressUpdate方法接收任务进度。 *\*/* @Override  
 **protected** String doInBackground(String... params) {  
 **try** {  
 **for** (**int** i = 0; i < 10; i++)  
 {  
 publishProgress(i);  
 Thread.*sleep*(1000);  
 }  
 } **catch** (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 **return "result"**;  
 }  
  
 */\*\*  
 \** 在主线程中执行，更新UI线程任务的执行进度。  
 *\*/* @Override  
 **protected void** onProgressUpdate(Integer... values) {  
 Log.*i*(**"--进度"**, **"\_"** + values[0]);  
 }  
  
 */\*\*  
 \** doInBackground（）执行完任务后返回结果，然后线程池执行任务的子线程  
 \* 通过Handler发送携带任务结果的消息通知UI线程调用AsyncTask的onPostExecute

方法。*\*/* @Override  
 **protected void** onPostExecute(String s) {  
 **super**.onPostExecute(s);  
 }  
 */\*\*  
 \** 当任务被取消后，会调用此方法并返回任务执行结果。  
 \* 如果执行super.onCancelled(s);则会调用onCancelled()方法。  
 *\*/* @Override  
 **protected void** onCancelled(String s) {  
 **super**.onCancelled(s);  
 }  
 */\*\*  
 \** 当任务被取消后调被用。  
 *\*/* @Override  
 **protected void** onCancelled() {  
 **super**.onCancelled();  
 }  
}

上面代码展示了实现抽象类AsyncTask来执行异步任务的详细使用。AsyncTask是个泛型类，它提供<Params, Progress, Result>三个泛型参数：

**Params**：代表调用通过实例化AsyncTask实现类的对象调用execute(Params... params)启动异步任务时传递进去的参数类型，这个传递进去的参数最终会传递到doInBackground()方法中。

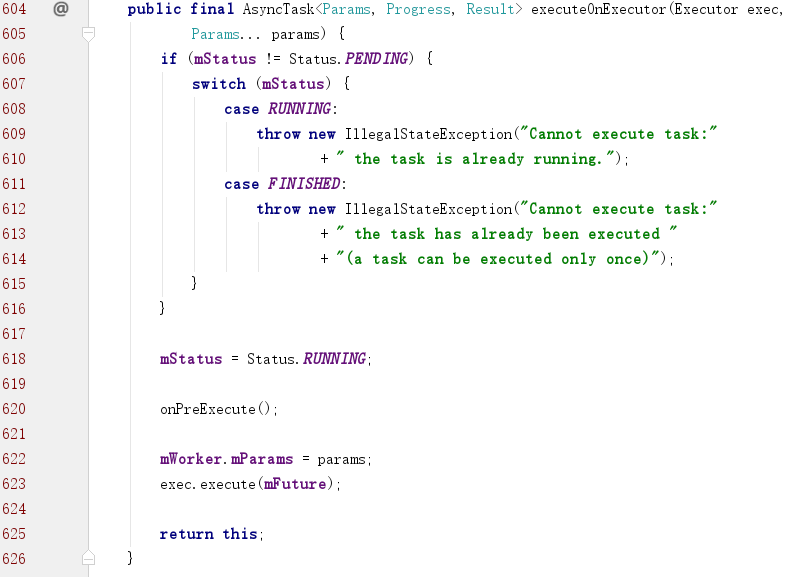
**Progress**：表示后台任务的执行进度的类型，即publishProgress()和onProgressUpdate()方法的参数类型。

**Result**：表示后台任务执行结果，即doInBackground()方法返回的类型。

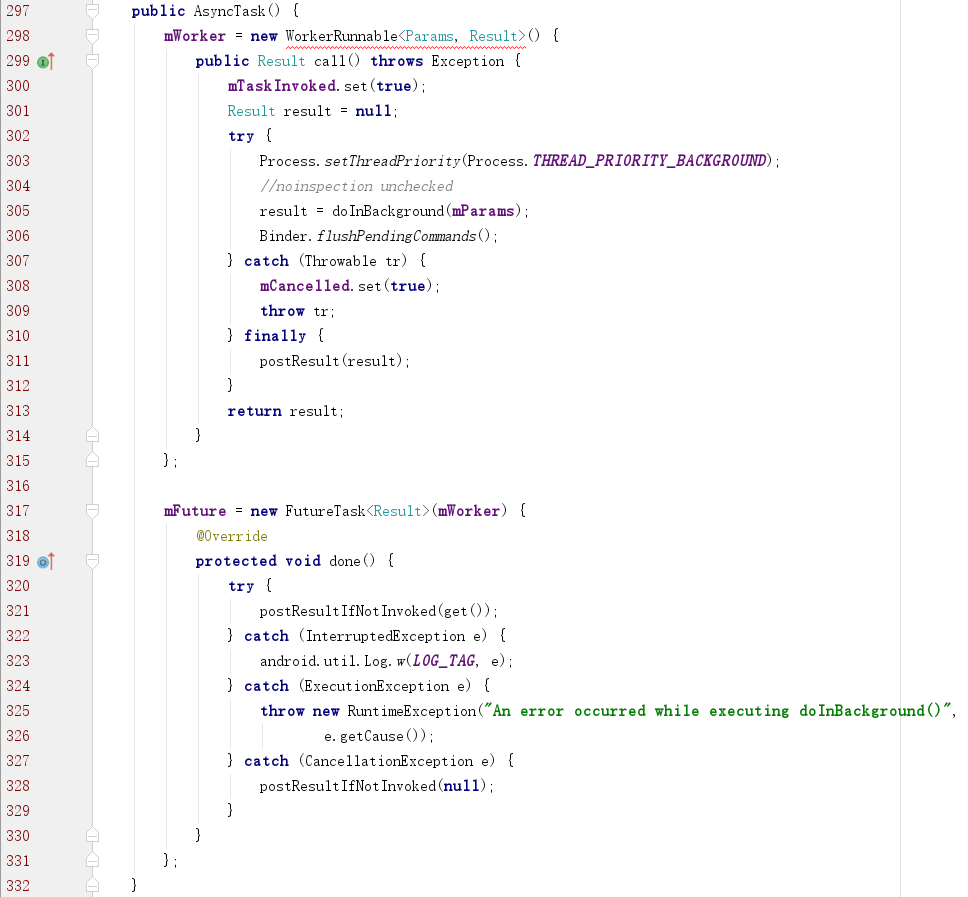
**启动任务**

**1500458073(1)**

第一种是通过AsyncTask默认的串行的线程池来执行异步任务。第二种是通过传递进去的并发式的线程池来执行异步任务。第一种启动方式中execute()方法内部也是直接调用的executeOnExecutor()方法，只不过是将AsyncTask的默认的串行线程池传递进去，因为AsyncTask的默认的串行线程池是个静态的对象，所以如果我们不传递自己的线程池进去，那么**我们创建的所有AsyncTask的实现类执行的异步任务都是在AsyncTask的默认的串行线程池串行执行的**：



先不看判断，直接看618行，进来后直接将状态设置为RUNNING，往上看606行的两个判断，如果状态为RUNNING或者FINISHED，就会抛出异常，即**一个AsyncTask实现类只能执行一个异步任务，只能调用一次execute()方法否则会抛出异常**。620行执行任务之前调用onPreExecute()方法做一些准备。622行将传进来的参数设置给mWorker。623行，假如设置的是其他的线程池，那么直接由其他的线程池执行任务，这里是默认的线程池执行异步任务，他执行的是一个mFuture对象，这个对象包括mWorker都是在构造方法中被创建：



AsyncTask类构造函数初始化了WorkerRunnable类实例mWorker和FutureTask类实例mFuture，并将mWorker传递给mFuture对象。WorkerRunnable类是AsynTask类的一个内部类并且实现了Callable接口，它里面有一个Params[]数组来存储execute()方法传递进来的参数。从executeOnExecutor()方法中知道线程池执行的是mFuture对象（FutureTask在线程池篇讲的很详细），其实他执行的是mFuture对象中的run方法，然后run方法又会执行传递进来的mWorker对象的call方法。我们看305行，在call方法中执行的是doInBackGround方法，在doInBackGround方法中我们又可以调用publishProgress方法来通知UI线程任务执行进度，然后在onProgressUpdate方法中更新UI界面。执行完doInBackGround方法，看311行通过postResult方法将任务的执行结果传递给UI线程：

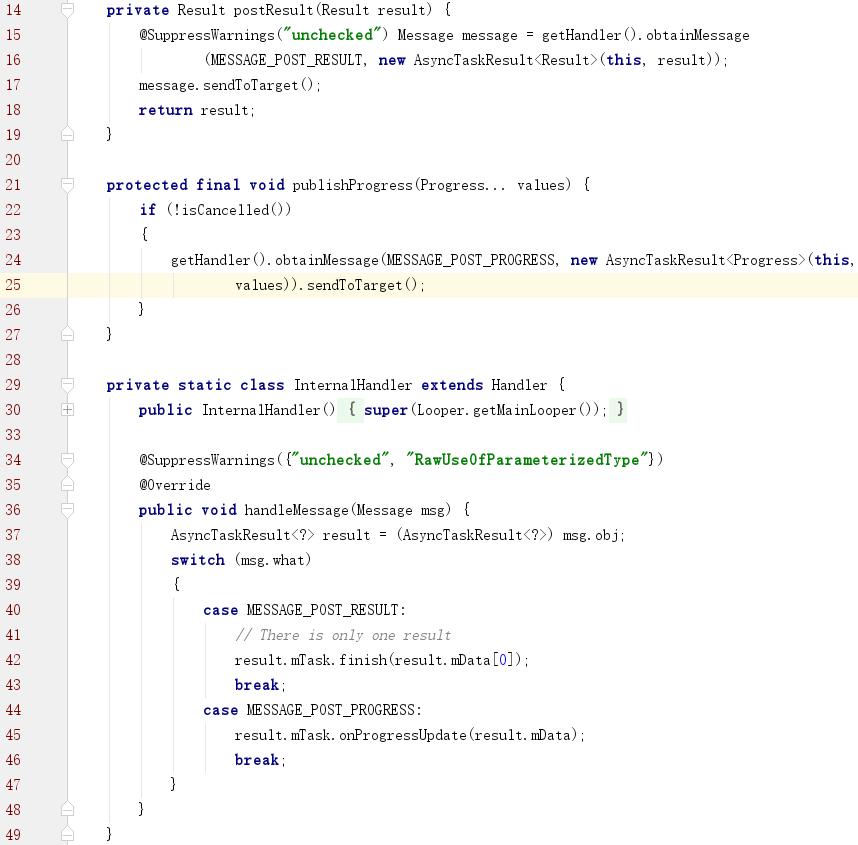
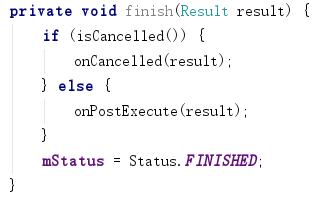


图1

15行和24行通过Handler发送消息携带的都是AsyncTaskResult的实例，它是一个静态内部类，它主要存储当前AsyncTask的对象和异步任务返回到UI线程的数据。这里我们主要看42行，AsyncTaskResult的mTask是指向当前AsyncTask实例化对象的，所以当异步任务执行完后会在UI线程调用AsyncTask的finish()方法：



AsyncTask实例的异步任务开始后是可以通过调用cancel()方法来取消任务的，假如任务已经取消那么最终会调用onCancelled()方法，他是一个空方法，在子类中可以被重写来处理一些任务取消后的事情。如果没有取消任务，则调用onPostExecute()方法通知UI线程执行完毕并返回任务结果。

InternalHandler就是我们用来向主线程发送消息的Handler。看图1中30行它的构造方法中初始化的是主线程的Looper，这个Looper就是用来遍历并处理主线程中的消息的。所以**不管我们从什么线程创建和执行AsyncTask，它总是能更新主线程的UI。**