详解 Android AsyncTask 用法与原理

chen_yip 关注

0.328 2017.05.31 18:48:49 字数 3,909 阅读 8,453

文章简介

AsyncTask 是Android 开发一个常用的多线程异步任务组件。网上资料很多也很杂，所以我决定整理一些关于AsyncTask必须知道的一些知识点，包括基本用法，实现原理以及和手动开Handler处理多线程的区别。入门的话先看用法就OK，耐心看完想必对你会有帮助。不过在进入正文之前先做个简单的AsyncTask背景介绍。

为什么要用AsyncTask

Android 开发中有两个很基本也很重要的常识

1.主线程不能执行耗时操作，否则该安卓程序会出现 Application Not Response异常，这也就是著名的ANR。(在Activity中操作超过 5s 会出现ANR，Broadcast Receiver 和 Service 分别是 10s 和 20s)
2.只有UI线程(也就是常说的主线程)才能执行UI操作，否则会抛出异常。所谓UI操作，也就是调用更新view组件内容的方法，举个例子：

```
newThread()
{
@Override
public void run() {
try{
 sleep(1000);
}catch(InterruptedException e){
 e.printStackTrace();
}
 textView.setText("Touch View Method");
}
}.start();
```

这里的textView是一个TextView组件，调用TextView的setText()方法就是执行了UI操作。程序会因为这段在子线程中执行UI操作的代码而抛出异常:
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

*另外，涉及到多线程的异步操作，稍不留神可能会引起内存泄漏，这篇文章要介绍的AsyncTask也不例外，但是内存泄漏不是只言片语能讲清楚的，目前关于这方面的知识可以先参考我这篇关于handler的泄漏，其实AsyncTask泄漏的本质一样。
http://www.jianshu.com/p/9440937263f2

•

正因为有了上面的两点规定，Android开发过程需要着重关注任务的分配：什么样的操作应该新开启一条线程来处理，什么样的操作应该在主线程中执行。AsyncTask 组件就是为了方便程序员高效解决这种任务分配问题而存在的。它可以通过重写特定方法体来指定特定的线程来执行需要执行的逻辑代码。有了这些背景，现在可以进入正文啦。

AsyncTask的基本用法

先看看最精简的AsyncTask的代码大概长什么样子,然后慢慢解释:

```
class ReverseTask extends AsyncTask<String,Integer,String> {
```

推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

银企直连的清算故事

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

```
    protected String doInBackground(String... params) {
        int progress = 0;
        try {
            for(int i = 0;i <= 10;i ++){
                progress += 10;
                Thread.sleep(200);
                publishProgress(progress);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return Reverse(params[0]);
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        progressDialog.setMessage(values[0]+"% Finished");
    }

    @Override
    protected void onPostExecute(String s) {
        textView.setText(s);
        progressDialog.dismiss();
    }
}
```

Reverse()代码:

```
String Reverse(String string){
    StringBuilder builder = new StringBuilder(string);
    return new String(builder.reverse());
}
```

最后执行这个任务:

```
ReverseTask reverseTask = new ReverseTask();
reverseTask.execute(text);
```

AsynTask是一个抽象类，使用AsynTask需要自定义自己的任务类并继承AsynTask。先不着急看每个方法的作用，先看看参数类型是怎么回事。

AsyncTask<Params, Progress, Result> (android.os)

AsyncTask.png

AsyncTask接受三个泛型参数类型, 这三个参数都是继承自object类:

1. Params: 对应示例代码的第一个String。
   这个参数有两个地方有用到
   (1)用于指定示例代码中 doInBackground(String... params) 方法中的参数类型, 在这里因为Params指定的是 String，所以doInBackground中的参数类型是String。
   (2)用于指定执行任务 reverseTask.execute(text)时传入的参数类型, 示例代码中的text就是String类型的。
   但是要注意，doInBackground中的params是一个不定长参数，它可以接收任意个String类型的参数。

2. Progress: 对应示例代码中的Integer。
   这个参数用于指定onProgressUpdate(Integer... values)方法的参数类型。其实也就是指定后台任务的进度单位。这里指定的是Integer，表示用整型来指定进度单位，如10%,20%,..,90%,100%。同样地，这里的values也是一个不定长参数。(注意，这里不能使用int，因为AsyncTask这三个泛型参数都继承自object类)

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

类型。在这里指定的是String，表示doInBackground结束后会返回一个String类型的数据到onPostExecute中，接下来 onPostExecute就可以使用这个返回的数据来做UI更新的逻辑了。

接下来看看每个方法的作用吧

1. onPreExecute(): 主线程执行
   这个方法是AsyncTask最先执行的方法，在这里一般可以做一些简单的UI初始化操作，如进度条的显示等。

2. doInBackground(Params... params): 子线程执行
   这个方法是AsyncTask一个最重要的方法。这个方法不会由主线程来执行，因此可以把耗时的逻辑操作交给这个方法来处理。当这个方法结束后，能够把处理完的结果返回到主线程中。这时，主线程中的界面元素就能够使用该返回结果更新数据了。

3. publishProgress(Progress... values): 子线程执行
   这个方法用于通知任务进度的更新情况。举个例子，比如你想每完成10%就更新一次进度条，那就可以每隔10%调用一次这个函数来告诉AsynTask进度条可以更新了，AsynTask得到这个通知后就会尝试去执onProgressUpdate这个函数来更新进度显示了。

4. onProgressUpdate(Progress... values): 主线程执行
   这个方法用于更新进度的显示，如进度条的更新。
   (看到这里，聪明的你肯定发现了为什么不能够在doInBackground中直接调用onProgressUpdate方法来更新进度条，而是要那么麻烦绕一个圈子调用publishProgress来通知asyntask完成这件事。没错，这个方法是在主线程中执行的，而doInBackground是在子线程中执行的，自然doInBackground是不能够触碰进度条这种UI组件的，因此要交给AsynTask的内部逻辑去完成这个线程的切换。*参见文章开头的背景知识)

5. onPostExecute(Result result): 主线程执行
   这个方法用于UI组件的更新。doInBackground返回的结果会被传到这个方法中作为更新UI的数据。

到这里，我们可以看出示例代码中的逻辑其实很简单。就是需要把一段传进来的文本逆序然后用一个TextView 来显示。
文本逆序的过程属于耗时操作，于是把它放在了doInBackground中。
textView的文本设置属于UI组件操作，于是把它放在了onPostExecute中。
当doInBackground完成逆序操作后会把逆序后的文本交给onPostExecute，然后textView使用这段文本更新自己的text内容。

另外，在逆序开始前，我们显示了一个progressDialog。并且在doInBackground的逻辑里添加了更新进度的逻辑。可以看到，这里更新进度只是让线程循环sleep 10次，每次更新一次进度。当然，这样的逻辑只是为了方便演示如何使用AsyncTask。
实际上，如果后台操作是用户不可见的，也就是不需要显示进度条的，上面的示例代码完全可以浓缩成这样

```
class ReverseTask extends AsyncTask<String,Integer,String>{z
    @Override
    protected String doInBackground(String... params) {
        return Reverse(params[0]);
    }

    @Override
```

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

怎么样，代码是不是一下精简多了，相信这段缩减后的代码不用多说你也一眼就能看明白了。

- 最后要提一点，如果像本文一样直接使用内部类引用Activity的组件来更新UI，切记，为了防止内存泄漏，要在Activity的onDestroy()中终止AsyncTask的任务，通过代码逻辑或者强制关闭任务，例如 ：
  cancel(boolean mayInterruptIfRunning)

```
reverseTask.cancel(true);
```

使用AsyncTask建议不要使用内部类，建议在外部定义一个task类，task中使用弱引用来指向Activity，通过这种方法来更新UI就能很大程度降低泄漏的危险。这里就不细讲了。

AsyncTask基本的用法就到这里结束啦，接下来让我们探索一下这个组件到底是怎么运作和实现的，使用这个组件有哪些地方需要注意。

AsyncTask的实现原理

要看懂这章首先希望你对java并发包的Executor线程池, Callable以及FutureTask有了解。如果没问题，那就开始吧。

要弄清楚AsyncTask是怎么运作的，那必须得看看AsyncTask的源码是怎么写的了。但是莫慌，我们不需要看懂每一个细节，大致了解内部的逻辑跳转就好了。
于是我们不需要从头开始看AsyncTask的代码。由于一旦我们调用task.execute()，任务就会开始执行，所以我们首先想知道的一定是执行任务的方法execute(Params... params)是怎样组织AsyncTask内部的一系列函数调用的，于是我们对准execute()方法,ctrl+左键进入。能看见如下代码:

```
* @see #executeOnExecutor(java.util.concurrent.Executor, Object[])
@MainThread
public final AsyncTask<Params, Progress, Result> execute(Params... params) {
   return executeOnExecutor(sDefaultExecutor, params);
}
```

可以看到，其实execute(Params... params)调用的是另外一个重载过的execute(java.util.concurrent.Executor, Object[])。
sDefaultExecutor是Android自己实现的一个线程池，稍后我们会看见它的真身。再次对准这个execute，ctrl+左键进入。

```
public final AsyncTask<Params, Progress, Result> executeOnExecutor(
            Executor exec, Params... params) {
    if (mStatus != AsyncTask.Status.PENDING) {
        switch (mStatus) {
            case RUNNING:
                throw new IllegalStateException("Cannot execute task:"
                        + " the task is already running.");
            case FINISHED:
                throw new IllegalStateException("Cannot execute task:"
                        + " the task has already been executed "
                        + "(a task can be executed only once)");
        }
    }

    mStatus = AsyncTask.Status.RUNNING;

    onPreExecute();

    mWorker.mParams = params;
    exec.execute(mFuture);

    return this;
}
```

推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

验证了这个函数执行在主线程中。接下来出现了一个mWorker的变量，这个变量在AsyncTask的私有变量中能看到声明

```
private final WorkerRunnable<Params, Result> mWorker;
private static abstract class WorkerRunnable<Params, Result> implements Callable<Resul
    Params[] mParams;
}
```

也就是说mWorker是一个callable变量(类似于runnable，它可以返回结果)，它的工作将在子线程中进行。
最后进入到exec.execute(mFuture)。exec，看上面，是参数列表里的Executor。mFuture,同样看AsyncTask的私有变量声明:

```
private final FutureTask<Result> mFuture;
mFuture = new FutureTask<Result>(mWorker) {
    @Override
    protected void done() {
        try {
            postResultIfNotInvoked(get());
        } catch (InterruptedException e) {
            android.util.Log.w(LOG_TAG, e);
        } catch (ExecutionException e) {
           throw new RuntimeException("An error occurred while executing doInBackgroun
                       e.getCause());
        } catch (CancellationException e) {
            postResultIfNotInvoked(null);
        }
    }
};
```

它是一个FutureTask，FutureTask用于支持callabe以实现多线程操作。知道这句话的两个变量后，我们能得出一个结论，execute(Params... params)又将转辗到这个Executor中的execute方法中，并且传入了一个futuretask。

接下来我们要怎么找出这个execute方法呢？既然exec是传进来的Executor,仔细回想一下，那个Executor上文提到过，就是sDefaultExecutor。那我们赶紧找找这个sDefaultExecutor是怎么实现的。

```
private static volatile Executor sDefaultExecutor = SERIAL_EXECUTOR;
private static class SerialExecutor implements Executor {
    final ArrayDeque<Runnable> mTasks = new ArrayDeque<Runnable>();
    Runnable mActive;

    public synchronized void execute(final Runnable r) {
        mTasks.offer(new Runnable() {
            public void run() {
                try {
                    r.run();
                } finally {
                    scheduleNext();
                }
        });
        if (mActive == null) {
            scheduleNext();
        }
    }

    protected synchronized void scheduleNext() {
         if ((mActive = mTasks.poll()) != null) {
            THREAD_POOL_EXECUTOR.execute(mActive);
        }
    }
 }
```

写了这么长一段，如果我们只关注它主体的话，实际上就是执行了r.run()。

推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

源码我就不贴出来了，有兴趣可以自己进去看看)。
所以我们现在要找的是这个callable的call()是怎么实现的。
这个callable是什么？上面提到过

```
mFuture = new FutureTask<Result>(mWorker)
```

所以callable是mworker，那call()方法呢？在AsyncTask的构造函数里面。

```
mWorker = new WorkerRunnable<Params, Result>() {
        public Result call() throws Exception {
           mTaskInvoked.set(true);

           Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
                //noinspection unchecked
            Result result = doInBackground(mParams);
            Binder.flushPendingCommands();
            return postResult(result);
        }
    };
```

终于看到另一个熟悉的方法了! 没错,就是doInBackground(mParams)
到这里，也就验证了doInBackground(mParams)是在FutureTask的子线程中执行的。那么我们接着寻找别的方法。从postResult(result) ctrl+左键进去。

```
private Result postResult(Result result) {
        @SuppressWarnings("unchecked")
    Message message = getHandler().obtainMessage(MESSAGE_POST_RESULT,
    new AsyncTaskResult<Result>(this, result))
    message.sendToTarget();
    return result;
}
```

也就是说，这个postResult其实是把result构造进一个Message对象，然后通过getHandler()获取一个handler处理这个消息。Handler处理消息其实也就是handleMessage()的重写了。所以，我们现在应该去看看这个handler的实现。ctrl+左键 getHandler()

```
private static Handler getHandler() {
    synchronized (AsyncTask.class) {
        if (sHandler == null) {
            sHandler = new InternalHandler();
        }
        return sHandler;
    }
}
```

只不过是一个单例InternalHandler的对象获取，再进去

```
private static class InternalHandler extends Handler {
        public InternalHandler() {
            super(Looper.getMainLooper());
        }

        @SuppressWarnings({"unchecked", "RawUseOfParameterizedType"})
        @Override
        public void handleMessage(Message msg) {
            AsyncTaskResult<?> result = (AsyncTaskResult<?>) msg.obj;
            switch (msg.what) {
                case MESSAGE_POST_RESULT:
                    // There is only one result
                    result.mTask.finish(result.mData[0]);
                    break;
                case MESSAGE_POST_PROGRESS:
                    result.mTask.onProgressUpdate(result.mData)
                    break;
```

推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

onProgressUpdate(result.mData)。
这个handler是绑定在AsyncTask线程中的，也就验证了onProgressUpdate是在主线程中执行的。不妨回去看看publishProgress函数

```
protected final void publishProgress(Progress... values) {
    if (!isCancelled()) {
        getHandler().obtainMessage(MESSAGE_POST_PROGRESS,
                new AsyncTaskResult<Progress>(this, values)).sendToTarget();
    }
}
```

回想我们上一章讲到的，在doInBackground中调用的是publishProgress来更新进度。在这里就完全匹配上了，其实AsyncTask也不过是把更新进度的包装成一个MESSAGE_POST_PROGRESS的Message然后通过handler来回到主线程处理进度条更新罢了。
最后我们猜也能够猜得出，onPostExecute()一定会在finish()函数里面。ctrl+左键进去

```
private void finish(Result result) {
    if (isCancelled()) {
        onCancelled(result);
    } else {
        onPostExecute(result);
    }
    mStatus = Status.FINISHED
  }
```

不出所料，onPostExecute是执行在AsynTask的finish中的，所以也在主线程。到这里我们就完全弄清楚AsynTask的运作了。不妨稍微总结一下:

用户调用task.execute() =>
onPreExecute()=>
交给线程池sDefaultExecutor调度=>
mFuture 配合 mWorker开启子线程=>
doInBackground()=>
交给内部单例InternalHandler处理返回结果并返回到主线程=>
根据Message处理onProgressUpdate()或onPostExecute()

AsyncTask的实现和运作就到这里结束啦，希望这段探索源码的过程能对你有帮助。最后我们稍微看看AsyncTask与手动开Handler的区别。

AsyncTask与手动开Handler的区别
其实Android多线程处理很常用的一个方法就是自己实现一个handler来异步处理消息。
优点： 这样的做法非常直观明白，而且代码弹性大，程序员有很大的掌控权。
缺点： 但是当线程和消息多起来了，不仅管理不方便，效率也可能产生问题，因为大量的线程在不断地出生和死亡，没有任何复用。

AsyncTask的优缺点也很明显
优点: 使用线程池管理多线程，资源利用和效率高，管理方便。通过第二章的介绍，我们知道其实AsyncTask的execute是可以传进一个Executor对象作为参数的。也就是说我们甚至可以自己实现自己的线程池来配套AsyncTask处理多线程问题。
缺点：如果第二章看的认真的话，不难发现AsyncTask的default Exectuor是一个Serial_Executor并且这个线程池是设为Static的串行管理线程池。也就是说，如果你使用默认的asynctask, 无论你开了多少个AsyncTask对象，所有这些对象其实是共用一个线程池的，而且这个线程池的策略只是很简单地按序一个个处理。

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

到这里我就把我对AsyncTask的见解讲完啦，希望看完这篇文章对你有帮助吧。

12人点赞 > Android

chen_yip 香港大学计算机科学在读研究生，未来的程序员一枚
总资产1 共写了1.9W字 获得43个赞 共16个粉丝 关注

39个大数据可视化工具

被以下专题收入，发现更多相似内容

Android Android开发 Java& A...

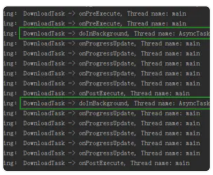
推荐阅读

更多精彩内容 >

Android中AsyncTask使用详解

在Android中我们可以通过Thread+Handler实现多线程通信，一种经典的使用场景是：在新线程中进行耗时...

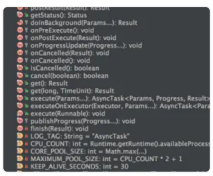
吕侯爷 阅读 1,391 评论 2 赞 23



Android Handler机制13之AsyncTask源码解析

Android Handler机制系列文章整体内容如下： Android Handler机制1之ThreadAnd...

隔壁老李头 阅读 2,313 评论 1 赞 14



推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

写下你的评论... 评论0 赞12

详解 Android AsyncTask 用法与原理

chen_yip 关注 赞赏支持

AsyncTask

Android开发者：你真的会用AsyncTask吗？ 导读.1 在Android应用开发中，我们需要时刻注意保证...

cxm11 阅读 2,102 评论 0 赞 29

Android中的线程状态之AsyncTask详解

由于Android的特性，如果要执行耗时操作，则必须方法子线程中执行。除了Thread可以开启子线程外，Andro...

Ruheng 阅读 21,000 评论 5 赞 19

相伴

你穿着我最爱看的红色衣服静静地在轮椅里坐着 我背着你最爱带的粉色布包慢慢地在后面推着 相伴左右是我们一生的浪漫

乐四月 阅读 90 评论 3 赞 1

推荐阅读

android AsyncTask 源码分析
阅读 74

Android 多线程：手把手教你使用AsyncTask
阅读 104

浅谈Android中多线程切换的几种方法
阅读 143

android点三
阅读 299

iOS刨根问底-深入理解RunLoop
阅读 245

写下你的评论... 评论0 赞12