

在Android中实现异步任务机制有两种方式，**Handler**和**AsyncTask**。

Handler已经在上一篇文章 [异步消息处理机制（Handler、Looper、MessageQueue）源码解析](#) 说过了。

本篇就说说AsyncTask的异步实现。

1、什么时候使用 **AsnyncTask**

在上一篇文章已经说了，主线程主要负责控制UI页面的显示、更新、交互等。为了有更好的用户体验，UI线程中的操作要求越短越好。

我们把耗时的操作（例如网络请求、数据库操作、复杂计算）放到单独的子线程中操作，以避免主线程的阻塞。但是在子线程中不能更新UI界面，这时候需要使用handler。

但如果耗时的操作太多，那么我们需要开启太多的子线程，这就会给系统带来巨大的负担，随之也会带来性能方面的问题。在这种情况下我们就可以考虑使用类**AsyncTask**来异步执行任务，不需要子线程和handler，就可以完成异步操作和刷新UI。

不要随意使用**AsyncTask**,除非你必须要与UI线程交互.默认情况下使用**Thread**即可,要注意需要将线程优先级调低.**AsyncTask**适合处理短时间的操作,长时间的操作,比如下载一个很大的视频,这就需要你使用自己的线程来下载,不管是断点下载还是其它的。

2、**AsnyncTask**原理

AsyncTask主要有二个部分：一个是与主线程的交互，另一个就是线程的管理调度。虽然可能多个**AsyncTask**的子类的实例，但是**AsyncTask**的内部**Handler**和**ThreadPoolExecutor**都是进程范围内共享的，其都是**static**的，也即属于类的，类的属性的作用范围是**CLASSPATH**，因为一个进程一个**VM**，所以是**AsyncTask**控制着进程范围内所有的子类实例。

AsyncTask内部会创建一个进程作用域的线程池来管理要运行的任务，也就是说当你调用了**AsyncTask**的**execute()**方法后，**AsyncTask**会把任务交给线程池，由线程池来管理创建**Thread**和运行**Therad**。

3、**AsyncTask**介绍

Android的**AsyncTask**比**Handler**更轻量级一些（只是代码上轻量一些，而实际上要比**handler**更耗资源），适用于简单的异步处理。

Android之所以有**Handler**和**AsyncTask**，都是为了不阻塞主线程（UI线程），因为UI的更新只能在主线程中完成，因此异步处理是不可避免的。

AsyncTask：对线程间的通讯做了包装，是后台线程和UI线程可以简易通讯：后台线程执行异步任务，将**result**告知UI线程。

使用**AsyncTask**分为两步：

① 继承**AsyncTask**类实现自己的类

```
public abstract class AsyncTask<Params, Progress, Result> {
```

Params: 输入参数，对应**excute()**方法中传递的参数。如果不需要传递参数，则直接设为**void**即可。

Progress: 后台任务执行的百分比

Result: 返回值类型，和**doInBackground()**方法的返回值类型保持一致。

②复写方法

最少要重写以下这两个方法：

- **doInBackground(Params...)**

在子线程（其他方法都在主线程执行）中执行比较耗时的操作，不能更新UI，可以在该方法中调用**publishProgress(Progress...)**来更新任务的进度。**Progress**方法是**AsycTask**中一个**final**方法只能调用不能重写。

- **onPostExecute(Result)**

使用在**doInBackground**得到的结果处理操作UI，在主线程执行，任务执行的结果作为此方法的参数返回。有时根据需求还要实现以下三个方法：

- onProgressUpdate(Progress...)

可以使用进度条增加用户体验度。此方法在主线程执行，用于显示任务执行的进度。

- onPreExecute()

这里是最终用户调用Execute时的接口，当任务执行之前开始调用此方法，可以在这里显示进度对话框。

- onCancelled()

用户调用取消时，要做的操作

4、AsyncTask示例

按照上面的步骤定义自己的异步类：

```
public class MyTask extends AsyncTask<String, Integer, String> {
    //执行的第一个方法用于在执行后台任务前做一些UI操作
    @Override
    protected void onPreExecute() {

    }

    //第二个执行方法,在onPreExecute()后执行,用于后台任务,不可在此方法内修改UI
    @Override
    protected String doInBackground(String... params) {
        //处理耗时操作
        return "后台任务执行完毕";
    }

    /*这个函数在doInBackground调用publishProgress(int i)时触发,虽然调用时只有一个参数
    但是这里取到的是一个数组,所以要用progresses[0]来取值
    第n个参数就用progress[n]来取值 */
    @Override
    protected void onProgressUpdate(Integer... progresses) {
        //"loading..." + progresses[0] + "%"
        super.onProgressUpdate(progresses);
    }

    /*doInBackground返回时触发,换句话说,就是doInBackground执行完后触发
    这里的result就是上面doInBackground执行后的返回值,所以这里是"后台任务执行完毕" */
    @Override
    protected void onPostExecute(String result) {

    }

    //onCancelled方法用于在取消执行中的任务时更改UI
    @Override
    protected void onCancelled() {

    }
}
```

在主线程申明该类的对象，调用对象的execute（）函数开始执行。

```
MyTask t = new MyTask();
t.execute(); //这里没有参数
```

5、使用AsyncTask需要注意的地方

- AsyncTask内部的Handler需要和主线程交互，所以AsyncTask的实例必须在UI线程中创建
- AsyncTaskResult的doInBackground(mParams)方法执行异步任务运行在子线程中，其他方法运行在主线程中，可以操作UI组件。
- 一个AsyncTask任务只能被执行一次。
- 运行中可以随时调用AsyncTask对象的cancel(boolean)方法取消任务，如果成功，调用isCancelled()会返回true，并且不会执行onPostExecute()方法了，而是执行onCancelled()方法。
- 对于想要立即开始执行的异步任务，要么直接使用Thread，要么单独创建线程池提供给AsyncTask。默认的AsyncTask不一定会

立即执行你的任务，除非你提供给他一个单独的线程池。如果不与主线程交互，直接创建一个**Thread**就可以了。