

AsyncTask

前言

- 多线程的应用在Android开发中是非常常见的，常用方法主要有：
 1. 继承Thread类
 2. 实现Runnable接口
 3. Handler
 4. AsyncTask
 5. HandlerThread

目录



)

1. 定义

- 一个 `Android` 已封装好的轻量级异步类
- 属于抽象类，即使用时需 实现子类

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

2. 作用

1. 实现多线程

在工作线程中执行任务，如 耗时任务

2. 异步通信、消息传递

实现工作线程 & 主线程（UI 线程）之间的通信，即：将工作线程的执行结果传递给主线程，从而在主线程中执行相关的 UI 操作；

从而保证线程安全

3. 优点

- 方便实现异步通信
不需使用“任务线程（如继承 `Thread` 类）+ `Handler`”的复杂组合
- 节省资源
采用线程池的缓存线程 + 复用线程，避免了频繁创建 & 销毁线程所带来的系统资源开销

4. 类 & 方法介绍

4.1 类定义

`AsyncTask` 类属于抽象类，即使用时需 实现子类

```
public abstract class AsyncTask<Params, Progress, Result> {  
    ...  
}  
  
// 类中参数为3种泛型类型  
// 整体作用：控制AsyncTask子类执行线程任务时各个阶段的返回类型  
// 具体说明：  
//   a. Params：开始异步任务执行时传入的参数类型，对应execute()中传递的参数  
//   b. Progress：异步任务执行过程中，返回下载进度值的类型  
//   c. Result：异步任务执行完成后，返回的结果类型，与doInBackground()的返回值类型保持一致  
// 注：  
//   a. 使用时并不是所有类型都被使用  
//   b. 若无被使用，可用java.lang.Void类型代替  
//   c. 若有不同业务，需额外再写1个AsyncTask的子类  
}
```

4.2 核心方法

- `AsyncTask` 核心 & 常用的方法如下：

| 核心方法 | 作用 | 调用时刻 | 备注 |
|------------------------------------|--|--|--|
| execute(Params... params) | 触发 执行异步线程任务 | 手动调用 | <ul style="list-style-type: none">• 必须在UI线程中调用• 运行在主线程 |
| onPreExecute() | 执行 线程任务前的操作 (根据需求复写) | <ul style="list-style-type: none">• 执行线程任务前 自动调用• 即 执行execute () 前 自动调用 (不能手动调用, 需让系统自动调用) | 用于界面的初始化操作, 如 显示进度条的对话框 |
| doInBackground (Params params) | <ul style="list-style-type: none">• 接收输入参数• 执行任务中的耗时操作 (必须复写, 从而自定义线程任务)• 返回 线程任务执行的结果 | <ul style="list-style-type: none">• 执行线程任务时 自动调用• 即 onPreExecute () 执行完成后 自动调用 (不能手动调用, 需让系统自动调用) | <ul style="list-style-type: none">• 不能更改UI组件的信息• 执行过程中, 可用publishProgress () 更新进度信息 |
| onProgressUpdate (Progress values) | 在主线程 显示线程任务执行的进度 (根据需求复写) | 调用publishProgress(Progress... values)时 自动调用 (不能手动调用, 需让系统自动调用) | / |
| onPostExecute (Result result) | <ul style="list-style-type: none">• 接收线程任务执行结果• 将执行结果显示到UI组件 (必须复写, 从而自定义UI操作) | 线程任务结束时 自动调用 (不能手动调用, 需让系统自动调用) | / |
| onCancelled () | 将异步任务设置为: 取消状态 (并不是真正的取消任务) | 异步任务被取消时 即自动调用 (需在doInBackground () 中判断终止任务) | 该方法被调用时, onPostExecute () 就不会被调用 |

• 方法执行顺序如下

| 类型 | 具体执行顺序 |
|------------|--|
| 基础使用 | (手动调用) execute () --> onPreExecute () --> doInBackground () --> onPostExecute () |
| 需显示进度 | (手动调用) execute () --> onPreExecute () -->doInBackground () --> publishProgress() --> onPostExecute () --> onProgressUpdate() |
| 执行线程任务时需终止 | (手动调用) execute () --> onPreExecute () --> doInBackground () --> onCancelled () |

5. 使用步骤

- AsyncTask 的使用步骤有3个:
 1. 创建 AsyncTask 子类 & 根据需求实现核心方法
 2. 创建 AsyncTask 子类的实例对象 (即 任务实例)
 3. 手动调用 execute () 从而执行异步线程任务

具体介绍如下

```
/**
 * 步骤1: 创建AsyncTask子类
 * 注:
 *   a. 继承AsyncTask类
 *   b. 为3个泛型参数指定类型; 若不使用, 可用java.lang.Void类型代替
 *   c. 根据需求, 在AsyncTask子类内实现核心方法
 */

private class MyTask extends AsyncTask<Params, Progress, Result> {

    ....

    // 方法1: onPreExecute ()
    // 作用: 执行 线程任务前的操作
    // 注: 根据需求复写
    @Override
    protected void onPreExecute() {
        ...
    }
}
```

```

    }

    // 方法2: doInBackground ()
    // 作用: 接收输入参数、执行任务中的耗时操作、返回 线程任务执行的结果
    // 注: 必须复写, 从而自定义线程任务
    @Override
    protected String doInBackground(String... params) {

        ...// 自定义的线程任务

        // 可调用publishProgress () 显示进度, 之后将执行onProgressUpdate ()
        publishProgress(count);

    }

    // 方法3: onProgressUpdate ()
    // 作用: 在主线程 显示线程任务执行的进度
    // 注: 根据需求复写
    @Override
    protected void onProgressUpdate(Integer... progresses) {

        ...

    }

    // 方法4: onPostExecute ()
    // 作用: 接收线程任务执行结果、将执行结果显示到UI组件
    // 注: 必须复写, 从而自定义UI操作
    @Override
    protected void onPostExecute(String result) {

        ...// UI操作

    }

    // 方法5: onCancelled()
    // 作用: 将异步任务设置为: 取消状态
    @Override
    protected void onCancelled() {

        ...

    }
}

/**
 * 步骤2: 创建AsyncTask子类的实例对象 (即 任务实例)
 * 注: AsyncTask子类的实例必须在UI线程中创建
 */
MyTask mTask = new MyTask();

/**
 * 步骤3: 手动调用execute(Params... params) 从而执行异步线程任务
 * 注:
 *     a. 必须在UI线程中调用
 *     b. 同一个AsyncTask实例对象只能执行1次, 若执行第2次将会抛出异常
 *     c. 执行任务中, 系统会自动调用AsyncTask的一系列方法: onPreExecute() 、
doInBackground()、onProgressUpdate() 、 onPostExecute()
 *     d. 不能手动调用上述方法
 */
mTask.execute();

```

6. 实例讲解

6.1 实例说明

1. 点击按钮 则 开启线程执行线程任务
2. 显示后台加载进度
3. 加载完毕后更新UI组件
4. 期间若点击取消按钮，则取消加载



7. 使用时的注意点

在使用 AsyncTask 时有一些问题需要注意的：

7.1 关于 生命周期

- 结论
AsyncTask 不与任何组件绑定生命周期
- 使用建议
在 Activity 或 Fragment 中使用 AsyncTask 时，最好在 Activity 或 Fragment 的 `onDestroy()` 调用 `cancel(boolean)`；

7.2 关于 内存泄漏

- 结论
若 AsyncTask 被声明为 Activity 的非静态内部类，当 Activity 需销毁时，会因 AsyncTask 保

留对 `Activity` 的引用 而导致 `Activity` 无法被回收，最终引起内存泄露

- 使用建议

`AsyncTask` 应被声明为 `Activity` 的静态内部类

7.3 线程任务执行结果 丢失

- 结论

当 `Activity` 重新创建时（屏幕旋转 / `Activity` 被意外销毁时后恢复），之前运行的

`AsyncTask`（非静态的内部类）持有的之前 `Activity` 引用已无效，故复写的 `onPostExecute()`

将不生效，即无法更新UI操作

- 使用建议

在 `Activity` 恢复时的对应方法 重启 任务线程

源码分析

参考地址: <https://www.jianshu.com/p/37502bbbb25a>