进程和线程

进程生命周期重要性

线程互相交互和线程与UI线程交互

1，指定代码在一个线程中运行。

执行的特定操作的一个或多个Runnable对象有时被称为一个任务。

只能通过UI线程操纵用户界面。

Android单线程模式需遵守以下规则：

1，不要阻塞UI线程。

2，不要在UI线程之外访问Android UI工具包。

工作线程

例如，以下代码演示了一个点击侦听器从单独的线程下载图像并将其显示在 `ImageView` 中：

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

乍看起来，这段代码似乎运行良好，因为它创建了一个新线程来处理网络操作。但是，它违反了单线程模式的第二条规则：不要在 UI 线程之外访问 Android UI 工具包–此示例从工作线程（而不是 UI 线程）修改了 `ImageView`。这可能导致出现不明确、不可预见的行为，但要跟踪此行为困难而又费时。

为解决此问题，Android 提供了几种途径来从其他线程访问 UI 线程。以下列出了几种有用的方法：

- `Activity.runOnUiThread(Runnable)`
- `View.post(Runnable)`
- `View.postDelayed(Runnable, long)`

例如，您可以通过使用 `View.post(Runnable)` 方法修复上述代码：

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

现在，上述实现属于线程安全型：在单独的线程中完成网络操作，而在 UI 线程中操纵 `ImageView`。

但是，随着操作日趋复杂，这类代码也会变得复杂且难以维护。要通过工作线程处理更复杂的交互，可以考虑在工作线程中使用 `Handler` 处理来自 UI 线程的消息。当然，最好的解决方案或许是扩展 `AsyncTask` 类，此类简化了与 UI 进行交互所需执行的工作线程任务。

使用AsyncTask：

允许对用户界面执行异步操作，首先阻塞工作线程中的操作，然后在UI线程中发布结果。

使用方式：

1，创建AsyncTask子类。

2，实现doInBackground方法回调。

3，更新UI，实现onPostExecute()。

4，从UI线程调用execute()方法运行任务。

例如，您可以通过以下方式使用 AsyncTask 来实现上述示例：

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

现在 UI 是安全的，代码也得到简化，因为任务分解成了两部分：一部分应在工作线程内完成，另一部分应在 UI 线程内完成。

下面简要概述了 AsyncTask 的工作方法，但要全面了解如何使用此类，您应阅读 AsyncTask 参考文档：

- 可以使用泛型指定参数类型、进度值和任务最终值
- 方法 doInBackground() 会在工作线程上自动执行
- onPreExecute()、onPostExecute() 和 onProgressUpdate() 均在 UI 线程中调用
- doInBackground() 返回的值将发送到 onPostExecute()
- 您可以随时在 doInBackground() 中调用publishProgress()，以在 UI 线程中执行 onProgressUpdate()
- 您可以随时取消任何线程中的任务

> **注意：**使用工作线程时可能会遇到另一个问题，即：运行时配置变更（例如，用户更改了屏幕方向）导致 Activity 意外重启，这可能会销毁工作线程。要了解如何在这种重启情况下坚持执行任务，以及如何在 Activity 被销毁时正确地取消任务，请参阅书架示例应用的源代码。

线程安全方法

在某些情况下，您实现的方法可能会从多个线程调用，因此编写这些方法时必须确保其满足线程安全的要求。

这一点主要适用于可以远程调用的方法，如绑定服务中的方法。如果对 IBinder 中所实现方法的调用源自运行 IBinder 的同一进程，则该方法在调用方的线程中执行。但是，如果调用源自其他进程，则该方法将在从线程池选择的某个线程中执行（而不是在进程的 UI 线程中执行），线程池由系统在与 IBinder 相同的进程中维护。例如，即使服务的 onBind() 方法将从服务进程的 UI 线程调用，在 onBind() 返回的对象中实现的方法（例如，实现 RPC 方法的子类）仍会从线程池中的线程调用。由于一个服务可以有多个客户端，因此可能会有多个池线程在同一时间使用同一 IBinder 方法。因此，IBinder 方法必须实现为线程安全方法。

同样，内容提供程序也可接收来自其他进程的数据请求。尽管 ContentResolver 和 ContentProvider 类隐藏了如何管理进程间通信的细节，但响应这些请求的 ContentProvider 方法（query()、insert()、delete()、update() 和 getType() 方法）将从内容提供程序所在进程的线程池中调用，而不是从进程的 UI 线程调用。由于这些方法可能会同时从任意数量的线程调用，因此它们也必须实现为线程安全方法。

进程间通信

Android 利用远程过程调用 (RPC) 提供了一种进程间通信 (IPC) 机制，通过这种机制，由 Activity 或其他应用组件调用的方法将（在其他进程中）远程执行，而所有结果将返回给调用方。这就要求把方法调用及其数据分解至操作系统可以识别的程度，并将其从本地进程和地址空间传输至远程进程和地址空间，然后在远程进程中重新组装并执行该调用。然后，返回值将沿相反方向传输回来。 Android 提供了执行这些 IPC 事务所需的全部代码，因此您只需集中精力定义和实现 RPC 编程接口即可。

要执行 IPC，必须使用 bindService() 将应用绑定到服务上。如需了解详细信息，请参阅服务开发者指南。

Android开启一个工作线程的方式

1，继承自Thread。

```
public class MyThread extends Thread {
    public void run(){
    }
}
```

```
newMyThread().start();
```

2，实现RunAble接口。

```
public class MyRunnable implements Runnable{
@Override
  public void run(){
    //TODOAuto-generatedmethodstub
      }
 }
new MyThread().start();
```

子线程与主线程通信：

1、Activity.runOnUiThread(Runnable)

```
mHandle.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                new Thread(new Runnable() {

                    @Override
                    public void run() {
                                    // 耗时操作
                                    loadNetWork();
                        MainActivity.this.runOnUiThread(new Runnable() {
                        @Override
                            public void run() {
                                mTextView.setText(来自网络的文字);
                            }
                        });

                    }
                });

            }
        });
```

2、 View.post(Runnable)

```
mHandle.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                new Thread(new Runnable() {

                    @Override
                    public void run() {
                              // 耗时操作
                              loadNetWork();
                        mTextView.post(new Runnable() {

                            @Override
                            public void run() {
                                mTextView.setText(来自网络的文字);
                            }
                        });

                    }
                });



            }
        });
```

3、View.postDelayed(Runnable,long)

```
mHandle.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                new Thread(new Runnable() {

                    @Override
                    public void run() {
                                // 耗时操作
                                loadNetWork();
                        mTextView.postDelayed(new Runnable() {

                            @Override
                            public void run() {
                                mTextView.setText(来自网络的文字);
                            }
                        }, 10);

                    }
                });



            }
        });
```

4、Handler(子线程调用Handler的
handle.sendMessage(msg);

```
Handler handle = new Handler() {

        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            mTextView.setText(来自网络的文字);
        }

    };

class MyThread extends Thread {

        @Override
        public void run() {
             // 耗时操作
            loadNetWork();

            Message msg = new Message();
            handle.sendMessage(msg);
            super.run();
        }


    }
```

5、AsyncTask

主线程调用：

```
aTask ak = new aTask();
ak.execute();
```

AsyncTask

```
private class aTask extends AsyncTask {

    //后台线程执行时
    @Override
    protected Object doInBackground(Object... params) {
        // 耗时操作
        return loadNetWork();
    }
    //后台线程执行结束后的操作，其中参数result为doInBackground返回的结果
    @Override
    protected void onPostExecute(Object result) {
        super.onPostExecute(result);
        mTextView.setText(result);
    }
    }
```