# **[Android的单线程模型](http://www.cnblogs.com/nio-nio/archive/2012/07/23/2604900.html)**

当一个程序第一次启动时，Android会同时启动一个对应的主线程（Main Thread），主线程主要负责处理与UI相关的事件，如：用户的按键事件，用户接触屏幕的事件以及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以主线程通常又被叫做UI线程。

在开发Android 应用时必须遵守单线程模型的原则： Android UI操作并不是线程安全的并且这些操作必须在UI线程中执行。

如果在非UI线程中直接操作UI线程，会抛出android.view.ViewRoot$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views，这与普通的java程序不同。

由于UI线程负责事件的监听和绘图，因此，必须保证UI线程能够随时响应用户的需求，UI线程里的操作应该向中断事件那样短小，费时的操作（如网络连接）需要另开线程，否则，如果UI线程超过5s没有响应用户请求，会弹出对话框提醒用户终止应用程序。

如果在新开的线程中需要对UI进行设定，就可能违反单线程模型，因此android采用一种复杂的Message Queue机制保证线程间通信。

# **Message Queue：**

Message Queue是一个消息队列，用来存放通过Handler发布的消息。Android在第一次启动程序时会默认会为UI thread创建一个关联的消息队列，可以通过Looper.myQueue()得到当前线程的消息队列，用来管理程序的一些上层组件，activities，broadcast receivers 等等。你可以在自己的子线程中创建Handler与UI thread通讯。

通过Handler你可以发布或者处理一个消息或者是一个Runnable的实例。每个Handler都会与唯一的一个线程以及该线程的消息队列管理。

Looper扮演着一个Handler和消息队列之间通讯桥梁的角色。程序组件首先通过Handler把消息传递给Looper，Looper把消息放入队列。Looper也把消息队列里的消息广播给所有的Handler，Handler接受到消息后调用handleMessage进行处理。

实例如下：

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

editText = (EditText) findViewById(R.id.weather\_city\_edit);

Button button = (Button) findViewById(R.id.goQuery);

button.setOnClickListener(this);

Looper looper = Looper.myLooper(); //得到当前线程的Looper实例，由于当前线程是UI线程也可以通过Looper.getMainLooper()得到 messageHandler = new MessageHandler(looper); //此处甚至可以不需要设置Looper，因为 Handler默认就使用当前线程的Looper

}

public void onClick(View v) {

new Thread() {

public void run() {

Message message = Message.obtain();

message.obj = "abc";

messageHandler.sendMessage(message); //发送消息

}

}.start();

}

Handler messageHandler = new Handler {

public MessageHandler(Looper looper) {

super(looper);

}

public void handleMessage(Message msg) {

setTitle((String) msg.obj);

}

}

对于这个实例，当这个activity执行玩oncreate,onstart,onresume后，就监听UI的各种事件和消息。

当我们点击一个按钮后，启动一个线程，线程执行结束后，通过handler发送一个消息，由于这个handler属于UI线程，因此这个消息也发送给UI线程，然后UI线程又把这个消息给handler处理，而这个handler是UI线程创造的，他可以访问UI组件，因此，就更新了页面。

由于通过handler需要自己管理线程类，如果业务稍微复杂，代码看起来就比较混乱，因此android提供了AsyncTask类来解决此问题。

# **AsyncTask:**

首先继承一下此类，实现以下若干方法，

onPreExecute(), 该方法将在执行实际的后台操作前被**UI thread**调用。可以在该方法中做一些准备工作，如在界面上显示一个进度条。

doInBackground(Params...), 将在onPreExecute 方法执行后马上执行，该方法运行在**后台**线程中。这里将主要负责执行那些很耗时的后台计算工作。

可以调用publishProgress方法来更新实时的任务进度。该方法是抽象方法，子类必须实现。

onProgressUpdate(Progress...),在publishProgress方法被调用后，UI thread将调用这个方法从而在界面上展示任务的进展情况，例如通过一个进度条进行展示。

onPostExecute(Result), 在doInBackground 执行完成后，onPostExecute 方法将被**UI thread**调用，后台的计算结果将通过该方法传递到UI thread.

使用时需要遵循以下规则：

**1)Task的实例必须在UI thread中创建**

**2)execute方法必须在UI thread中调用**

**3)不要手动的调用这些方法，只调用execute即可**

**4)该task只能被执行一次，否则多次调用时将会出现异常**

示例如下：

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

editText = (EditText) findViewById(R.id.weather\_city\_edit);

Button button = (Button) findViewById(R.id.goQuery);

button.setOnClickListener(this);

}

public void onClick(View v) {

new GetWeatherTask().execute(“aaa”);

}

class GetWeatherTask extends AsyncTask<String, Integer, String> {

protected String doInBackground(String... params) {

return getWetherByCity(params[0]);

}

protected void onPostExecute(String result) {

setTitle(result);

}

}