华清远见 FARSIGHT 嵌入式培训专家

嵌入式学院 华清远见旗下品牌

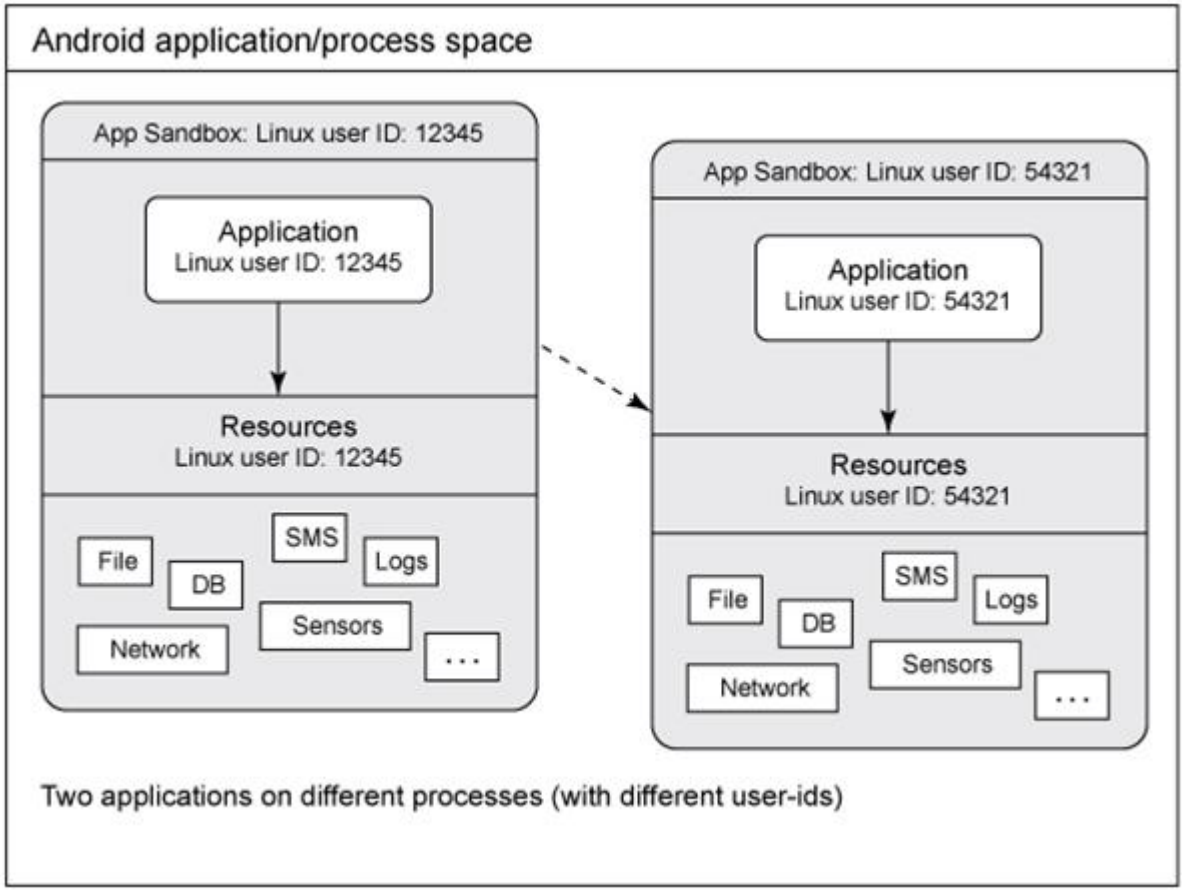
ANDROID进程和线程

版权声明

Android进程和线程

- 在安装Android应用程序的时候，Android会为每个程序分配一个Linux用户ID，并设置相应的权限，这样其它应用程序就不能访问此应用程序所拥有的数据和资源了。
- 下图中，两个 Android 应用程序，各自运行在其自己的基本沙箱或进程上。它们有不同的Linux user ID

Android进程和线程



Android进程和线程

- 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。

Android进程和线程

- 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
- 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线程里的操作应该像中断事件那样短小，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。

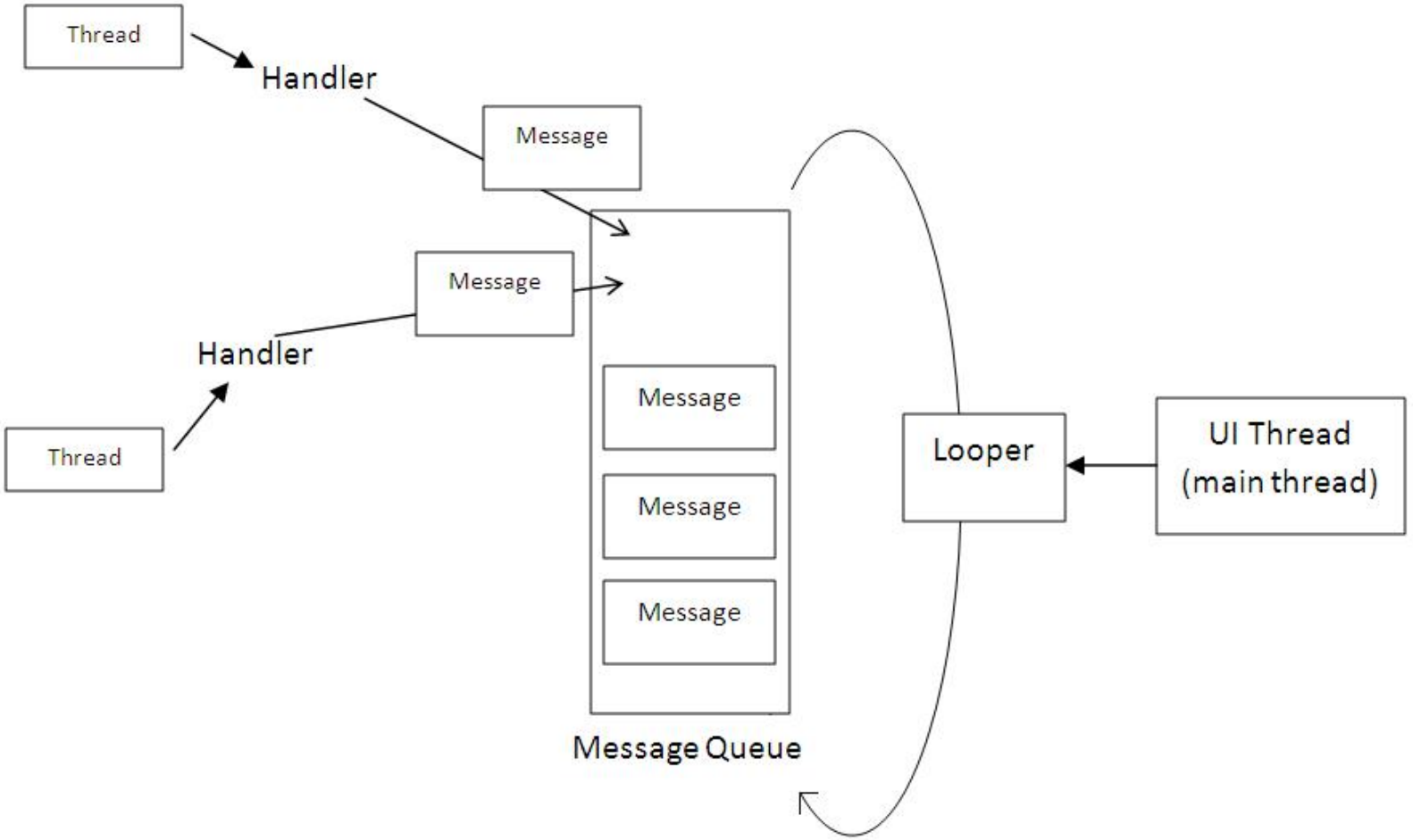
Android进程与线程

- 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，如Activities、BroadcastReceivers 等等。你可以在自己的子线程中创建Handler与UI thread通讯。

Handler消息传递机制

- Android通过Looper、Handler来实现消息循环机制，Android消息循环是针对线程的（每个线程都可以有自己的消息队列和消息循环）。
- Android系统中，Looper负责管理线程的消息队列和消息循环。我们可以通过Loop.myLooper()得到当前线程的Looper对象，通过Loop.getMainLooper()可以获得当前进程的主线程的Looper对象。
- 一个线程可以存在（当然也可以不存在）一个消息队列和一个消息循环（Looper）。
- 构造Handler的时候可以指定一个Looper对象，如果不指定则利用当前线程的Looper创建。

Handler消息传递机制



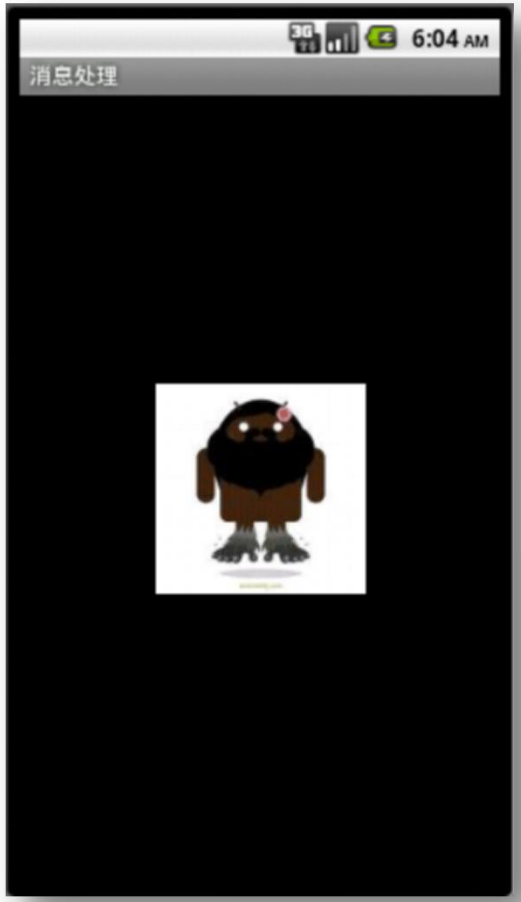
Handler消息传递机制

- 在Android平台中，新启动的线程是无法访问Activity中的Widget的，当然也不能将运行状态传递出来，这就需要有Handler机制了
- 另外，正如前面所说，我们希望能在一个子线程中获取数据并将其显示到界面上，如果直接在子线程中设置界面，则将会报错（android.view.ViewRoot$CalledFromWrongThreadException），此时也需要使用Handler来处理

Handler

方法	描述
void handleMessage(Message message)	通过这个方法接收消息
boolean sendEmptyMessage(int what)	发送只有一个what值的消息
boolean sendMessage(Message message)	发送消息到Handler，Handler即可用handleMessage处理
boolean hasMessage(int what)	判断是否有what值的消息
boolean post(Runnable r)	将一个线程添加到消息队列

Handler – Example 1



Handler – Example 2

AsyncTask

- Handler和AsyncTask，都是为了不阻塞主线程（UI线程），且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。
- 要使用AsyncTask，需要自己编写一个类，继承AsyncTask类，并且实现其中的方法
- AsyncTask定义了三种泛型类型 Params，Progress和Result，即AsyncTask<Params,Progress,Result>
  - Params 启动任务执行的输入参数，比如HTTP请求的URL。
  - Progress 后台任务执行的百分比。
  - Result 后台执行任务最终返回的结果，比如String。

AsyncTask

- AsyncTask的执行分为四个步骤，每一步都对应一个回调方法，开发者需要实现至少一个方法（doInBackground(Params...)）。这些方法都是回调方法，在任务的执行过程中，这些方法被自动调用：
  - 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。该方法是抽象方法，子类必须实现。在这个方法中可以调用 publishProgress()方法来更新实时的任务进度。

AsyncTask

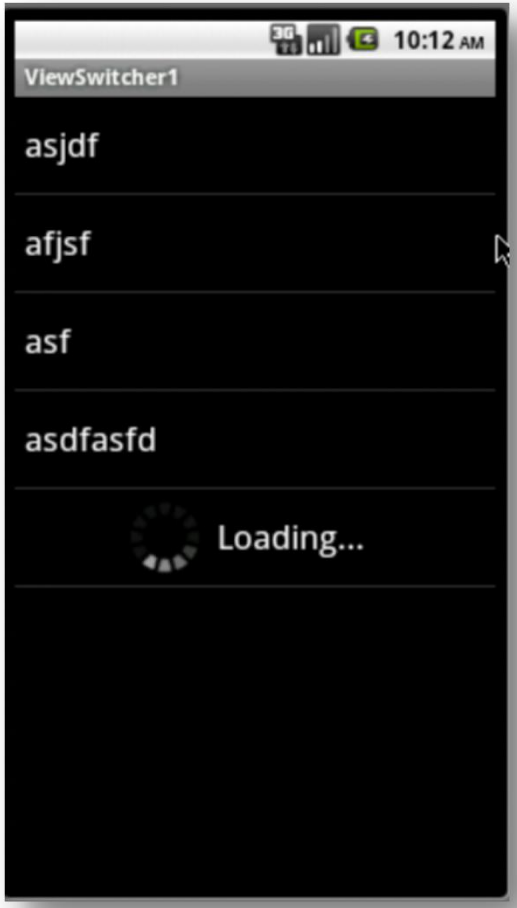
- onProgressUpdate(Progress...)：在publishProgress方法被调用后，UI thread将调用这个方法从而在界面上展示任务的进展情况，例如通过一个进度条进行展示。
- onPostExecute(Result)： 在doInBackground 执行完成后，onPostExecute 方法将被UI thread调用，后台的计算结果将通过该方法传递到UI thread.

AsyncTask

- 使用AsyncTask类，以下是几条必须遵守的准则：
  - Task的实例必须在UI thread中创建
  - execute方法必须在UI thread中调用
  - 不要手动的调用onPreExecute(), onPostExecute(Result)，doInBackground(Params...), onProgressUpdate(Progress...)这几个方法
  - 该task只能被执行一次，否则多次调用时将会出现异常

AsyncTask


10:12 AM
ViewSwitcher1
asjdf
afjsf
asf
asdfasfd
更多...


10:12 AM
ViewSwitcher1
asjdf
afjsf
asf
asdfasfd
Loading...

AsyncTask

- 一开始要在res文件夹底下新增两个XML文件，接着将一开始新增的两个XML文件，作为两个View，并使用ViewSwitcher的方法去做两个View之间的切换。
- 当按下“更多” 的按钮时，ViewSwitcher就会切换到另外一个View，当后台任务处理完成后，才会切换回原本的View。

AsyncTask

布局文件(res/layout/button.xml)：

```
<?xml version="1.0" encoding="utf-8"?>
<!--使用ViewSwitcher切换的第一个View-->
<Button
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/btn_loadmorecontacts"
    android:text="更多..."
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"

android:textAppearance="?android:attr/textAppearanceLarge"

android:minHeight="?android:attr/listPreferredItemHeight"
    android:textColor="#FFFFFF"

android:background="@android:drawable/list_selector_backgro
und"
    android:clickable="true"
    android:onClick="onClick"
/>
```

AsyncTask

布局文件(res/layout/progress.xml)：

```
<?xml version="1.0" encoding="utf-8"?>
<!--使用ViewSwitcher切换的第二个View-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal" android:id="@+id/relativeLayout1"
    android:minHeight="?android:attr/listPreferredItemHeight">
    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
    />
    <TextView
        android:text="Loading…"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_toRightOf="@+id/progressBar"
        android:layout_centerVertical="true"
        android:gravity="center"
        android:padding="10dip"
        android:textColor="#FFFFFF"
    />
</RelativeLayout>
```

AsyncTask

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
switcher = new ViewSwitcher(this);

button = (Button) View.inflate(this, R.layout.button, null);
button.setOnClickListener(this);
progress = (RelativeLayout) View.inflate(this,
R.layout.progress, null);

/* 将button与progressbar加入switcher中 */
switcher.addView(button);
switcher.addView(progress);

/* 取得ListView并将switcher加入 */
getListView().addFooterView(switcher);

/* 设定ListAdapter，其中第二个参数可选择样式 */
setListAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, ITEMS));
}
```

AsyncTask

```
private class GetMoreItemsTask extends
AsyncTask<Object,Object,Object> {
@Override
protected Object doInBackground(Object... params) {
/* 此处可编写后台任务的程序代码 */
try {
/* 线程调用sleep方法 ，实际项目中应该是执行耗时操作*/
Thread.sleep(3000);
} catch (InterruptedException e) {
e.printStackTrace();
}
return null;
}

@Override
protected void onPostExecute(Object result) {
switcher.showPrevious();
}
}
```

Activity.runOnUiThread(Runnable)

- 另外，在Activity中也定义了一个runOnUiThread(Runnable)方法，用于处理UI线程的问题：

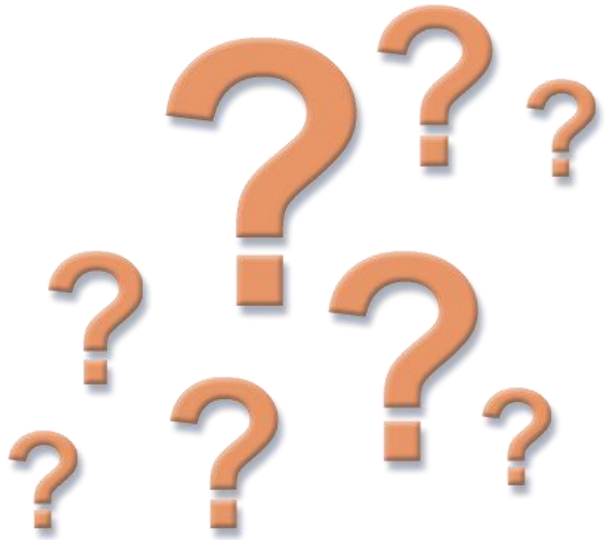
```
public void onClick( View v ) {
        new Thread( new Runnable() {
                public void run() {
                    // 耗时操作
                    Activity.runOnUiThread( new Runnable()
                    {
                            myText.setText( 来自网络的信息);
                    });
        }
          }).start();
}
```

View.post()和View.postDelayed()

- 在View中，也定义了View.post(Runnable)/ View.postDelayed(Runnable, long)两个方法来处理View的线程问题
  - postDelayed可以指定任务的延迟时间（毫秒）

```
public void onClick( View v ) {
        new Thread( new Runnable() {
                public void run() {
                    // 耗时操作
                    myText.post( new Runnable() {
                    myText.setText( ...);
                  });
                }
        }).start();
    }
```

Q&A



谢谢！