Android进程与线程基本知识

本文介绍Android平台中进程与线程的基本知识。

很早的时候就想介绍一下Android中的进程和线程，但由于其他的事情一直给耽搁了，直到现在才能和大家一起分享下。

1.Android进程基本知识：

我们先来了解下Android中的进程基本知识。

当一个程序第一次启动的时候，Android会启动一个LINUX进程和一个主线程。默认的情况下，所有该程序的组件都将在该进程和线程中运行。 同时，Android会为每个应用程序分配一个单独的LINUX用户。Android会尽量保留一个正在运行进程，只在内存资源出现不足时，Android会尝试停止一些进程从而释放足够的资源给其他新的进程使用， 也能保证用户正在访问的当前进程有足够的资源去及时地响应用户的事件。

我们可以将一些组件运行在其他进程中，并且可以为任意的进程添加线程。组件运行在哪个进程中是在manifest文件里设置的，其中<Activity>，<Service>，<receiver>和<provider>都有一个process属性来指定该组件运行在哪个进程之中。我们可以设置这个属性，使得每个组件运行在它们自己的进程中，或是几个组件共同享用一个进程，或是不共同享用。<application>元素也有一个process属性，用来指定所有的组件的默认属性。

Android中的所有组件都在指定的进程中的主线程中实例化的，对组件的系统调用也是由主线程发出的。每个实例不会建立新的线程。对系统调用进行响应的方法——例如负责执行用户动作的View.onKeyDown()和组件的生命周期函数——都是运行在这个主线程中的。这意味着当系统调用这个组件时，这个组件不能长时间的阻塞主线程。例如进行网络操作时或是更新UI时，如果运行时间较长，就不能直接在主线程中运行，因为这样会阻塞这个进程中其他的组件，我们可以将这样的组件分配到新建的线程中或是其他的线程中运行。

Android会根据进程中运行的组件类别以及组件的状态来判断该进程的重要性，Android会首先停止那些不重要的进程。按照重要性从高到低一共有五个级别：

前台进程

前台进程是用户当前正在使用的进程。只有一些前台进程可以在任何时候都存在。他们是最后一个被结束的，当内存低到根本连他们都不能运行的时候。一般来说， 在这种情况下，设备会进行内存调度，中止一些前台进程来保持对用户交互的响应。

如果有以下的情形的那么就是前台进程：

这个进程运行着一个正在和用户交互的Activity（这个Activity的onResume()方法被调用）。

这个进程里有绑定到当前正在和用户交互的确Activity的一个service。

这个进程里有一个service对象，这个service对象正在执行一个它的生命周期的回调函数(onCreate(), onStart(), onDestroy())

这个进程里有一个正在的onReceive()方法的BroadCastReiver对象。

可见进程

可见进程不包含前台的组件但是会在屏幕上显示一个可见的进程是的重要程度很高，除非前台进程需要获取它的资源，不然不会被中止。

如果有如下的一种情形就是可见进程：

这个进程中含有一个不位于前台的Activity，但是仍然对用户是可见的(这个Activity的onPause()方法被调用)，这是很可能发生的，例如，如果前台Activity是一个对话框的话，就会允许在它后面看到前一个Activity。

这个进程里有一个绑定到一个可见的Activity的Service。

服务进程

运行着一个通过startService() 方法启动的service，这个service不属于上面提到的2种更高重要性的。service所在的进程虽然对用户不是直接可见的，但是他们执行了用户非常关注的任务（比如播放mp3，从网络下载数据）。只要前台进程和可见进程有足够的内存，系统不会回收他们。

后台进程

运行着一个对用户不可见的activity（调用过 onStop() 方法).这些进程对用户体验没有直接的影响，可以在服务进程、可见进程、前台进 程需要内存的时候回收。通常，系统中会有很多不可见进程在运行，他们被保存在LRU (least recently used) 列表中，以便内存不足的时候被第一时间回收。如果一个activity正 确的执行了它的生命周期，关闭这个进程对于用户体验没有太大的影响。

空进程

未运行任何程序组件。运行这些进程的唯一原因是作为一个缓存，缩短下次程序需要重新使用的启动时间。系统经常中止这些进程，这样可以调节程序缓存和系统缓存的平衡。

Android 对进程的重要性评级的时候，选取它最高的级别。例如，如果一个进程含有一个service和一个可视activity，进程将被归入一个可视进程而不是service进程。

另外，当被另外的一个进程依赖的时候，某个进程的级别可能会增高。一个为其他进程服务的进程永远不会比被服务的进程重要级低。因为服务进程比后台activity进程重要级高，因此一个要进行耗时工作的activity最好启动一个service来做这个工作，而不是开启一个子进程－－特别是这个操作需要的时间比activity存在的时间还要长的时候。例如，在后台播放音乐，向网上上传摄像头拍到的图片，使用service可以使进程最少获取到“服务进程”级别的重要级，而不用考虑activity目前是什么状态。broadcast receivers做费时的工作的时候，也应该启用一个服务而不是开一个线程。

2. 单线程模型

线程在代码是使用标准的java Thread对象来建立，那么在Android系统中提供了一系列方便的类来管理线程——Looper用来在一个线程中执行消息循环，Handler用来处理消息，HandlerThread创建带有消息循环的线程。具体可以看下面的详细介绍。

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。

2.1 子线程更新UI Android的UI是单线程(Single-threaded)的。

为了避免拖住GUI，一些较费时的对象应该交给独立的线程去执行。如果幕后的线程来执行UI对象，Android就会发出错误讯息 CalledFromWrongThreadException。以后遇到这样的异常抛出时就要知道怎么回事了！

2.2 Message Queue

在单线程模型下，为了解决类似的问题，Android设计了一个Message Queue(消息队列)， 线程间可以通过该Message Queue并结合Handler和Looper组件进行信息交换。下面将对它们进行分别介绍：

2.2.1. Message 消息

理解为线程间交流的信息，处理数据后台线程需要更新UI，则发送Message内含一些数据给UI线程。

2.2.2. Handler 处理者

是Message的主要处理者，负责Message的发送，Message内容的执行处理。后台线程就是通过传进来的Handler对象引用来sendMessage(Message)。而使用Handler，需要implement 该类的 handleMessage(Message) 方法，它是处理这些Message的操作内容，例如Update UI。通常需要子类化Handler来实现handleMessage方法。

2.2.3. Message Queue 消息队列

用来存放通过Handler发布的消息，按照先进先出执行。 每个message queue都会有一个对应的Handler。Handler会向message queue通过两种方法发送消息：sendMessage或post。这两种消息都会插在message queue队尾并按先进先出执行。但通过这两种方法发送的消息执行的方式略有不同：通过sendMessage发送的是一个message对象,会被Handler的handleMessage()函数处理；而通过post方法发送的是一个runnable对象，则会自己执行。

2.2.4. Looper Looper是每条线程里的Message Queue的管家。

Android没有Global的Message Queue，而Android会自动替主线程(UI线程)建立Message Queue，但在子线程里并没有建立Message Queue。所以调用Looper.getMainLooper()得到的主线程的Looper不为NULL，但调用Looper.myLooper()得到当前线程的Looper就有可能为NULL。

对于子线程使用Looper，API Doc提供了正确的使用方法：

```
package com.test;
import  android.os.Handler;
import  android.os.Looper;
import  android.os.Message;

class  LooperThread  extends  Thread  {
        public  Handler  mHandler;
```

```
        public  void  run()  {
                Looper.prepare();  //创建本线程的Looper并创建一个MessageQueue
                mHandler  =  new  Handler()  {
                        public  void  handleMessage(Message  msg)  {

                        //  process  incoming  messages  here
                        }
                };

                Looper.loop();  //开始运行Looper,监听Message  Queue
        }
}
```

这个Message机制的大概流程：

1. 在Looper.loop()方法运行开始后，循环地按照接收顺序取出Message Queue里面的非NULL的Message。

2. 一开始Message Queue里面的Message都是NULL的。当Handler.sendMessage(Message)到Message Queue，该函数里面设置了那个Message对象的target属性是当前的Handler对象。随后Looper取出了那个Message，则调用该Message的target指向的Hander的dispatchMessage函数对Message进行处理。

在dispatchMessage方法里，如何处理Message则由用户指定，三个判断，优先级从高到低：

1) Message里面的Callback，一个实现了Runnable接口的对象，其中run函数做处理工作；

2) Handler里面的mCallback指向的一个实现了Callback接口的对象，由其handleMessage进行处理；

3) 处理消息Handler对象对应的类继承并实现了其中handleMessage函数，通过这个实现的handleMessage函数处理消息。

由此可见，我们实现的handleMessage方法是优先级最低的！

3. Handler处理完该Message (update UI) 后，Looper则设置该Message为NULL，以便回收！

3.Android另外提供了一个工具类：AsyncTask。

它使得UI thread的使用变得异常简单。它使创建需要与用户界面交互的长时间运行的任务变得更简单，不需要借助线程和Handler即可实现。

1) 子类化AsyncTask

2) 实现AsyncTask中定义的下面一个或几个方法

onPreExecute() 开始执行前的准备工作；

doInBackground(Params...) 开始执行后台处理，可以调用publishProgress方法来更新实时的任务进度；

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) 执行完成后的操作，传送结果给UI 线程。

这4个方法都不能手动调用。而且除了doInBackground(Params...)方法，其余3个方法都是被UI线程所调用的，所以要求：

1) AsyncTask的实例必须在UI thread中创建；

2) AsyncTask.execute方法必须在UI thread中调用；

同时要注意：该task只能被执行一次，否则多次调用时将会出现异常。

在使用过程中，发现AsyncTask的构造函数的参数设置需要看明白：

AsyncTask<Params, Progress, Result> Params对应doInBackground(Params...)的参数类型。

而new AsyncTask().execute(Params... params)，就是传进来的Params数据，你可以execute(data)来传送一个数据，或者execute(data1, data2, data3)这样多个数据。

Progress对应onProgressUpdate(Progress...)的参数类型；

Result对应onPostExecute(Result)的参数类型。 当以上的参数类型都不需要指明某个时，则使用Void，注意不是void。不明白的可以参考上面的例子，或者API Doc里面的例子。

下面是关于AsyncTask的使用示例：

```
((Button)  findViewById(R.id.load_AsyncTask)).setOnClickListener(new  View.OnClickListener(){

        @Override
        public  void  onClick(View  view)  {
                data  =  null;
                data  =  new  ArrayList<String>();

                adapter  =  null;

                //显示ProgressDialog放到AsyncTask.onPreExecute()里
                //showDialog(PROGRESS_DIALOG);
                new  ProgressTask().execute(data);
        }
});

private  class  ProgressTask  extends  AsyncTask<ArrayList<String>,  Void,  Integer>  {

/*  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。*/
@Override
protected  void  onPreExecute()  {
        //  先显示ProgressDialog
        showDialog(PROGRESS_DIALOG);
}

/*  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。  */
@Override
protected  Integer  doInBackground(ArrayList<String>...  datas)  {
        ArrayList<String>  data  =  datas[0];
        for  (int  i=0;  i<8;  i++)  {
                data.add("ListItem");
        }
        return  STATE_FINISH;
}

/*  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.
  */
@Override
protected  void  onPostExecute(Integer  result)  {
        int  state  =  result.intValue();
        switch(state){
        case  STATE_FINISH:
```

```
                dismissDialog(PROGRESS_DIALOG);
                Toast.makeText(getApplicationContext(),
                               "加载完成!",
                               Toast.LENGTH_LONG)
                     .show();

                adapter  =  new  ArrayAdapter<String>(getApplicationContext(),
                               android.R.layout.simple_list_item_1,
                               data  );

                setListAdapter(adapter);

                break;

        case  STATE_ERROR:
              dismissDialog(PROGRESS_DIALOG);
              Toast.makeText(getApplicationContext(),
                             "处理过程发生错误!",
                             Toast.LENGTH_LONG)
                   .show();

              adapter  =  new  ArrayAdapter<String>(getApplicationContext(),
                             android.R.layout.simple_list_item_1,
                             data  );

                   setListAdapter(adapter);

                   break;

      default:

      }
}
```

以上是从网络获取数据，加载到ListView中示例。

4.Android中如何结束进程

4.1 Android 结束进程，关闭程序的方法 即采用下面这个类

Void android.app.ActivityManager.restartPackage(String packageName)

public void restartPackage (String packageName)

Since: API Level 3

Have the system perform a force stop of everything associated with the given application package. All processes that share its uid will be killed, all services it has running stopped, all activities removed, etc. In addition, a ACTION_PACKAGE_RESTARTED broadcast will be sent, so that any of its registered alarms can be stopped, notifications removed, etc. You must hold the permission RESTART_PACKAGES to be able to call this method. Parameters packageName The name of the package to be stopped.

使用这个类的具体源代码

```
final  ActivityManager  am  =  (ActivityManager)getSystemService(Context.ACTIVITY_SERVICE);
am.restartPackage(getPackageName());
```

不要忘记了在配置文件中设置权限：

```
<uses-permission  android:name="android.permission.RESTART_PACKAGES"></uses-permission>
```

结束进程还有

4.2 android.os.Process.killProcess(pid)

只能终止本程序的进程，无法终止其它的

public static final void killProcess (int pid)

Kill the process with the given PID. Note that, though this API allows us to request to kill any process based on its PID, the kernel will still impose standard restrictions on which PIDs you are actually able to kill. Typically this means only the process running the caller's packages/application and any additional processes created by that app; packages sharing a common UID will also be able to kill each other's processes.

具体代码如下：

Process.killProcess(Process.myPid());

```
Process.killProcess(Process.myPid());
```

public void finish ()

Call this when your activity is done and should be closed. The ActivityResult is propagated back to whoever launched you via onActivityResult().

这是结束当前activity的方法。 要主动的结束一个活动Activity，这里需要注意finish是结束掉一个Activity，而不是一个进程。这个方法最后会调用Activity的生命周期函数onDestroy方法，结束当前的Activity，从任务栈中弹出当前的Activity，激活下一个Activity。当然其他的finish系列方法，我们不在这里做详细讨论。

4.3 System.exit(int code)

例如： System.exit(0);

该方法只能用于结束当前进程自身，在程序遇到异常，无法正常执行时，可以 通过这个方法强制退出。

需要注意的是： android.os.Process.killProcess(pid) 和 System.exit(int code)会导致进程非正常退出，进程退出时不会去执行Activity的onPause、onStop和onDestroy方法，那么进程很有可能错过了保存数据的机会。因此，这两个方法最好使用在出现异常的时候！大家需要注意其使用方法。

4.4 在android2.2版本之后则不能再使用restartPackage()方法，而应该使用killBackgroundProcesses()方法

manager.killBackgroundProcesses(getPackageName());

使用示例代码如下：

```
ActivityManager  manager  =  (ActivityManager)getSystemService(ACTIVITY_SERVICE);
manager.killBackgroundProcesses(getPackageName());
```

```
//需要在xml中加入权限声明
<uses-permission  android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
```

另外，在android2.2以后，如果服务在ondestroy里加上了start自己，用kill backgroudprocess通常无法结束自己。

4.5还有一种最新发现的方法，利用反射调用forceStopPackage来结束进程

```
Method forceStopPackage = am.getClass().getDeclaredMethod("forceStopPackage", String.class);
forceStopPackage.setAccessible(true);
forceStopPackage.invoke(am, yourpkgname);
```

配置文件中需要添加定义：

```
android:sharedUserId="android.uid.system"
```

另外需要再在配置文件添加权限：

```
<uses-permission android:name="android.permission.FORCE_STOP_PACKAGES"></uses-permission>
```

并且采用系统platform签名 因为需要用FORCE_STOP_PACKAGES权限，该权限只赋予系统签名级程序 即可实现强制停止指定程序

4.6 还有一种方法 利用linux的kill -9命令

4.7 退出到主屏幕（是对当前进程的一种处理）

这个方法，也是退出当前进程的一个方法。如果我们在进程中创建了很多的Activity， 但是又不想关闭时去退出不在任务栈顶的Activity ，那么就可以直接使用这个方法了。

功能：当按下返回键时，就返回到主屏幕，并带有参数 FLAG_ACTIVITY_CLEAR_TOP ， 会清理掉当前的活动。

以下是按下返回键同时不重复时，返回到主屏幕的示例：

```
package com.test.android;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.KeyEvent;

/**
 * Andriod按返回键返回到主屏幕示例
 * @Description: Andriod按返回键返回到主屏幕示例

 * @FileName: MyTestProjectActivity.java

 * @Package com.test.android

 * @Author Hanyonglu

 * @Date 2012-4-11 上午11:57:31

 * @Version V1.0
 */
public class MyTestProjectActivity extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }
```

```
        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
                // event.getRepeatCount(): 按下返回键，同时没有重复
                if (keyCode == KeyEvent.KEYCODE_BACK && event.getRepeatCount() == 0) {
                        Intent home = new Intent(Intent.ACTION_MAIN);
                        home.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        home.addCategory(Intent.CATEGORY_HOME);
                        startActivity(home);
                }

                return super.onKeyDown(keyCode, event);
        }
}
```

这种方法其实并没有将进程完全地退出，只是将该程序进入到了后台运行，以便下次更快的启动，所以想要程序进入到后台运行可以考虑采用这种方式。关于这点呢，大家了解就可以了。

以上就是关于Android中进程和线程的基本知识，个人觉得理解这些知识点很重要，虽然它不能让我们在立即能够让我们享受到做出一个产品的成就感，但是可以增强我们的内力，让我们对Android的一些运行机制掌握地更加清楚。

最后，希望转载的朋友能够尊重作者的劳动成果，加上转载地址：http://www.cnblogs.com/hanyonglu/archive/2012/04/12/2443262.html 谢谢。

完毕。^_^