**[Android应用程序的Activity启动过程简要介绍和学习计划](http://blog.csdn.net/luoshengyang/article/details/6685853" \o "Android应用程序的Activity启动过程简要介绍和学习计划)**

分类： [Android](http://blog.csdn.net/Luoshengyang/article/category/838604) 2011-08-14 15:46 1047人阅读 [评论](http://blog.csdn.net/luoshengyang/article/details/6685853#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/luoshengyang/article/details/6685853#report)

        在Android系统中，Activity和Service是应用程序的核心组件，它们以松藕合的方式组合在一起构成了一个完整的应用程序，这得益于应用程序框架层提供了一套完整的机制来协助应用程序启动这些Activity和Service，以及提供Binder机制帮助它们相互间进行通信。在前面的文章[Android进程间通信（IPC）机制Binder简要介绍和学习计划](http://blog.csdn.net/luoshengyang/article/details/6618363)和[Android系统在新进程中启动自定义服务过程（startService）的原理分析](http://blog.csdn.net/luoshengyang/article/details/6677029)中，我们已经系统地介绍了Binder机制和Service的启动过程了，在本文中，简要介绍Activity的启动过程以及后续学习计划。

        在Android系统中，有两种操作会引发Activity的启动，一种用户点击应用程序图标时，Launcher会为我们启动应用程序的主Activity；应用程序的默认Activity启动起来后，它又可以在内部通过调用startActvity接口启动新的Activity，依此类推，每一个Activity都可以在内部启动新的Activity。通过这种连锁反应，按需启动Activity，从而完成应用程序的功能。

        这里，我们通过一个具体的例子来说明如何启动Android应用程序的Activity。Activity的启动方式有两种，一种是显式的，一种是隐式的，隐式启动可以使得Activity之间的藕合性更加松散，因此，这里只关注隐式启动Activity的方法。

        首先在Android源代码工程的packages/experimental目录下创建一个应用程序工程目录Activity。关于如何获得Android源代码工程，请参考[在Ubuntu上下载、编译和安装Android最新源代码](http://blog.csdn.net/luoshengyang/article/details/6559955)一文；关于如何在Android源代码工程中创建应用程序工程，请参考[在Ubuntu上为Android系统内置Java应用程序测试Application Frameworks层的硬件服务](http://blog.csdn.net/luoshengyang/article/details/6580267)一文。这里，工程名称就是Activity了，它定义了一个路径为shy.luo.activity的package，这个例子的源代码主要就是实现在这里了。下面，将会逐一介绍这个package里面的文件。

       应用程序的默认Activity定义在src/shy/luo/activity/MainActivity.java文件中：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. package shy.luo.activity;
3. import shy.luo.activity.R;
5. import android.app.Activity;
6. import android.content.Intent;
7. import android.os.Bundle;
8. import android.util.Log;
9. import android.view.View;
10. import android.view.View.OnClickListener;
11. import android.widget.Button;
13. public class MainActivity extends Activity  implements OnClickListener {
14. private final static String LOG\_TAG = "shy.luo.activity.MainActivity";
16. private Button startButton = null;
18. @Override
19. public void onCreate(Bundle savedInstanceState) {
20. super.onCreate(savedInstanceState);
21. setContentView(R.layout.main);
23. startButton = (Button)findViewById(R.id.button\_start);
24. startButton.setOnClickListener(this);
26. Log.i(LOG\_TAG, "Main Activity Created.");
27. }
29. @Override
30. public void onClick(View v) {
31. if(v.equals(startButton)) {
32. Intent intent = new Intent("shy.luo.activity.subactivity");
33. startActivity(intent);
34. }
35. }
36. }

它的实现很简单，当点击它上面的一个按钮的时候，就会启动另外一个名字为“shy.luo.activity.subactivity”的Actvity。

        名字为“shy.luo.activity.subactivity”的Actvity实现在src/shy/luo/activity/SubActivity.java文件中：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. package shy.luo.activity;
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.util.Log;
6. import android.view.View;
7. import android.view.View.OnClickListener;
8. import android.widget.Button;
10. public class SubActivity extends Activity implements OnClickListener {
11. private final static String LOG\_TAG = "shy.luo.activity.SubActivity";
13. private Button finishButton = null;
15. @Override
16. public void onCreate(Bundle savedInstanceState) {
17. super.onCreate(savedInstanceState);
18. setContentView(R.layout.sub);
20. finishButton = (Button)findViewById(R.id.button\_finish);
21. finishButton.setOnClickListener(this);
23. Log.i(LOG\_TAG, "Sub Activity Created.");
24. }
26. @Override
27. public void onClick(View v) {
28. if(v.equals(finishButton)) {
29. finish();
30. }
31. }
32. }

它的实现也很简单，当点击上面的一个铵钮的时候，就结束自己，回到前面一个Activity中去。

        这里我们可以看到，Android应用程序架构中非常核心的一点：MainActivity不需要知道SubActivity的存在，即它不直接拥有SubActivity的接口，但是它可以通过一个字符串来告诉应用程序框架层，它要启动的Activity的名称是什么，其它的事情就交给应用程序框架层来做，当然，应用程序框架层会根据这个字符串来找到其对应的Activity，然后把它启动起来。这样，就使得Android应用程序中的Activity藕合性很松散，从而使得Android应用程序的模块性程度很高，并且有利于以后程序的维护和更新，对于大型的客户端软件来说，这一点是非常重要的。

        当然，应用程序框架能够根据名字来找到相应的Activity，是需要应用程序本身来配合的，这就是要通过应用程序的配置文件AndroidManifest.xml来实现了：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<manifest** xmlns:android="http://schemas.android.com/apk/res/android"
3. package="shy.luo.activity"
4. android:versionCode="1"
5. android:versionName="1.0"**>**
6. **<application** android:icon="@drawable/icon" android:label="@string/app\_name"**>**
7. **<activity** android:name=".MainActivity"
8. android:label="@string/app\_name"**>**
9. **<intent-filter>**
10. **<action** android:name="android.intent.action.MAIN" **/>**
11. **<category** android:name="android.intent.category.LAUNCHER" **/>**
12. **</intent-filter>**
13. **</activity>**
14. **<activity** android:name=".SubActivity"
15. android:label="@string/sub\_activity"**>**
16. **<intent-filter>**
17. **<action** android:name="shy.luo.activity.subactivity"**/>**
18. **<category** android:name="android.intent.category.DEFAULT"**/>**
19. **</intent-filter>**
20. **</activity>**
21. **</application>**
22. **</manifest>**

从这个配置文件中，我们可以看到，MainActivity被配置成了应用程序的默认Activity，即用户在手机屏幕上点击Activity应用程序图标时，Launcher就会默认启动MainActivity这个Activity：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<activity** android:name=".MainActivity"
2. android:label="@string/app\_name"**>**
3. **<intent-filter>**
4. **<action** android:name="android.intent.action.MAIN" **/>**
5. **<category** android:name="android.intent.category.LAUNCHER" **/>**
6. **</intent-filter>**
7. **</activity>**

这个配置文件也将名字“shy.luo.activity.subactivity”和SubActivity关联了起来，因此，应用程序框架层能够根据名字来找到它：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<activity** android:name=".SubActivity"
2. android:label="@string/sub\_activity"**>**
3. **<intent-filter>**
4. **<action** android:name="shy.luo.activity.subactivity"**/>**
5. **<category** android:name="android.intent.category.DEFAULT"**/>**
6. **</intent-filter>**
7. **</activity>**

下面再列出这个应用程序的界面配置文件和字符串文件。

        界面配置文件在res/layout目录中，main.xml文件对应MainActivity的界面：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
3. android:orientation="vertical"
4. android:layout\_width="fill\_parent"
5. android:layout\_height="fill\_parent"
6. android:gravity="center"**>**
7. **<Button**
8. android:id="@+id/button\_start"
9. android:layout\_width="wrap\_content"
10. android:layout\_height="wrap\_content"
11. android:gravity="center"
12. android:text="@string/start" **>**
13. **</Button>**
14. **</LinearLayout>**

        而sub.xml对应SubActivity的界面：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
3. android:orientation="vertical"
4. android:layout\_width="fill\_parent"
5. android:layout\_height="fill\_parent"
6. android:gravity="center"**>**
7. **<Button**
8. android:id="@+id/button\_finish"
9. android:layout\_width="wrap\_content"
10. android:layout\_height="wrap\_content"
11. android:gravity="center"
12. android:text="@string/finish" **>**
13. **</Button>**
14. **</LinearLayout>**

字符串文件位于res/values/strings.xml文件中：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<resources>**
3. **<string** name="app\_name"**>**Activity**</string>**
4. **<string** name="sub\_activity"**>**Sub Activity**</string>**
5. **<string** name="start"**>**Start sub-activity**</string>**
6. **<string** name="finish"**>**Finish activity**</string>**
7. **</resources>**

最后，我们还要在工程目录下放置一个编译脚本文件Android.mk：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. LOCAL\_PATH:= $(call my-dir)
2. include $(CLEAR\_VARS)
4. LOCAL\_MODULE\_TAGS := optional
6. LOCAL\_SRC\_FILES := $(call all-subdir-java-files)
8. LOCAL\_PACKAGE\_NAME := Activity
10. include $(BUILD\_PACKAGE)

这样，整个例子的源代码实现就介绍完了，接下来就要编译了。有关如何单独编译Android源代码工程的模块，以及如何打包system.img，请参考[如何单独编译Android源代码中的模块](http://blog.csdn.net/luoshengyang/article/details/6566662)一文。  
        执行以下命令进行编译和打包：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. USER-NAME@MACHINE-NAME:~/Android$ mmm packages/experimental/Activity
2. USER-NAME@MACHINE-NAME:~/Android$ make snod

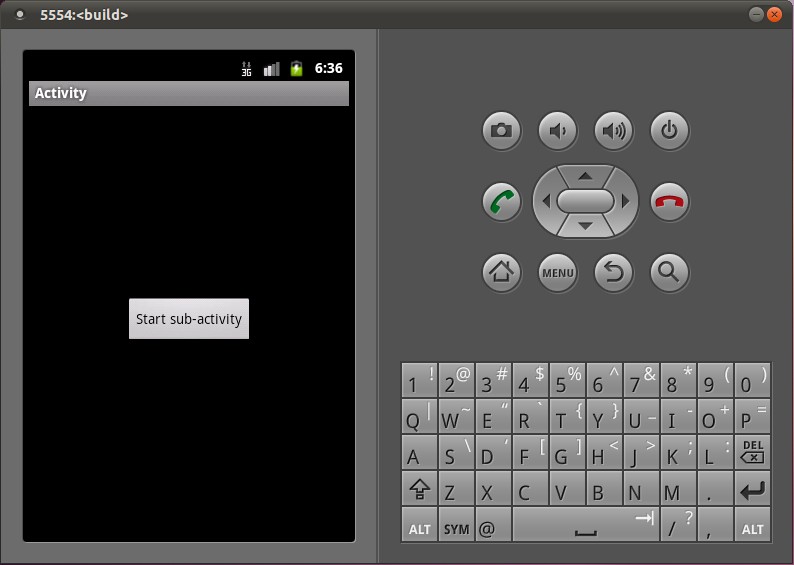
这样，打包好的Android系统镜像文件system.img就包含我们前面创建的Activity应用程序了。  
       再接下来，就是运行模拟器来运行我们的例子了。关于如何在Android源代码工程中运行模拟器，请参考[在Ubuntu上下载、编译和安装Android最新源代码](http://blog.csdn.net/luoshengyang/article/details/6559955)一文。  
       执行以下命令启动模拟器：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

1. USER-NAME@MACHINE-NAME:~/Android$ emulator

模拟器启动起，就可以在屏幕上看到Activity应用程序图标了：  


         点击Activity这个应用程序图标后，Launcher就会把MainActivity启动起来：



        点击上面的Start sub-activity铵钮，MainActivity内部就会通过startActivity接口来启动SubActivity：

[view plaincopy to clipboardprint?](http://blog.csdn.net/luoshengyang/article/details/6685853)

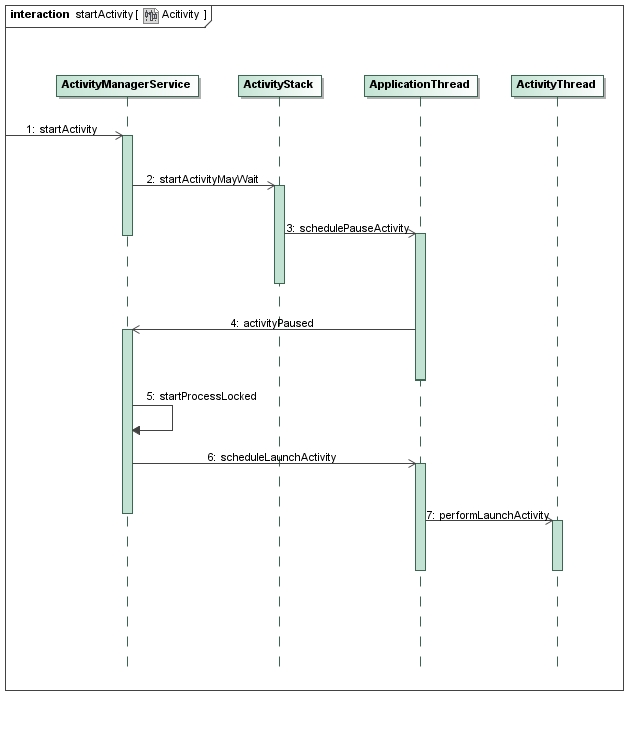
1. Intent intent = new Intent("shy.luo.activity.subactivity");
2. startActivity(intent);

如下图所示：



        无论是通过点击应用程序图标来启动Activity，还是通过Activity内部调用startActivity接口来启动新的Activity，都要借助于应用程序框架层的ActivityManagerService服务进程。在前面一篇文章[Android系统在新进程中启动自定义服务过程（startService）的原理分析](http://blog.csdn.net/luoshengyang/article/details/6677029)中，我们已经看到，Service也是由ActivityManagerService进程来启动的。在Android应用程序框架层中，ActivityManagerService是一个非常重要的接口，它不但负责启动Activity和Service，还负责管理Activity和Service。

        Android应用程序框架层中的ActivityManagerService启动Activity的过程大致如下图所示：



         在这个图中，ActivityManagerService和ActivityStack位于同一个进程中，而ApplicationThread和ActivityThread位于另一个进程中。其中，ActivityManagerService是负责管理Activity的生命周期的，ActivityManagerService还借助ActivityStack是来把所有的Activity按照后进先出的顺序放在一个堆栈中；对于每一个应用程序来说，都有一个ActivityThread来表示应用程序的主进程，而每一个ActivityThread都包含有一个ApplicationThread实例，它是一个Binder对象，负责和其它进程进行通信。

        下面简要介绍一下启动的过程：

        Step 1. 无论是通过Launcher来启动Activity，还是通过Activity内部调用startActivity接口来启动新的Activity，都通过Binder进程间通信进入到ActivityManagerService进程中，并且调用ActivityManagerService.startActivity接口；

        Step 2. ActivityManagerService调用ActivityStack.startActivityMayWait来做准备要启动的Activity的相关信息；

        Step 3. ActivityStack通知ApplicationThread要进行Activity启动调度了，这里的ApplicationThread代表的是调用ActivityManagerService.startActivity接口的进程，对于通过点击应用程序图标的情景来说，这个进程就是Launcher了，而对于通过在Activity内部调用startActivity的情景来说，这个进程就是这个Activity所在的进程了；

        Step 4. ApplicationThread不执行真正的启动操作，它通过调用ActivityManagerService.activityPaused接口进入到ActivityManagerService进程中，看看是否需要创建新的进程来启动Activity；

        Step 5. 对于通过点击应用程序图标来启动Activity的情景来说，ActivityManagerService在这一步中，会调用startProcessLocked来创建一个新的进程，而对于通过在Activity内部调用startActivity来启动新的Activity来说，这一步是不需要执行的，因为新的Activity就在原来的Activity所在的进程中进行启动；

        Step 6. ActivityManagerServic调用ApplicationThread.scheduleLaunchActivity接口，通知相应的进程执行启动Activity的操作；

        Step 7. ApplicationThread把这个启动Activity的操作转发给ActivityThread，ActivityThread通过ClassLoader导入相应的Activity类，然后把它启动起来。

        这样，Android应用程序的Activity启动过程就简要介绍到这里了，在接下来的两篇文章中，我们将根据Activity的这两种启动情景，深入到应用程序框架层的源代码里面去，一步一步地分析它们的启动过程：

        1. [Android应用程序启动过程的源代码分析；](http://blog.csdn.net/luoshengyang/article/details/6689748)

        2. [Android应用程序内部启动Activity过程（startActivity）的源代码分析。](http://blog.csdn.net/luoshengyang/article/details/6703247)