前面我们在分析Activity启动过程的时候，看到同一个应用程序的Activity一般都是在同一个进程中启动，事实上，Activity也可以像Service一样在新的进程中启动，这样，一个应用程序就可以跨越好几个进程了，本文就分析一下在新的进程中启动Activity的方法和过程。

        在前面[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)一文中，我们提到，在Android系统中，每一个应用程序都是由一些Activity和Service组成的，一般Service运行在独立的进程中，而Activity有可能运行在同一个进程中，也有可能运行在不同的进程中。在前面[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)一文中，我们已经介绍了使用Activity.startService接口来在新进程中启动Service的过程，然后又在前面[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)一文中介绍了使用Activity.startActivity接口来在原来的进程中启动Activity的过程，现在，我们就来看一下同一个Android应用程序如何在新的进程中启动新的Activity。

        老规矩，我们通过例子来介绍Android应用程序在新的进程中启动新的Activity的方法以及分析其过程。首先在Android源代码工程中创建一个Android应用程序工程，名字就称为Process吧。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。这个应用程序工程定义了一个名为shy.luo.process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/process/MainActivity.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. package shy.luo.process;
3. import android.app.Activity;
4. import android.content.Intent;
5. import android.os.Bundle;
6. import android.util.Log;
7. import android.view.View;
8. import android.view.View.OnClickListener;
9. import android.widget.Button;
11. public class MainActivity extends Activity  implements OnClickListener {
12. private final static String LOG\_TAG = "shy.luo.process.MainActivity";
14. private Button startButton = null;
16. @Override
17. public void onCreate(Bundle savedInstanceState) {
18. super.onCreate(savedInstanceState);
19. setContentView(R.layout.main);
21. startButton = (Button)findViewById(R.id.button\_start);
22. startButton.setOnClickListener(this);
24. Log.i(LOG\_TAG, "Main Activity Created.");
25. }
27. @Override
28. public void onClick(View v) {
29. if(v.equals(startButton)) {
30. Intent intent = new Intent("shy.luo.process.subactivity");
31. startActivity(intent);
32. }
33. }
34. }

package shy.luo.process;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

private final static String LOG\_TAG = "shy.luo.process.MainActivity";

private Button startButton = null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

startButton = (Button)findViewById(R.id.button\_start);

startButton.setOnClickListener(this);

Log.i(LOG\_TAG, "Main Activity Created.");

}

@Override

public void onClick(View v) {

if(v.equals(startButton)) {

Intent intent = new Intent("shy.luo.process.subactivity");

startActivity(intent);

}

}

}

        和前面文章的例子一样，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.process.subactivity”的Actvity。  
        名字为“shy.luo.process.subactivity”的Actvity实现在src/shy/luo/process/SubActivity.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. package shy.luo.process;
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.process.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. }

package shy.luo.process;

import android.app.Activity;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

public class SubActivity extends Activity implements OnClickListener {

private final static String LOG\_TAG = "shy.luo.process.SubActivity";

private Button finishButton = null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.sub);

finishButton = (Button)findViewById(R.id.button\_finish);

finishButton.setOnClickListener(this);

Log.i(LOG\_TAG, "Sub Activity Created.");

}

@Override

public void onClick(View v) {

if(v.equals(finishButton)) {

finish();

}

}

}

        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。  
        再来重点看一下应用程序的配置文件AndroidManifest.xml：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<manifest** xmlns:android="http://schemas.android.com/apk/res/android"
3. package="shy.luo.task"
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. android:process=":shy.luo.process.main"
10. **<intent-filter>**
11. **<action** android:name="android.intent.action.MAIN" **/>**
12. **<category** android:name="android.intent.category.LAUNCHER" **/>**
13. **</intent-filter>**
14. **</activity>**
15. **<activity** android:name=".SubActivity"
16. android:label="@string/sub\_activity"
17. android:process=":shy.luo.process.sub"**>**
18. **<intent-filter>**
19. **<action** android:name="shy.luo.task.subactivity"**/>**
20. **<category** android:name="android.intent.category.DEFAULT"**/>**
21. **</intent-filter>**
22. **</activity>**
23. **</application>**
24. **</manifest>**

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="shy.luo.task"

android:versionCode="1"

android:versionName="1.0">

<application android:icon="@drawable/icon" android:label="@string/app\_name">

<activity android:name=".MainActivity"

android:label="@string/app\_name">

android:process=":shy.luo.process.main"

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity android:name=".SubActivity"

android:label="@string/sub\_activity"

android:process=":shy.luo.process.sub">

<intent-filter>

<action android:name="shy.luo.task.subactivity"/>

<category android:name="android.intent.category.DEFAULT"/>

</intent-filter>

</activity>

</application>

</manifest>

        为了使MainActivity和SubActivity在不同的进程中启动，我们分别配置了这两个Activity的android:process属性。在官方文档<http://developer.android.com/guide/topics/manifest/activity-element.html>中，是这样介绍Activity的android:process属性的：

        The name of the process in which the activity should run. Normally, all components of an application run in the default process created for the application. It has the same name as the application package. The [<application>](http://developer.android.com/guide/topics/manifest/application-element.html) element's [process](http://developer.android.com/guide/topics/manifest/application-element.html#proc) attribute can set a different default for all components. But each component can override the default, allowing you to spread your application across multiple processes.  
        If the name assigned to this attribute begins with a colon (':'), a new process, private to the application, is created when it's needed and the activity runs in that process. If the process name begins with a lowercase character, the activity will run in a global process of that name, provided that it has permission to do so. This allows components in different applications to share a process, reducing resource usage.  
        大意为，一般情况下，同一个应用程序的Activity组件都是运行在同一个进程中，但是，如果Activity配置了android:process这个属性，那么，它就会运行在自己的进程中。如果android:process属性的值以":"开头，则表示这个进程是私有的；如果android:process属性的值以小写字母开头，则表示这是一个全局进程，允许其它应用程序组件也在这个进程中运行。

        因此，这里我们以":"开头，表示创建的是私有的进程。事实上，这里我们不要前面的":"也是可以的，但是必须保证这个属性性字符串内至少有一个"."字符，具体可以看一下解析AndroidManiefst.xml文件的源代码，在frameworks/base/core/java/android/content/pm/PackageParser.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. public class PackageParser {
3. ......
5. private boolean parseApplication(Package owner, Resources res,
6. XmlPullParser parser, AttributeSet attrs, int flags, String[] outError)
7. throws XmlPullParserException, IOException {
8. final ApplicationInfo ai = owner.applicationInfo;
9. final String pkgName = owner.applicationInfo.packageName;
11. TypedArray sa = res.obtainAttributes(attrs,
12. com.android.internal.R.styleable.AndroidManifestApplication);
14. ......
16. if (outError[0] == null) {
17. CharSequence pname;
18. if (owner.applicationInfo.targetSdkVersion >= Build.VERSION\_CODES.FROYO) {
19. pname = sa.getNonConfigurationString(
20. com.android.internal.R.styleable.AndroidManifestApplication\_process, 0);
21. } else {
22. // Some older apps have been seen to use a resource reference
23. // here that on older builds was ignored (with a warning).  We
24. // need to continue to do this for them so they don't break.
25. pname = sa.getNonResourceString(
26. com.android.internal.R.styleable.AndroidManifestApplication\_process);
27. }
28. ai.processName = buildProcessName(ai.packageName, null, pname,
29. flags, mSeparateProcesses, outError);
31. ......
32. }
34. ......
36. }
38. private static String buildProcessName(String pkg, String defProc,
39. CharSequence procSeq, int flags, String[] separateProcesses,
40. String[] outError) {
41. if ((flags&PARSE\_IGNORE\_PROCESSES) != 0 && !"system".equals(procSeq)) {
42. return defProc != null ? defProc : pkg;
43. }
44. if (separateProcesses != null) {
45. for (int i=separateProcesses.length-1; i>=0; i--) {
46. String sp = separateProcesses[i];
47. if (sp.equals(pkg) || sp.equals(defProc) || sp.equals(procSeq)) {
48. return pkg;
49. }
50. }
51. }
52. if (procSeq == null || procSeq.length() <= 0) {
53. return defProc;
54. }
55. return buildCompoundName(pkg, procSeq, "process", outError);
56. }
58. private static String buildCompoundName(String pkg,
59. CharSequence procSeq, String type, String[] outError) {
60. String proc = procSeq.toString();
61. char c = proc.charAt(0);
62. if (pkg != null && c == ':') {
63. if (proc.length() < 2) {
64. outError[0] = "Bad " + type + " name " + proc + " in package " + pkg
65. + ": must be at least two characters";
66. return null;
67. }
68. String subName = proc.substring(1);
69. String nameError = validateName(subName, false);
70. if (nameError != null) {
71. outError[0] = "Invalid " + type + " name " + proc + " in package "
72. + pkg + ": " + nameError;
73. return null;
74. }
75. return (pkg + proc).intern();
76. }
77. String nameError = validateName(proc, true);
78. if (nameError != null && !"system".equals(proc)) {
79. outError[0] = "Invalid " + type + " name " + proc + " in package "
80. + pkg + ": " + nameError;
81. return null;
82. }
83. return proc.intern();
84. }

87. private static String validateName(String name, boolean requiresSeparator) {
88. final int N = name.length();
89. boolean hasSep = false;
90. boolean front = true;
91. for (int i=0; i<N; i++) {
92. final char c = name.charAt(i);
93. if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
94. front = false;
95. continue;
96. }
97. if (!front) {
98. if ((c >= '0' && c <= '9') || c == '\_') {
99. continue;
100. }
101. }
102. if (c == '.') {
103. hasSep = true;
104. front = true;
105. continue;
106. }
107. return "bad character '" + c + "'";
108. }
110. return hasSep || !requiresSeparator
111. ? null : "must have at least one '.' separator";
112. }
114. ......
116. }

public class PackageParser {

......

private boolean parseApplication(Package owner, Resources res,

XmlPullParser parser, AttributeSet attrs, int flags, String[] outError)

throws XmlPullParserException, IOException {

final ApplicationInfo ai = owner.applicationInfo;

final String pkgName = owner.applicationInfo.packageName;

TypedArray sa = res.obtainAttributes(attrs,

com.android.internal.R.styleable.AndroidManifestApplication);

......

if (outError[0] == null) {

CharSequence pname;

if (owner.applicationInfo.targetSdkVersion >= Build.VERSION\_CODES.FROYO) {

pname = sa.getNonConfigurationString(

com.android.internal.R.styleable.AndroidManifestApplication\_process, 0);

} else {

// Some older apps have been seen to use a resource reference

// here that on older builds was ignored (with a warning). We

// need to continue to do this for them so they don't break.

pname = sa.getNonResourceString(

com.android.internal.R.styleable.AndroidManifestApplication\_process);

}

ai.processName = buildProcessName(ai.packageName, null, pname,

flags, mSeparateProcesses, outError);

......

}

......

}

private static String buildProcessName(String pkg, String defProc,

CharSequence procSeq, int flags, String[] separateProcesses,

String[] outError) {

if ((flags&PARSE\_IGNORE\_PROCESSES) != 0 && !"system".equals(procSeq)) {

return defProc != null ? defProc : pkg;

}

if (separateProcesses != null) {

for (int i=separateProcesses.length-1; i>=0; i--) {

String sp = separateProcesses[i];

if (sp.equals(pkg) || sp.equals(defProc) || sp.equals(procSeq)) {

return pkg;

}

}

}

if (procSeq == null || procSeq.length() <= 0) {

return defProc;

}

return buildCompoundName(pkg, procSeq, "process", outError);

}

private static String buildCompoundName(String pkg,

CharSequence procSeq, String type, String[] outError) {

String proc = procSeq.toString();

char c = proc.charAt(0);

if (pkg != null && c == ':') {

if (proc.length() < 2) {

outError[0] = "Bad " + type + " name " + proc + " in package " + pkg

+ ": must be at least two characters";

return null;

}

String subName = proc.substring(1);

String nameError = validateName(subName, false);

if (nameError != null) {

outError[0] = "Invalid " + type + " name " + proc + " in package "

+ pkg + ": " + nameError;

return null;

}

return (pkg + proc).intern();

}

String nameError = validateName(proc, true);

if (nameError != null && !"system".equals(proc)) {

outError[0] = "Invalid " + type + " name " + proc + " in package "

+ pkg + ": " + nameError;

return null;

}

return proc.intern();

}

private static String validateName(String name, boolean requiresSeparator) {

final int N = name.length();

boolean hasSep = false;

boolean front = true;

for (int i=0; i<N; i++) {

final char c = name.charAt(i);

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {

front = false;

continue;

}

if (!front) {

if ((c >= '0' && c <= '9') || c == '\_') {

continue;

}

}

if (c == '.') {

hasSep = true;

front = true;

continue;

}

return "bad character '" + c + "'";

}

return hasSep || !requiresSeparator

? null : "must have at least one '.' separator";

}

......

}

        从调用parseApplication函数解析application标签开始，通过调用buildProcessName函数对android:process属性进解析，接着又会调用buildCompoundName进一步解析，这里传进来的参数pkg就为"shy.luo.process"，参数procSeq为MainActivity的属性android:process的值":shy.luo.process.main"，进一步将这个字符串保存在本地变量proc中。如果proc的第一个字符是":"，则只需要调用validateName函数来验证proc字符串里面的字符都是合法组成就可以了，即以大小写字母或者"."开头，后面可以跟数字或者"\_"字符；如果proc的第一个字符不是":"，除了保证proc字符里面的字符都是合法组成外，还要求至少有一个"."字符。

        MainActivity和SubActivity的android:process属性配置就介绍到这里了，其它更多的信息读者可以参考官方文档<http://developer.android.com/guide/topics/manifest/activity-element.html>或者源代码文件frameworks/base/core/java/android/content/pm/PackageParser.java。

        再来看界面配置文件，它们定义在res/layout目录中，main.xml文件对应MainActivity的界面：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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>**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:gravity="center">

<Button

android:id="@+id/button\_start"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:gravity="center"

android:text="@string/start" >

</Button>

</LinearLayout>

        而sub.xml对应SubActivity的界面：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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>**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:gravity="center">

<Button

android:id="@+id/button\_finish"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:gravity="center"

android:text="@string/finish" >

</Button>

</LinearLayout>

        字符串文件位于res/values/strings.xml文件中：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Process</string>

<string name="sub\_activity">Sub Activity</string>

<string name="start">Start activity in new process</string>

<string name="finish">Finish activity</string>

</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：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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 := Process
10. include $(BUILD\_PACKAGE)

LOCAL\_PATH:= $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_MODULE\_TAGS := optional

LOCAL\_SRC\_FILES := $(call all-subdir-java-files)

LOCAL\_PACKAGE\_NAME := Process

include $(BUILD\_PACKAGE)

       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：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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

USER-NAME@MACHINE-NAME:~/Android$ mmm packages/experimental/Process

USER-NAME@MACHINE-NAME:~/Android$ make snod

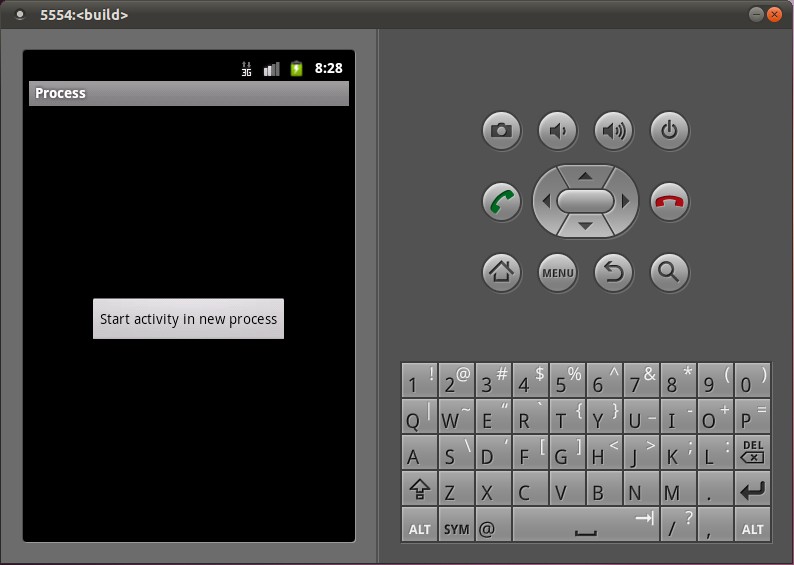
       这样，打包好的Android系统镜像文件system.img就包含我们前面创建的Process应用程序了。  
       再接下来，就是运行模拟器来运行我们的例子了。关于如何在Android源代码工程中运行模拟器，请参考[在Ubuntu上下载、编译和安装Android最新源代码](http://blog.csdn.net/luoshengyang/article/details/6559955)一文。  
       执行以下命令启动模拟器：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

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

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

       模拟器启动起，就可以App Launcher中找到Process应用程序图标，接着把它启动起来：



      点击中间的按钮，就会在新的进程中启动SubActivity：



        现在，我们如何来确认SubActivity是不是在新的进程中启动呢？Android源代码工程为我们准备了adb工具，可以查看模拟器上系统运行的状况，执行下面的命令查看：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. USER-NAME@MACHINE-NAME:~/Android$ adb shell dumpsys activity

USER-NAME@MACHINE-NAME:~/Android$ adb shell dumpsys activity

       这个命令输出的内容比较多，这里我们只关心系统中的任务和进程部分：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. ......
3. Running activities (most recent first):
4. TaskRecord{40770440 #3 A shy.luo.process}
5. Run #2: HistoryRecord{406d4b20 shy.luo.process/.SubActivity}
6. Run #1: HistoryRecord{40662bd8 shy.luo.process/.MainActivity}
7. TaskRecord{40679eb8 #2 A com.android.launcher}
8. Run #0: HistoryRecord{40677570 com.android.launcher/com.android.launcher2.Launcher}
10. ......
12. PID mappings:
13. ......
15. PID #416: ProcessRecord{4064b720 416:shy.luo.process:shy.luo.process.main/10037}
16. PID #425: ProcessRecord{406ddc30 425:shy.luo.process:shy.luo.process.sub/10037}
18. ......

......

Running activities (most recent first):

TaskRecord{40770440 #3 A shy.luo.process}

Run #2: HistoryRecord{406d4b20 shy.luo.process/.SubActivity}

Run #1: HistoryRecord{40662bd8 shy.luo.process/.MainActivity}

TaskRecord{40679eb8 #2 A com.android.launcher}

Run #0: HistoryRecord{40677570 com.android.launcher/com.android.launcher2.Launcher}

......

PID mappings:

......

PID #416: ProcessRecord{4064b720 416:shy.luo.process:shy.luo.process.main/10037}

PID #425: ProcessRecord{406ddc30 425:shy.luo.process:shy.luo.process.sub/10037}

......

        这里我们看到，虽然MainActivity和SubActivity都是在同一个应用程序并且运行在同一个任务中，然而，它们却是运行在两个不同的进程中，这就可以看到Android系统中任务这个概念的强大之处了，它使得我们在开发应用程序的时候，可以把相对独立的模块放在独立的进程中运行，以降低模块之间的耦合性，同时，我们又不必去考虑一个应用程序在两个进程中运行的细节的问题，Android系统中的任务会为我们打点好一切。

        在启动Activity的时候，系统是如何做到在新的进程中来启动这个Activity的呢？在前面两篇文章[Android应用程序启动过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6689748)和[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)中，分别在Step 22和Step 21中分析了Activity在启动过程中与进程相关的函数ActivityStack.startSpecificActivityLocked函数中，它定义在frameworks/base/services/java/com/android/server/am/ActivityStack.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6720261)

1. public class ActivityStack {
3. ......
5. private final void startSpecificActivityLocked(ActivityRecord r,
6. boolean andResume, boolean checkConfig) {
7. // Is this activity's application already running?
8. ProcessRecord app = mService.getProcessRecordLocked(r.processName,
9. r.info.applicationInfo.uid);
11. ......
13. if (app != null && app.thread != null) {
14. try {
15. realStartActivityLocked(r, app, andResume, checkConfig);
16. return;
17. } catch (RemoteException e) {
18. ......
19. }
20. }
22. mService.startProcessLocked(r.processName, r.info.applicationInfo, true, 0,
23. "activity", r.intent.getComponent(), false);
24. }

27. ......
29. }

public class ActivityStack {

......

private final void startSpecificActivityLocked(ActivityRecord r,

boolean andResume, boolean checkConfig) {

// Is this activity's application already running?

ProcessRecord app = mService.getProcessRecordLocked(r.processName,

r.info.applicationInfo.uid);

......

if (app != null && app.thread != null) {

try {

realStartActivityLocked(r, app, andResume, checkConfig);

return;

} catch (RemoteException e) {

......

}

}

mService.startProcessLocked(r.processName, r.info.applicationInfo, true, 0,

"activity", r.intent.getComponent(), false);

}

......

}

        从这个函数可以看出，决定一个Activity是在新的进程中启动还是在原有的进程中启动的因素有两个，一个是看这个Activity的process属性的值，另一个是这个Activity所在的应用程序的uid。应用程序的UID是由系统分配的，而Activity的process属性值，如前所述，是可以在AndroidManifest.xml文件中进行配置的，如果没有配置，它默认就为application标签的process属性值，如果application标签的process属性值也没有配置，那么，它们就默认为应用程序的package名。这里就是根据processName和uid在系统查找是否已有相应的进程存在，如果已经有了，就会调用realStartActivityLocked来直接启动Activity，否则的话，就要通过调用ActivityManagerService.startProcessLocked函数来创建一个新的进程，然后在新进程中启动这个Activity了。对于前者，可以参考[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)一文，而后者，可以参考[Android应用程序启动过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6689748)一文。

        至此，Android应用程序在新的进程中启动新的Activity的方法和过程分析就结束了。在实际开发中，一个应用程序一般很少会在一个新的进程中启动另外一个Activity，如果真的需要这样做，还要考虑如何与应用程序中其它进程的Activity进行通信，这时候不妨考虑使用[Binder进程间通信机制](http://blog.csdn.net/luoshengyang/article/details/6618363)。写这篇文章的目的，更多是让我们去了解Android应用程序的架构，这种架构可以使得应用程序组件以松耦合的方式组合在一起，便于后续的扩展和维护，这是非常值得我们学习的。