在Android应用程序中，可以配置Activity以四种方式来启动，其中最令人迷惑的就是"singleTask"这种方式了，官方文档称以这种方式启动的Activity总是属于一个任务的根Activity。果真如此吗？本文将为你解开Activity的"singleTask"之谜。

        在解开这个谜之前，我们先来简单了解一下在Android应用程序中，任务（Task）是个什么样的概念。我们知道，Activity是Android应用程序的基础组件之一，在应用程序运行时，每一个Activity代表一个用户操作。用户为了完成某个功能而执行的一系列操作就形成了一个Activity序列，这个序列在Android应用程序中就称之为任务，它是从用户体验的角度出发，把一组相关的Activity组织在一起而抽象出来的概念。

        对初学者来说，在开发Android应用程序时，对任务的概念可能不是那么的直观，一般我们只关注如何实现应用程序中的每一个Activity。事实上，Android系统中的任务更多的是体现是应用程序运行的时候，因此，它相对于Activity来说是动态存在的，这就是为什么我们在开发时对任务这个概念不是那么直观的原因。不过，我们在开发Android应用程序时，还是可以配置Activity的任务属性的，即告诉系统，它是要在新的任务中启动呢，还是在已有的任务中启动，亦或是其它的Activity能不能与它共享同一个任务，具体配置请参考官方文档：

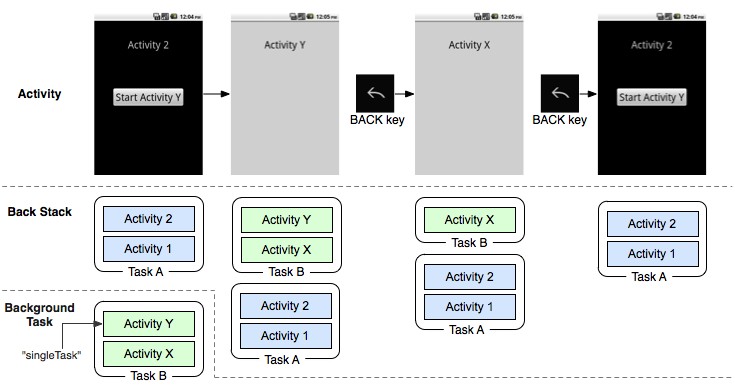
<http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html>

       它是这样介绍以"singleTask"方式启动的Activity的：

       The system creates a new task and instantiates the activity at the root of the new task. However, if an instance of the activity already exists in a separate task, the system routes the intent to the existing instance through a call to its onNewIntent() method, rather than creating a new instance. Only one instance of the activity can exist at a time.

       它明确说明，以"singleTask"方式启动的Activity，全局只有唯一个实例存在，因此，当我们第一次启动这个Activity时，系统便会创建一个新的任务，并且初始化一个这样的Activity的实例，放在新任务的底部，如果下次再启动这个Activity时，系统发现已经存在这样的Activity实例，就会调用这个Activity实例的onNewIntent成员函数，从而把它激活起来。从这句话就可以推断出，以"singleTask"方式启动的Activity总是属于一个任务的根Activity。

       但是文档接着举例子说明，当用户按下键盘上的Back键时，如果此时在前台中运行的任务堆栈顶端是一个"singleTask"的Activity，系统会回到当前任务的下一个Activity中去，而不是回到前一个Activity中去，如下图所示：



        真是坑爹啊！有木有！前面刚说"singleTask"会在新的任务中运行，并且位于任务堆栈的底部，这里在Task B中，一个赤裸裸的带着"singleTask"标签的箭头无情地指向Task B堆栈顶端的Activity Y，刚转身就翻脸不认人了呢！

        狮屎胜于熊便，我们来做一个实验吧，看看到底在启动这个"singleTask"的Activity的时候，它是位于新任务堆栈的底部呢，还是在已有任务的顶部。

        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Task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.task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/task/MainActivity.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. package shy.luo.task;
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.task.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.task.subactivity");
31. startActivity(intent);
32. }
33. }
34. }

package shy.luo.task;

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.task.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.task.subactivity");

startActivity(intent);

}

}

}

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

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. package shy.luo.task;
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.task.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.task;

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.task.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。

        再来看一下应用程序的配置文件AndroidManifest.xml：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

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. **<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. android:launchMode="singleTask"**>**
17. **<intent-filter>**
18. **<action** android:name="shy.luo.task.subactivity"**/>**
19. **<category** android:name="android.intent.category.DEFAULT"**/>**
20. **</intent-filter>**
21. **</activity>**
22. **</application>**
23. **</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">

<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:launchMode="singleTask">

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

</manifest>

       注意，这里的SubActivity的launchMode属性配置为"singleTask"。  
       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/6714543)

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/6714543)

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/6714543)

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

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

<resources>

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

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

<string name="start">Start singleTask activity</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/6714543)

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 := Task
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 := Task

include $(BUILD\_PACKAGE)

        这样，原材料就准备好了，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：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

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

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

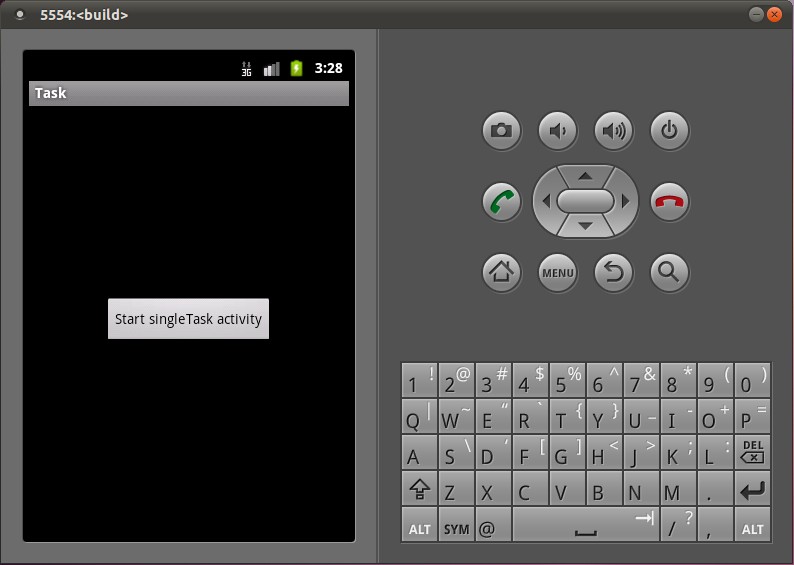
USER-NAME@MACHINE-NAME:~/Android$ make snod

       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Task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：

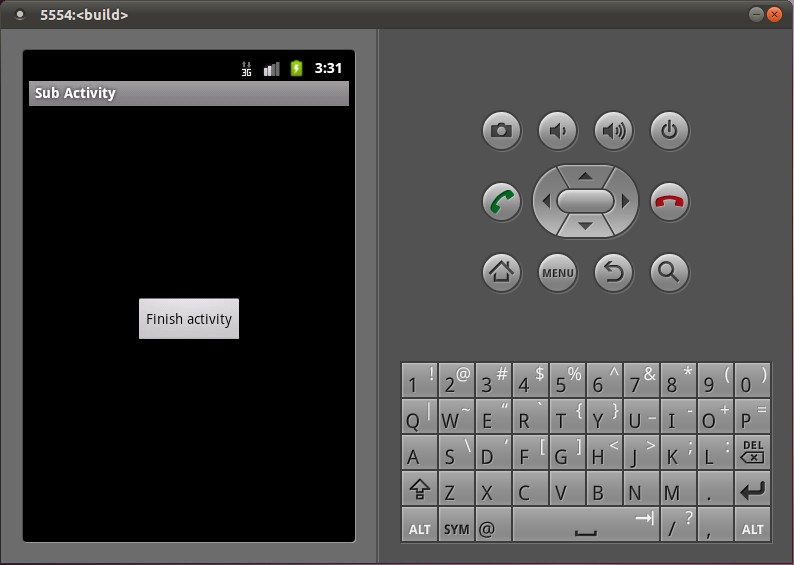
**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

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Task应用程序图标，接着把它启动起来：  


         点击中间的按钮，就会以"singleTask"的方式来启动SubActivity：



        现在，我们如何来确认SubActivity是不是在新的任务中启动并且位于这个新任务的堆栈底部呢？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/6714543)

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TaskRecord部分：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. Running activities (most recent first):
2. TaskRecord{4070d8f8 #3 A shy.luo.task}
3. Run #2: HistoryRecord{406a13f8 shy.luo.task/.SubActivity}
4. Run #1: HistoryRecord{406a0e00 shy.luo.task/.MainActivity}
5. TaskRecord{4067a510 #2 A com.android.launcher}
6. Run #0: HistoryRecord{40677518 com.android.launcher/com.android.launcher2.Launcher}

Running activities (most recent first):

TaskRecord{4070d8f8 #3 A shy.luo.task}

Run #2: HistoryRecord{406a13f8 shy.luo.task/.SubActivity}

Run #1: HistoryRecord{406a0e00 shy.luo.task/.MainActivity}

TaskRecord{4067a510 #2 A com.android.launcher}

Run #0: HistoryRecord{40677518 com.android.launcher/com.android.launcher2.Launcher}

        果然，SubActivity和MainActivity都是运行在TaskRecord#3中，并且SubActivity在MainActivity的上面。这是怎么回事呢？碰到这种情况，Linus Torvalds告诫我们：Read the fucking source code；去年张麻子又说：枪在手，跟我走；我们没有枪，但是有source code，因此，我要说：跟着代码走。

        前面我们在两篇文章[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)时，分别在Step 9和Step 8中分析了Activity在启动过程中与任务相关的函数ActivityStack.startActivityUncheckedLocked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/6714543)

1. public class ActivityStack {
3. ......
5. final int startActivityUncheckedLocked(ActivityRecord r,
6. ActivityRecord sourceRecord, Uri[] grantedUriPermissions,
7. int grantedMode, boolean onlyIfNeeded, boolean doResume) {
8. final Intent intent = r.intent;
9. final int callingUid = r.launchedFromUid;
11. int launchFlags = intent.getFlags();
13. ......
15. ActivityRecord notTop = (launchFlags&Intent.FLAG\_ACTIVITY\_PREVIOUS\_IS\_TOP)
16. != 0 ? r : null;
18. ......
20. if (sourceRecord == null) {
21. ......
22. } else if (sourceRecord.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {
23. ......
24. } else if (r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE
25. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK) {
26. // The activity being started is a single instance...  it always
27. // gets launched into its own task.
28. launchFlags |= Intent.FLAG\_ACTIVITY\_NEW\_TASK;
29. }
31. ......
33. boolean addingToTask = false;
34. if (((launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0 &&
35. (launchFlags&Intent.FLAG\_ACTIVITY\_MULTIPLE\_TASK) == 0)
36. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK
37. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {
38. // If bring to front is requested, and no result is requested, and
39. // we can find a task that was started with this same
40. // component, then instead of launching bring that one to the front.
41. if (r.resultTo == null) {
42. // See if there is a task to bring to the front.  If this is
43. // a SINGLE\_INSTANCE activity, there can be one and only one
44. // instance of it in the history, and it is always in its own
45. // unique task, so we do a special search.
46. ActivityRecord taskTop = r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE
47. ? findTaskLocked(intent, r.info)
48. : findActivityLocked(intent, r.info);
49. if (taskTop != null) {
51. ......
53. if ((launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0
54. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK
55. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {
56. // In this situation we want to remove all activities
57. // from the task up to the one being started.  In most
58. // cases this means we are resetting the task to its
59. // initial state.
60. ActivityRecord top = performClearTaskLocked(
61. taskTop.task.taskId, r, launchFlags, true);
62. if (top != null) {
63. ......
64. } else {
65. // A special case: we need to
66. // start the activity because it is not currently
67. // running, and the caller has asked to clear the
68. // current task to have this activity at the top.
69. addingToTask = true;
70. // Now pretend like this activity is being started
71. // by the top of its task, so it is put in the
72. // right place.
73. sourceRecord = taskTop;
74. }
75. } else if (r.realActivity.equals(taskTop.task.realActivity)) {
76. ......
77. } else if ((launchFlags&Intent.FLAG\_ACTIVITY\_RESET\_TASK\_IF\_NEEDED) == 0) {
78. ......
79. } else if (!taskTop.task.rootWasReset) {
80. ......
81. }
83. ......
84. }
85. }
86. }
88. ......
90. if (r.packageName != null) {
91. // If the activity being launched is the same as the one currently
92. // at the top, then we need to check if it should only be launched
93. // once.
94. ActivityRecord top = topRunningNonDelayedActivityLocked(notTop);
95. if (top != null && r.resultTo == null) {
96. if (top.realActivity.equals(r.realActivity)) {
97. if (top.app != null && top.app.thread != null) {
98. ......
99. }
100. }
101. }
103. } else {
104. ......
105. }
107. boolean newTask = false;
109. // Should this be considered a new task?
110. if (r.resultTo == null && !addingToTask
111. && (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {
112. // todo: should do better management of integers.
113. mService.mCurTask++;
114. if (mService.mCurTask <= 0) {
115. mService.mCurTask = 1;
116. }
117. r.task = new TaskRecord(mService.mCurTask, r.info, intent,
118. (r.info.flags&ActivityInfo.FLAG\_CLEAR\_TASK\_ON\_LAUNCH) != 0);
119. if (DEBUG\_TASKS) Slog.v(TAG, "Starting new activity " + r
120. + " in new task " + r.task);
121. newTask = true;
122. if (mMainStack) {
123. mService.addRecentTaskLocked(r.task);
124. }
125. } else if (sourceRecord != null) {
126. if (!addingToTask &&
127. (launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0) {
128. ......
129. } else if (!addingToTask &&
130. (launchFlags&Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT) != 0) {
131. ......
132. }
133. // An existing activity is starting this new activity, so we want
134. // to keep the new one in the same task as the one that is starting
135. // it.
136. r.task = sourceRecord.task;
138. ......
140. } else {
141. ......
142. }
144. ......
146. startActivityLocked(r, newTask, doResume);
147. return START\_SUCCESS;
148. }
150. ......
152. }

public class ActivityStack {

......

final int startActivityUncheckedLocked(ActivityRecord r,

ActivityRecord sourceRecord, Uri[] grantedUriPermissions,

int grantedMode, boolean onlyIfNeeded, boolean doResume) {

final Intent intent = r.intent;

final int callingUid = r.launchedFromUid;

int launchFlags = intent.getFlags();

......

ActivityRecord notTop = (launchFlags&Intent.FLAG\_ACTIVITY\_PREVIOUS\_IS\_TOP)

!= 0 ? r : null;

......

if (sourceRecord == null) {

......

} else if (sourceRecord.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {

......

} else if (r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK) {

// The activity being started is a single instance... it always

// gets launched into its own task.

launchFlags |= Intent.FLAG\_ACTIVITY\_NEW\_TASK;

}

......

boolean addingToTask = false;

if (((launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0 &&

(launchFlags&Intent.FLAG\_ACTIVITY\_MULTIPLE\_TASK) == 0)

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {

// If bring to front is requested, and no result is requested, and

// we can find a task that was started with this same

// component, then instead of launching bring that one to the front.

if (r.resultTo == null) {

// See if there is a task to bring to the front. If this is

// a SINGLE\_INSTANCE activity, there can be one and only one

// instance of it in the history, and it is always in its own

// unique task, so we do a special search.

ActivityRecord taskTop = r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE

? findTaskLocked(intent, r.info)

: findActivityLocked(intent, r.info);

if (taskTop != null) {

......

if ((launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {

// In this situation we want to remove all activities

// from the task up to the one being started. In most

// cases this means we are resetting the task to its

// initial state.

ActivityRecord top = performClearTaskLocked(

taskTop.task.taskId, r, launchFlags, true);

if (top != null) {

......

} else {

// A special case: we need to

// start the activity because it is not currently

// running, and the caller has asked to clear the

// current task to have this activity at the top.

addingToTask = true;

// Now pretend like this activity is being started

// by the top of its task, so it is put in the

// right place.

sourceRecord = taskTop;

}

} else if (r.realActivity.equals(taskTop.task.realActivity)) {

......

} else if ((launchFlags&Intent.FLAG\_ACTIVITY\_RESET\_TASK\_IF\_NEEDED) == 0) {

......

} else if (!taskTop.task.rootWasReset) {

......

}

......

}

}

}

......

if (r.packageName != null) {

// If the activity being launched is the same as the one currently

// at the top, then we need to check if it should only be launched

// once.

ActivityRecord top = topRunningNonDelayedActivityLocked(notTop);

if (top != null && r.resultTo == null) {

if (top.realActivity.equals(r.realActivity)) {

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

......

}

}

}

} else {

......

}

boolean newTask = false;

// Should this be considered a new task?

if (r.resultTo == null && !addingToTask

&& (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {

// todo: should do better management of integers.

mService.mCurTask++;

if (mService.mCurTask <= 0) {

mService.mCurTask = 1;

}

r.task = new TaskRecord(mService.mCurTask, r.info, intent,

(r.info.flags&ActivityInfo.FLAG\_CLEAR\_TASK\_ON\_LAUNCH) != 0);

if (DEBUG\_TASKS) Slog.v(TAG, "Starting new activity " + r

+ " in new task " + r.task);

newTask = true;

if (mMainStack) {

mService.addRecentTaskLocked(r.task);

}

} else if (sourceRecord != null) {

if (!addingToTask &&

(launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0) {

......

} else if (!addingToTask &&

(launchFlags&Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT) != 0) {

......

}

// An existing activity is starting this new activity, so we want

// to keep the new one in the same task as the one that is starting

// it.

r.task = sourceRecord.task;

......

} else {

......

}

......

startActivityLocked(r, newTask, doResume);

return START\_SUCCESS;

}

......

}

        首先是获得用来启动Activity的Intent的Flags，并且保存在launchFlags变量中，这里，launcFlags的Intent.FLAG\_ACTIVITY\_PREVIOUS\_IS\_TOP位没有置位，因此，notTop为null。

        接下来的这个if语句：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. if (sourceRecord == null) {
2. ......
3. } else if (sourceRecord.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {
4. ......
5. } else if (r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE
6. || r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK) {
7. // The activity being started is a single instance...  it always
8. // gets launched into its own task.
9. launchFlags |= Intent.FLAG\_ACTIVITY\_NEW\_TASK;
10. }

if (sourceRecord == null) {

......

} else if (sourceRecord.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {

......

} else if (r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_INSTANCE

|| r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK) {

// The activity being started is a single instance... it always

// gets launched into its own task.

launchFlags |= Intent.FLAG\_ACTIVITY\_NEW\_TASK;

}

        这里变量r的类型为ActivityRecord，它表示即将在启动的Activity，在这个例子中，即为SubActivity，因此，这里的r.launchMode等于ActivityInfo.LAUNCH\_SINGLE\_TASK，于是，无条件将launchFlags的Intent.FLAG\_ACTIVITY\_PREVIOUS\_IS\_TOP位置为1，表示这个SubActivity要在新的任务中启动，但是别急，还要看看其它条件是否满足，如果条件都满足，才可以在新的任务中启动这个SubActivity。  
        接下将addingToTask变量初始化为false，这个变量也将决定是否要将SubActivity在新的任务中启动，从名字我们就可以看出，默认不增加到原有的任务中启动，即要在新的任务中启动。这里的r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK条成立，条件r.resultTo == null也成立，它表这个Activity不需要将结果返回给启动它的Activity。于是会进入接下来的if语句中，执行：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. ActivityRecord taskTop = r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE
2. ? findTaskLocked(intent, r.info)
3. : findActivityLocked(intent, r.info)

ActivityRecord taskTop = r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE

? findTaskLocked(intent, r.info)

: findActivityLocked(intent, r.info)

        这里的条件r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE成立，于是执行findTaskLocked函数，这个函数也是定义在frameworks/base/services/java/com/android/server/am/ActivityStack.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. public class ActivityStack {
3. ......
5. /\*\*
6. \* Returns the top activity in any existing task matching the given
7. \* Intent.  Returns null if no such task is found.
8. \*/
9. private ActivityRecord findTaskLocked(Intent intent, ActivityInfo info) {
10. ComponentName cls = intent.getComponent();
11. if (info.targetActivity != null) {
12. cls = new ComponentName(info.packageName, info.targetActivity);
13. }
15. TaskRecord cp = null;
17. final int N = mHistory.size();
18. for (int i=(N-1); i>=0; i--) {
19. ActivityRecord r = (ActivityRecord)mHistory.get(i);
20. if (!r.finishing && r.task != cp
21. && r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {
22. cp = r.task;
23. //Slog.i(TAG, "Comparing existing cls=" + r.task.intent.getComponent().flattenToShortString()
24. //        + "/aff=" + r.task.affinity + " to new cls="
25. //        + intent.getComponent().flattenToShortString() + "/aff=" + taskAffinity);
26. if (r.task.affinity != null) {
27. if (r.task.affinity.equals(info.taskAffinity)) {
28. //Slog.i(TAG, "Found matching affinity!");
29. return r;
30. }
31. } else if (r.task.intent != null
32. && r.task.intent.getComponent().equals(cls)) {
33. //Slog.i(TAG, "Found matching class!");
34. //dump();
35. //Slog.i(TAG, "For Intent " + intent + " bringing to top: " + r.intent);
36. return r;
37. } else if (r.task.affinityIntent != null
38. && r.task.affinityIntent.getComponent().equals(cls)) {
39. //Slog.i(TAG, "Found matching class!");
40. //dump();
41. //Slog.i(TAG, "For Intent " + intent + " bringing to top: " + r.intent);
42. return r;
43. }
44. }
45. }
47. return null;
48. }
50. ......
52. }

public class ActivityStack {

......

/\*\*

\* Returns the top activity in any existing task matching the given

\* Intent. Returns null if no such task is found.

\*/

private ActivityRecord findTaskLocked(Intent intent, ActivityInfo info) {

ComponentName cls = intent.getComponent();

if (info.targetActivity != null) {

cls = new ComponentName(info.packageName, info.targetActivity);

}

TaskRecord cp = null;

final int N = mHistory.size();

for (int i=(N-1); i>=0; i--) {

ActivityRecord r = (ActivityRecord)mHistory.get(i);

if (!r.finishing && r.task != cp

&& r.launchMode != ActivityInfo.LAUNCH\_SINGLE\_INSTANCE) {

cp = r.task;

//Slog.i(TAG, "Comparing existing cls=" + r.task.intent.getComponent().flattenToShortString()

// + "/aff=" + r.task.affinity + " to new cls="

// + intent.getComponent().flattenToShortString() + "/aff=" + taskAffinity);

if (r.task.affinity != null) {

if (r.task.affinity.equals(info.taskAffinity)) {

//Slog.i(TAG, "Found matching affinity!");

return r;

}

} else if (r.task.intent != null

&& r.task.intent.getComponent().equals(cls)) {

//Slog.i(TAG, "Found matching class!");

//dump();

//Slog.i(TAG, "For Intent " + intent + " bringing to top: " + r.intent);

return r;

} else if (r.task.affinityIntent != null

&& r.task.affinityIntent.getComponent().equals(cls)) {

//Slog.i(TAG, "Found matching class!");

//dump();

//Slog.i(TAG, "For Intent " + intent + " bringing to top: " + r.intent);

return r;

}

}

}

return null;

}

......

}

        这个函数无非就是根据即将要启动的SubActivity的taskAffinity属性值在系统中查找这样的一个Task：Task的affinity属性值与即将要启动的Activity的taskAffinity属性值一致。如果存在，就返回这个Task堆栈顶端的Activity回去。在上面的AndroidManifest.xml文件中，没有配置MainActivity和SubActivity的taskAffinity属性，于是它们的taskAffinity属性值就默认为父标签application的taskAffinity属性值，这里，标签application的taskAffinity也没有配置，于是它们就默认为包名，即"shy.luo.task"。由于在启动SubActivity之前，MainActivity已经启动，MainActivity启动的时候，会在一个新的任务里面启动，而这个新的任务的affinity属性就等于它的第一个Activity的taskAffinity属性值。于是，这个函数会动回表示MainActivity的ActivityRecord回去.

        回到前面的startActivityUncheckedLocked函数中，这里的taskTop就表示MainActivity，它不为null，于是继续往前执行。由于条件r.launchMode == ActivityInfo.LAUNCH\_SINGLE\_TASK成立，于是执行下面语句：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. ActivityRecord top = performClearTaskLocked(
2. kTop.task.taskId, r, launchFlags, true);

ActivityRecord top = performClearTaskLocked(

taskTop.task.taskId, r, launchFlags, true);

        函数performClearTaskLocked也是定义在frameworks/base/services/java/com/android/server/am/ActivityStack.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. public class ActivityStack {
3. ......
5. /\*\*
6. \* Perform clear operation as requested by
7. \* {@link Intent#FLAG\_ACTIVITY\_CLEAR\_TOP}: search from the top of the
8. \* stack to the given task, then look for
9. \* an instance of that activity in the stack and, if found, finish all
10. \* activities on top of it and return the instance.
11. \*
12. \* @param newR Description of the new activity being started.
13. \* @return Returns the old activity that should be continue to be used,
14. \* or null if none was found.
15. \*/
16. private final ActivityRecord performClearTaskLocked(int taskId,
17. ActivityRecord newR, int launchFlags, boolean doClear) {
18. int i = mHistory.size();
20. // First find the requested task.
21. while (i > 0) {
22. i--;
23. ActivityRecord r = (ActivityRecord)mHistory.get(i);
24. if (r.task.taskId == taskId) {
25. i++;
26. break;
27. }
28. }
30. // Now clear it.
31. while (i > 0) {
32. i--;
33. ActivityRecord r = (ActivityRecord)mHistory.get(i);
34. if (r.finishing) {
35. continue;
36. }
37. if (r.task.taskId != taskId) {
38. return null;
39. }
40. if (r.realActivity.equals(newR.realActivity)) {
41. // Here it is!  Now finish everything in front...
42. ActivityRecord ret = r;
43. if (doClear) {
44. while (i < (mHistory.size()-1)) {
45. i++;
46. r = (ActivityRecord)mHistory.get(i);
47. if (r.finishing) {
48. continue;
49. }
50. if (finishActivityLocked(r, i, Activity.RESULT\_CANCELED,
51. null, "clear")) {
52. i--;
53. }
54. }
55. }
57. // Finally, if this is a normal launch mode (that is, not
58. // expecting onNewIntent()), then we will finish the current
59. // instance of the activity so a new fresh one can be started.
60. if (ret.launchMode == ActivityInfo.LAUNCH\_MULTIPLE
61. && (launchFlags&Intent.FLAG\_ACTIVITY\_SINGLE\_TOP) == 0) {
62. if (!ret.finishing) {
63. int index = indexOfTokenLocked(ret);
64. if (index >= 0) {
65. finishActivityLocked(ret, index, Activity.RESULT\_CANCELED,
66. null, "clear");
67. }
68. return null;
69. }
70. }
72. return ret;
73. }
74. }
76. return null;
77. }
79. ......
81. }

public class ActivityStack {

......

/\*\*

\* Perform clear operation as requested by

\* {@link Intent#FLAG\_ACTIVITY\_CLEAR\_TOP}: search from the top of the

\* stack to the given task, then look for

\* an instance of that activity in the stack and, if found, finish all

\* activities on top of it and return the instance.

\*

\* @param newR Description of the new activity being started.

\* @return Returns the old activity that should be continue to be used,

\* or null if none was found.

\*/

private final ActivityRecord performClearTaskLocked(int taskId,

ActivityRecord newR, int launchFlags, boolean doClear) {

int i = mHistory.size();

// First find the requested task.

while (i > 0) {

i--;

ActivityRecord r = (ActivityRecord)mHistory.get(i);

if (r.task.taskId == taskId) {

i++;

break;

}

}

// Now clear it.

while (i > 0) {

i--;

ActivityRecord r = (ActivityRecord)mHistory.get(i);

if (r.finishing) {

continue;

}

if (r.task.taskId != taskId) {

return null;

}

if (r.realActivity.equals(newR.realActivity)) {

// Here it is! Now finish everything in front...

ActivityRecord ret = r;

if (doClear) {

while (i < (mHistory.size()-1)) {

i++;

r = (ActivityRecord)mHistory.get(i);

if (r.finishing) {

continue;

}

if (finishActivityLocked(r, i, Activity.RESULT\_CANCELED,

null, "clear")) {

i--;

}

}

}

// Finally, if this is a normal launch mode (that is, not

// expecting onNewIntent()), then we will finish the current

// instance of the activity so a new fresh one can be started.

if (ret.launchMode == ActivityInfo.LAUNCH\_MULTIPLE

&& (launchFlags&Intent.FLAG\_ACTIVITY\_SINGLE\_TOP) == 0) {

if (!ret.finishing) {

int index = indexOfTokenLocked(ret);

if (index >= 0) {

finishActivityLocked(ret, index, Activity.RESULT\_CANCELED,

null, "clear");

}

return null;

}

}

return ret;

}

}

return null;

}

......

}

        这个函数中作用无非就是找到ID等于参数taskId的任务，然后在这个任务中查找是否已经存在即将要启动的Activity的实例，如果存在，就会把这个Actvity实例上面直到任务堆栈顶端的Activity通过调用finishActivityLocked函数将它们结束掉。在这个例子中，就是要在属性值affinity等于"shy.luo.task"的任务中看看是否存在SubActivity类型的实例，如果有，就把它上面的Activity都结束掉。这里，属性值affinity等于"shy.luo.task"的任务只有一个MainActivity，而且它不是SubActivity的实例，所以这个函数就返回null了。

        回到前面的startActivityUncheckedLocked函数中，这里的变量top就为null了，于是执行下面的else语句：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. if (top != null) {
2. ......
3. } else {
4. // A special case: we need to
5. // start the activity because it is not currently
6. // running, and the caller has asked to clear the
7. // current task to have this activity at the top.
8. addingToTask = true;
9. // Now pretend like this activity is being started
10. // by the top of its task, so it is put in the
11. // right place.
12. sourceRecord = taskTop;
13. }

if (top != null) {

......

} else {

// A special case: we need to

// start the activity because it is not currently

// running, and the caller has asked to clear the

// current task to have this activity at the top.

addingToTask = true;

// Now pretend like this activity is being started

// by the top of its task, so it is put in the

// right place.

sourceRecord = taskTop;

}

        于是，变量addingToTask值就为true了，同时将变量sourceRecord的值设置为taskTop，即前面调用findTaskLocked函数的返回值，这里，它就是表示MainActivity了。

        继续往下看，下面这个if语句：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. if (r.packageName != null) {
2. // If the activity being launched is the same as the one currently
3. // at the top, then we need to check if it should only be launched
4. // once.
5. ActivityRecord top = topRunningNonDelayedActivityLocked(notTop);
6. if (top != null && r.resultTo == null) {
7. if (top.realActivity.equals(r.realActivity)) {
8. if (top.app != null && top.app.thread != null) {
9. ......
10. }
11. }
12. }
14. } else {
15. ......
16. }

if (r.packageName != null) {

// If the activity being launched is the same as the one currently

// at the top, then we need to check if it should only be launched

// once.

ActivityRecord top = topRunningNonDelayedActivityLocked(notTop);

if (top != null && r.resultTo == null) {

if (top.realActivity.equals(r.realActivity)) {

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

......

}

}

}

} else {

......

}

        它是例行性地检查当前任务顶端的Activity，是否是即将启动的Activity的实例，如果是否的话，在某些情况下，它什么也不做，就结束这个函数调用了。这里，当前任务顶端的Activity为MainActivity，它不是SubActivity实例，于是继续往下执行：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. boolean newTask = false;
3. // Should this be considered a new task?
4. if (r.resultTo == null && !addingToTask
5. && (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {
6. ......
8. } else if (sourceRecord != null) {
9. if (!addingToTask &&
10. (launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0) {
11. ......
12. } else if (!addingToTask &&
13. (launchFlags&Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT) != 0) {
14. ......
15. }
16. // An existing activity is starting this new activity, so we want
17. // to keep the new one in the same task as the one that is starting
18. // it.
19. r.task = sourceRecord.task;
21. ......
23. } else {
24. ......
25. }

boolean newTask = false;

// Should this be considered a new task?

if (r.resultTo == null && !addingToTask

&& (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {

......

} else if (sourceRecord != null) {

if (!addingToTask &&

(launchFlags&Intent.FLAG\_ACTIVITY\_CLEAR\_TOP) != 0) {

......

} else if (!addingToTask &&

(launchFlags&Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT) != 0) {

......

}

// An existing activity is starting this new activity, so we want

// to keep the new one in the same task as the one that is starting

// it.

r.task = sourceRecord.task;

......

} else {

......

}

        这里首先将newTask变量初始化为false，表示不要在新的任务中启动这个SubActivity。由于前面的已经把addingToTask设置为true，因此，这里会执行中间的else if语句，即这里会把r.task设置为sourceRecord.task，即把SubActivity放在MainActivity所在的任务中启动。

        最后，就是调用startActivityLocked函数继续进行启动Activity的操作了。后面的操作这里就不跟下去了，有兴趣的读者可以参考两篇文章[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)。

        到这里，思路就理清了，虽然SubActivity的launchMode被设置为"singleTask"模式，但是它并不像官方文档描述的一样：The system creates a new task and instantiates the activity at the root of the new task，而是在跟它有相同taskAffinity的任务中启动，并且位于这个任务的堆栈顶端，于是，前面那个图中，就会出现一个带着"singleTask"标签的箭头指向一个任务堆栈顶端的Activity Y了。  
        那么，我们有没有办法让一个"singleTask"的Activity在新的任务中启动呢？答案是肯定的。从上面的代码分析中，只要我们能够进入函数startActivityUncheckedLocked的这个if语句中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. if (r.resultTo == null && !addingToTask
2. && (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {
3. // todo: should do better management of integers.
4. mService.mCurTask++;
5. if (mService.mCurTask <= 0) {
6. mService.mCurTask = 1;
7. }
8. r.task = new TaskRecord(mService.mCurTask, r.info, intent,
9. (r.info.flags&ActivityInfo.FLAG\_CLEAR\_TASK\_ON\_LAUNCH) != 0);
10. if (DEBUG\_TASKS) Slog.v(TAG, "Starting new activity " + r
11. + " in new task " + r.task);
12. newTask = true;
13. if (mMainStack) {
14. mService.addRecentTaskLocked(r.task);
15. }
16. }

if (r.resultTo == null && !addingToTask

&& (launchFlags&Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0) {

// todo: should do better management of integers.

mService.mCurTask++;

if (mService.mCurTask <= 0) {

mService.mCurTask = 1;

}

r.task = new TaskRecord(mService.mCurTask, r.info, intent,

(r.info.flags&ActivityInfo.FLAG\_CLEAR\_TASK\_ON\_LAUNCH) != 0);

if (DEBUG\_TASKS) Slog.v(TAG, "Starting new activity " + r

+ " in new task " + r.task);

newTask = true;

if (mMainStack) {

mService.addRecentTaskLocked(r.task);

}

}

        那么，这个即将要启动的Activity就会在新的任务中启动了。进入这个if语句需要满足三个条件，r.resultTo为null，launchFlags的Intent.FLAG\_ACTIVITY\_NEW\_TASK位为1，并且addingToTask值为false。从上面的分析中可以看到，当即将要启动的Activity的launchMode为"singleTask"，并且调用startActivity时不要求返回要启动的Activity的执行结果时，前面两个条件可以满足，要满足第三个条件，只要当前系统不存在affinity属性值等于即将要启动的Activity的taskAffinity属性值的任务就可以了。

        我们可以稍微修改一下上面的AndroidManifest.xml配置文件来做一下这个实验：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

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:taskAffinity="shy.luo.task.main.activity">
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:launchMode="singleTask"
18. android:taskAffinity="shy.luo.task.sub.activity">
19. <intent-filter>
20. <action android:name="shy.luo.task.subactivity"/>
21. <category android:name="android.intent.category.DEFAULT"/>
22. </intent-filter>
23. </activity>
24. </application>
25. </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:taskAffinity="shy.luo.task.main.activity">

<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:launchMode="singleTask"

android:taskAffinity="shy.luo.task.sub.activity">

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

</manifest>

        注意，这里我们设置MainActivity的taskAffinity属性值为"shy.luo.task.main.activity"，设置SubActivity的taskAffinity属性值为"shy.luo.task.sub.activity"。重新编译一下程序，在模拟器上把这个应用程序再次跑起来，用“adb shell dumpsys activity”命令再来查看一下系统运行的的任务，就会看到：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6714543)

1. Running activities (most recent first):
2. TaskRecord{4069c020 #4 A shy.luo.task.sub.activity}
3. Run #2: HistoryRecord{40725040 shy.luo.task/.SubActivity}
4. TaskRecord{40695220 #3 A shy.luo.task.main.activity}
5. Run #1: HistoryRecord{406b26b8 shy.luo.task/.MainActivity}
6. TaskRecord{40599c90 #2 A com.android.launcher}
7. Run #0: HistoryRecord{40646628 com.android.launcher/com.android.launcher2.Launcher}

Running activities (most recent first):

TaskRecord{4069c020 #4 A shy.luo.task.sub.activity}

Run #2: HistoryRecord{40725040 shy.luo.task/.SubActivity}

TaskRecord{40695220 #3 A shy.luo.task.main.activity}

Run #1: HistoryRecord{406b26b8 shy.luo.task/.MainActivity}

TaskRecord{40599c90 #2 A com.android.launcher}

Run #0: HistoryRecord{40646628 com.android.launcher/com.android.launcher2.Launcher}

        这里就可以看到，SubActivity和MainActivity就分别运行在不同的任务中了。

        至此，我们总结一下，设置了"singleTask"启动模式的Activity的特点：

        1. 设置了"singleTask"启动模式的Activity，它在启动的时候，会先在系统中查找属性值affinity等于它的属性值taskAffinity的任务存在；如果存在这样的任务，它就会在这个任务中启动，否则就会在新任务中启动。因此，如果我们想要设置了"singleTask"启动模式的Activity在新的任务中启动，就要为它设置一个独立的taskAffinity属性值。

        2. 如果设置了"singleTask"启动模式的Activity不是在新的任务中启动时，它会在已有的任务中查看是否已经存在相应的Activity实例，如果存在，就会把位于这个Activity实例上面的Activity全部结束掉，即最终这个Activity实例会位于任务的堆栈顶端中。

        看来，要解开Activity的"singleTask"之谜，还是要自力更生啊，不过，如果我们仔细阅读官方文档，在<http://developer.android.com/guide/topics/manifest/activity-element.html>中，有这样的描述：

        As shown in the table above, standard is the default mode and is appropriate for most types of activities. SingleTop is also a common and useful launch mode for many types of activities. The other modes — singleTask and singleInstance —**are not appropriate for most applications**, since they result in an interaction model that is likely to be unfamiliar to users and is very different from most other applications.  
        Regardless of the launch mode that you choose, make sure to test the usability of the activity during launch and when navigating back to it from other activities and tasks using the BACK key.

       这样看，官方文档也没有坑我们呢，它告诫我们：**make sure to test the usability of the activity during launch**。