Android Coding Assessment Test Prepare

deepwaterooo

August 12, 2022

Contents

1 启动模式与任务栈 **1**

2 standard：标准模式 **1**

3 singleTop：栈顶复用模式 **1**

4 singleTask：栈内复用模式 **2**

5 singleInstance：单实例模式 **3**

6 综合比较与汇总 **5**

7 设定方法 **5**
7.1 manifest.xml 文件中设置。 . . . 5
7.2 Intent 设置标记位 . . . 5
7.2.1 FLAG_ACTIVITY_NEW_TASK: . . . 5
7.2.2 FLAG_ACTIVITY_SINGLE_TOP: . . . 6
7.2.3 FLAG_ACTIVITY_CLEAR_TOP: . . . 6
7.2.4 FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS: . . . 6
7.2.5 Intent.FLAG_ACTIVITY_NO_HISTORY . . . 6
7.2.6 FLAG_ACTIVITY_CLEAR_TASK . . . 6
7.2.7 注意事项: . . . 6
7.3 Intent flag 标记位进阶:（这个难度比较高一点儿，改天脑袋清醒的时候再好好理解消化一下) . . . 7
7.3.1 FLAG_ACTIVITY_CLEAR_TOP . . . 7
7.3.2 FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET . . . 7
7.3.3 FLAG_ACTIVITY_RESET_TASK_IF_NEEDED . . . 8
7.3.4 FLAG_ACTIVITY_NEW_TASK . . . 8
7.3.5 FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS . . . 8
7.3.6 FLAG_ACTIVITY_FORWARD_RESULT . . . 8

8 启动模式与 startActivityForResult **9**
8.1 LaunchMode 与 StartActivityForResult . . . 9
8.2 启动任务（Task)：这里要再消化一下! . . . 10
8.3 Activity Stack . . . 10

9 onNewIntent **11**

10 Activity 所需的任务栈与 TaskAffinity **11**
10.1 TaskAffinity 和 singleTask 启动模式结合使用 . . . 11
10.2 当 TaskAffinity 和 allowTaskReparenting 结合使用 . . . 12
10.2.1 allowTaskReparenting 属性 . . . 12
10.2.2 注意点 . . . 13
10.2.3 taskAffinity 在两种情况下起作用： . . . 14
10.3 清空任务栈 . . . 14
10.3.1 android:alwaysRetainTaskState . . . 14
10.3.2 clearTaskOnLaunch . . . 15
10.3.3 android:finishOnTaskLaunch . . . 15

11 启动模式的应用场景 **15**
11.1 SingleTask 模式的运用场景 . . . 15
11.2 SingleTop 模式的运用场景 . . . 15
11.3 SingleInstance 模式的运用场景 . . . 16
11.3.1 第一种情况 . . . 16
11.3.2 第二种情况 . . . 16
11.3.3 第三种情况 . . . 16
11.3.4 总结 . . . 16
11.4 standard 运用场景 . . . 16

12 Activity 时的生命周期不同 **16**

13 实际中的栈管理类 **17**

14 AndroidManifest 配置相关参数的说明详解 **18**
14.1 android:allowEmbedded . . . 18
14.2 android:stateNotNeeded="true" 的作用 . . . 18

15 ActivityStack **18**

16 安卓如何管理多个任务 (没被称谓多个任务栈，虽然本质上仍是多个任务栈，每个任务都是一个栈结构) **18**
16.1 Android 多任务运行机制 . . . 18
16.2 定义启动模式 . . . 19

1 启动模式与任务栈

- 安卓有四种启动模式：standard、singleTop、singleTask 和 singleInstance，想要更改模式可以在 AndroidManifest.xml 中 activity 标签下添加 launchMode 标签。下面是各种模式的详细介绍 (下文中所有的栈，均指任务栈)：
  - Activity 所需的任务栈?
    - 这要从一个参数说起：TaskAffinity(任务相关性)。这个参数标识了一个 activity 所需的任务栈的名字，默认情况下，所有 activity 所需的任务栈的名字为应用的包名。
    - TaskAffinity 参数标识着 Activity 所需要的任务栈的名称，默认情况下，一个应用中所有 Activity 所需要的任务栈名称都为该应用的包名。
    - TaskAffinity 属性一般跟 singleTask 模式或者跟 allowTaskReparenting 属性结合使用，在其他情况下没有实际意义。

2 standard：标准模式

- 这也是系统的默认模式。每次启动一个 Activity 都会重新创建一个新的实例，不管实例是否已经存在。被创建的实例的生命周期符合典型的 Activity 的生命周期。在这种模式下，谁启动了这个 Activity，那么这个 Activity 就运行在启动它的那个 Activity 所在的栈中。比如 ActivityA 启动了 ActivityB（B 也是 standard 模式），那么 B 就会进入到 A 所在的栈中。
  - 如果我们用 ApplicationContext 去启动 standard 模式的时候 Activity 的时候会报错，错误的原因是因为 standard 模式的 Activity 默认会进入启动它的 Activity 所属的任务栈中，但是由于非 Activity 类型的 Context（如 ApplicationContext）并没有所谓的任务栈，所有这就有问题了。解决这个问题的方法是为待启动 Activity 指定 FLAG_ACTIVITY_NEW_TASK 标记位，这样启动的时候就会为它创建一个新的任务栈，这时候待启动 Activity 实际上是一 singleTask 模式启动的。后文会再次强调一下

3 singleTop：栈顶复用模式

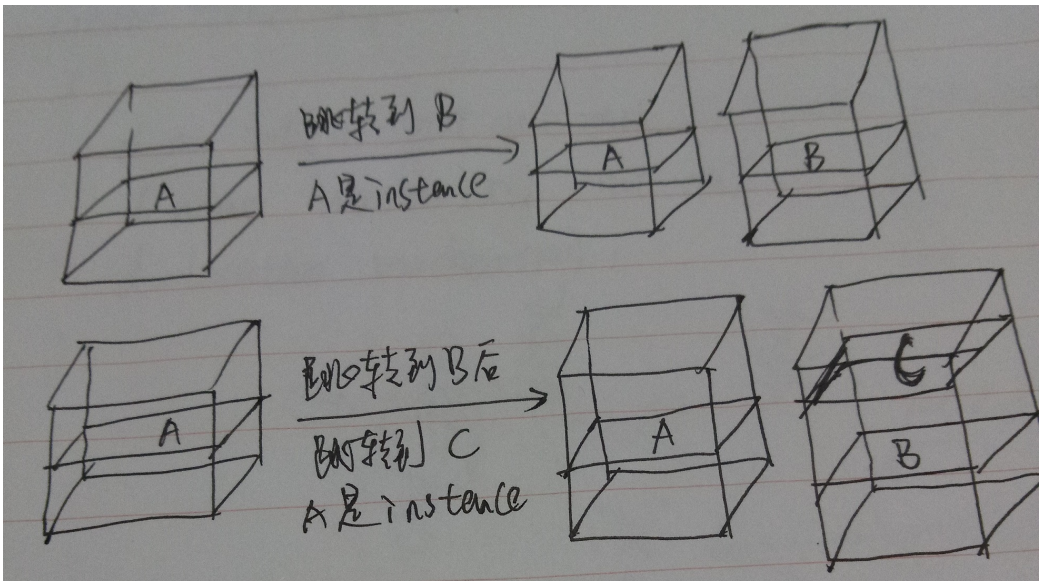
- 在这种模式下，如果新 Activity 已经位于任务栈的栈顶（处于完全可见状态），那么此 Activity 不会被重新创建。但是如果新的 Activity 不是位于栈顶（处于不完全可见状态），那么新 Activity 仍然会重新创建。在 singleTop（栈顶复用模式）下，如果 Activity 位于栈顶，我们再次启动该方法，那么该方法会回调 onNewIntent() 方法 (不会创建新的 instance)，而 onCreate、onStart 方法不会被调用。
- 也允许出现重复的活动实例，分为两种情况，页面不在栈顶和页面已经在栈顶的情况。
  - （1）页面不在栈顶。此时还是和 standard 一样重新生成一个新的 activity 实例。
  - （2）页面在栈顶。此时直接使用栈顶的这个活动实例，不会重新生成新的。
- 如果当前任务的顶部已存在 Activity 的一个实例，则系统会通过调用该实例的 onNewIntent() 方法向其传送 Intent，通过此方法的参数我们可以取出当前请求的信息；而不是创建 Activity 的新实例。需要注意的是，这个 Activity 的 onCreate 、onStart 不会被系统调用，因为它并没有发生改变。
- **Activity 可以多次实例化，而每个实例均可属于不同的任务，并且一个任务可以拥有多个实例；**这句话是正确的，但**前提**是位于返回栈顶部的 Activity 并不是 Activity 的现有实例，如果是 Activity 的现有实例，而且启动模式是 singleTop，则不会多次实例化。
- 例如，假设任务的返回栈包含根 Activity A 以及 Activity B、C 和位于顶部的 D（堆栈是 A-B-C-D；D 位于顶部）。收到针对 D 类 Activity 的 Intent。如果 D 具有默认的"standard" 启动模式，则会启动该类的新实例，且堆栈会变成 A-B-C-D-D。但是，如果 D 的启动模式是"singleTop"，则 D 的现有实例会通过 onNewIntent() 接收 Intent，因为它位于堆栈的顶部；而堆栈仍为 A-B-C-D。但是，**如果收到针对 B 类 Activity 的 Intent，则会向堆栈添加 B 的新实例，即便其启动模式为"singleTop" 也是如此，也就是说，如果 B 没有位于栈顶，即便 B 的启动模式是"singleTop"，在收到针对 B 类 Activity 的 Intent 时，也会向堆栈添加 B 的新实例**。
- 注：**为某个 Activity 创建新实例时，用户可以按“返回”按钮返回到前一个 Activity。但是，当 Activity 的现有实例处理新 Intent 时，则在新 Intent 到达 onNewIntent() 之前，用户无法按“返回”按钮返回到 Activity 的状态**。
- 总结：这种模式通常适用于就收到消息后现实的界面，比如 QQ 接收到消息后弹出 Activity，如果一次来了 10 条消息，总不能一次弹 10 个 Activity。

4 singleTask：栈内复用模式

- 系统创建新任务并实例化位于新任务底部的 Activity。但是，如果该 Activity 的一个实例已存在于一个单独的任务中，则系统会通过调用现有实例的 onNewIntent() 方法向其传送 Intent，而不是创建新实例。一次只能存在 Activity 的一个实例。
- 注：**尽管 Activity 在新任务中启动，但是用户按“返回”按钮仍会返回到前一个 Activity**。
- singleTask 模式与 singlTop 模式类似，只不过 **singleTop** 是**检测栈顶元素是否是需要启动的 Activity** ，而 **singleTask** 是**检测整个 Activity 栈中是否存在当前需要启动的 Activity**。如果存在，则将该 Activity 置于栈顶，并将该 Activity 以上的 Activity 都销毁（由于 singleTask 默认具有 clearTop 的效果）。
- 不过上面的这一条，是指**在同一个 App 中启动这个 singleTask 的 Activity**，如果是**其他程序以 singleTask 模式来启动这个 Activity** ，那么它将创建一个新的任务栈。不过这里有一点需要注意的是，如果启动的模式为 singleTask 的 Activity 已经在后台一个任务栈中了，那么启动后，而后台的这个任务栈将一起被切换到到前台。
- 举例如下：
  - 比如目前任务栈 S1 中的情况为 ABC，这个时候 Activity D 以 singleTask 模式请求启动，其所需要的任务栈为 S2 ，由于 S2 和 D 的实例均不存在，所以系统会先创建任务栈 S2 ，然后再创建 D 的实例并将其入栈到 S2 。
  - 另外一种情况，假设 D 所需的任务栈为 S1，其他情况如上面例子 l 所示，那么由于 S1 已经存在，所以系统会直接创建 D 的实例并将其入栈到 S1。
  - 如果 D 所需的任务栈为 S1，并且当前任务栈 S1 的情况为 ADBC ，根据栈内复用的原则，此时 D 不会重新创建，系统会把 D 切换到栈顶并调用其 onNewlntent 方法，同时由于 singleTask 默认具有 clearTop 的效果，会导致栈内所有在 D 上面的 Activity 全部出栈，于是最终 S1 中的情况为 AD。这一点比较特殊。
- 总结：使用这个模式创建的 Activity 不是在新的任务栈中被打开，就是将已经打开的 Activity 切换到前台，所以**这种启动模式通常可以用来退出整个应用：将主 Activity 设为 singleTask 模式，然后在要退出的 Activity 中转到主 Activity，从而将主 Activity 之上的 Activity 都清除，然后重写主 Activity 的 onNewIntent() 方法，在方法中加上一行代码 finish()，将最后一个 Activity 结束掉**。
- 这是一种单实例模式，在这种模式下，一个 Activity 在一个栈中只能有一个实例，类似单例模式。详细讲解，当一个具有 singleTask 模式的 Activity 请求启动后，例如 Activity A，系统首先会寻找是否存在 A 想要的任务栈，如果不存在，就重新创建一个新的任务栈，然后创建 A 的实例后把 A 放到栈中。如果存在 A 所需要的任务栈，再看 Activity A 是否在栈中有实例存在，如果有实例存在，那么系统就会把 A 调到栈顶并调用 onNewIntent 方法，如果实例不存在，就创建 A 的实例并把 A 压入栈中。
  - 小例子：活动启动的顺序是 ABCDB，第一次启动 B 的时候栈内没有实例，系统会创建新的 B 的实例，但第二次启动 B 的时候，系统查询任务栈，发现有实例，系统就会把活动 B 移到栈顶，并且把 B 之上的所有活动出栈。
- 不会出现重复的活动实例，这种模式下有两种情形，栈内有这个活动的实例的情形和栈内没有这个实例的情形。
  - （1）假设栈内原来有 A、B 两个实例，此时跳转到 A 页面，不管 A 页面位不位于栈顶，只要栈内存在 A 活动的实例，那么就把 A 以上的实例全部销毁出栈，总之让 A 位于栈顶得到用户观看。
  - （2）假设栈内没有 C，此时跳转到 C 页面，会创建新的 C 活动实例。

5 singleInstance：单实例模式

- 这是一种加强的 singleTask 模式，它除了具有 singleTask 模式所具有的所有特性外，还加强一点，那就是具有此种模式的 Activity 只能单独的位于一个任务栈中 (具有独占性：独占一个任务栈)。简单而言，如果一个活动是 singleInstance 的启动模式，那么该活动就只能单独位于一个栈中。
  - 小例子：如果一个活动是 singleInstance 模式，那么活动 C 会单独创建一个新的任务栈，而返回栈（应该是安卓多任务管理中的多个任务时，不同任务的前后返回顺序）中，活动 C 处于的任务栈会先压入返回栈的栈低，再把另外一个活动栈放入返回栈中（这句话仍然是没有看懂，感觉是写错了，先压 B 任务栈，再压入新开启的任务 C，下图中的 ABC 更像是以 singleInstance 启动的三个活动，分别独占一个各自的任务栈）。
- 不会出现重复的活动实例，此时比较特殊，持有这种模式的活动实例单独有一个栈来存储它，栈内只有它一个实例，如果多个应用启动这个活动，那么他们共同享用这个栈内的唯一实例，不会生成新的。这个模式的应用比如说**手机的锁屏页面**。



- 这种模式的返回模式，出栈顺序是 C-B-A, 入栈顺序是 A-B-C, 最先出现，最后死亡
- 下面是另一种的表达方式，但是有引导思考的地方，也需要去学习这种**通过对比加强理解的思维模式**。
- 与"singleTask" 相同, 只是系统不会将任何其他 Activity 启动到包含实例的任务中。该 Activity 始终是其任务唯一仅有的成员；由此 Activity 启动的任何 Activity 均在单独的任务中打开。而且**代码中不能实现相同的启动模式效果**。
- 这是一种加强的 singleTask 模式，它除了具有 singleTask 模式的所有特性外，还加强了一点，那就是具有此种模式的 Activity 只能单独地位于一个任务栈中，换句话说，比如 Activity A 是 singleInstance 模式，当 A 启动后，系统会为它创建一个新的任务栈，然后 A 独自在这个新的任务找中，由于栈内复用的特性，后续的请求均不会创建新的 Activity，除非这个独特的任务栈被系统销毁了。
- singleInstance 这种启动模式和使用的浏览器工作原理类似。在多个程序中访问浏览器时, 如果当前浏览器没有打开，则打开浏览器，否则会在当前打开的浏览器中访问。申明为 singleInstance 的 Activity 会出现在一个新的任务栈中，而该任务栈中只存在这一个

Activity 。举个例子来说，如果应用 A 的任务栈中创建了 MainActivity 实例，且启动模式为 singleInstance，如果应用 B 也要激活 MainActivity，则不需要创建，两个应用共享该 Activity 实例。这种启动模式常用于需要与程序分离的界面，如在 SetupWizard 中调用紧急呼叫，就是使用这种启动模式。

- 比较一下：不同于以上 3 种启动模式，**指定为 singleInstance 模式的活动会启用一个新的返回找来管理这个活动**（其实**如果 singleTask 模式指定了不同的 taskAftinity ，也会启动一个新的返回栈**）。那么**这样做有什么意义呢？**
- **特别适用的场景**：想象以下场景，假设我们的**程序中有一个活动是允许其他程序调用的，如果我们想实现其他一个或是多个不同的应用程序和我们的程序可以共享这个活动的实例，应该如何实现呢？**
- 使用前面 3 种启动模式肯定是做不到的，因为每个应用程序都会有自己的返回栈，同一个活动在不同的返回栈中入栈时必然是创建了新的实例。而使用 singleInstance 模式就可以解决这个问题，在这种模式下会有一个单独的返回栈来管理这个活动，不管是哪个应用程序来访问这个活动，都共用的同一个返回栈，也就解决了共享活动实例的问题。（具体事例可参考郭霖的第一行代码中例子）

6 综合比较与汇总

launchMode	简述	详解	主要用途
standard	标准模式、默认模式 android:launchMode ="standard" 或不指定属性	Standard 是默认启动模式，没有指定 android:launchMode 属性时默认是 standard。在启动 activity 时不管栈中是何种状态，都会新生成一个 activity 压入栈中，可以存在多个连续相同的 activity。	最常用的启动模式，适用于没有特殊要求的启动模式。
singleTop	栈顶复用 android:launchMode ="singleTop"	singleTop 模式下启动 activity 时会匹配当前栈顶 activity 与要启动的 activity 是否相同，相同则调用 onNewIntent()方法。如果不同则会 new 一个 activity 并压入栈中。	适合用于接受通知推送启动的界面，如新闻客户端收到 7 条新闻推送，则尽可能复用 activity 去显示推送，如果打开 7 个 activity 是很耗内存和低性能的方法。
singleTask	栈内复用 android:launchMode ="singleTask"	singleTask 模式会确认当前任务栈中是否存在即将新建 activity 的实例，如果存在，则调用其 onNewIntent()方法，并调用 clearTop 方法清理其上面的 activity。如果不存在则会新建一个 task 栈，并把新 new 出的 activity 压入新生成的任务栈中	适合用作程序主入口，比如浏览器主界面，不管多少个应用启动浏览器都只会启动一个主界面，其余情况都使用 onNewIntent，并清空其上的其它界面。
singleInstance	单实例模式 android:launchMode ="singleInstance"	singleInstance 模式不管任务栈中是否存在该 activity，都会新建一个任务栈存放 activity。每个任务栈中只有一个 activity，每个 activity 占用一个任务栈。	在需要系统中只存在一个该 activity 实例时使用，可以保证 activity 的全局唯一性。

7 设定方法

- 怎么设定这四种模式，有两种方法，

7.1 manifest.xml 文件中设置。

```
<activity android:name=".Activity1"
          android:launchMode="standard"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

7.2 Intent 设置标记位

- Intent 设置标记位方式的优先级高于 manifest 中指定 launchMode 的方式.

7.2.1 FLAG_ACTIVITY_NEW_TASK:

- 效果和在 manifest 中设置 launchMode 为 singleTask 相同。
- 该标志位表示使用一个新的 Task 来启动一个 Activity，相当于在清单文件中给 Activity 指定“singleTask”启动模式。通常我们在 Service 启动 Activity 时，由于 Service 中并没有 Activity 任务栈，所以必须使用该 Flag 来创建一个新的 Task。

7.2.2 FLAG_ACTIVITY_SINGLE_TOP:

- 这个 FLAG 就相当于加载模式中的 singletop，比如说原来栈中情况是 A,B,C,D 在 D 中启动 D，栈中的情况还是 A,B,C,D

7.2.3 FLAG_ACTIVITY_CLEAR_TOP:

- 具有此标记的 activity 启动时，在同一任务栈中所有位于它上面的 activity 都要出栈。一般和 FLAG_ACTIVITY_NEW_TASK 配合使用。这种情况下，被启动的 activity 的实例如果已经存在，那么会调用它的 onNewIntent 方法。
- 这个 FLAG 就相当于加载模式中的 SingleTask，这种 FLAG 启动的 Activity 会把要启动的 Activity 之上的 Activity 全部弹出栈空间。类如：原来栈中的情况是 A,B,C,D 这个时候从 D 中跳转到 B，这个时候栈中的情况就是 A,B 了.

7.2.4 FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS:

- 具有此标记的 activity 不会出现在历史 activity 列表中。
- 使用该标识位启动的 Activity 不添加到最近应用列表，也即我们从最近应用里面查看不到我们启动的这个 activity。
- 等同于在 manifest 中设置 activity 属性

```
<android:excludeFromRecents="true"/>
```

7.2.5 Intent.FLAG_ACTIVITY_NO_HISTORY

-使用该模式来启动 Activity，当该 Activity 启动其他 Activity 后，该 Activity 就被销毁了，不会保留在任务栈中。如 A-B,B 中以这种模式启动 C，C 再启动 D，则任务栈只有 ABD。

7.2.6 FLAG_ACTIVITY_CLEAR_TASK

- 如果在传递给 Context.startActivity() 的意图中设置了该标志，则会导致在启动 activity 之前清除与该 activity 关联的任何现有任务。也就是说，activity 成为一个空任务的新根，任何旧 activity 都 finish 了。
- 这只能与 FLAG_ACTIVITY_NEW_TASK 一起使用。

7.2.7 注意事项:

- 当通过非 activity 的 context 来启动一个 activity 时，需要增加 intent flag FLAG_ACTIVITY_NEW_TAS

```
Intent i = new Intent(this, Wakeup.class);
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

1. 对 Intent.FLAG_ACTIVITY_NEW_TASK 这个属性，是不是一定新开一个栈?
   - 这个问题的答案是：不一定
   - 假设现在有一个栈 1，里面是 A,B,C。此时，在 C 中启动 D 的时候，设置 FLAG_ACTIVITY_NEW_TA 标记，此时会有两种情况：
     - 1. 如果 D 这个 Activity 在 Manifest.xml 中的声明中添加了 Task Affinity，系统首先会查找有没有和 D 的 Task Affinity 相同的 Task 栈存在，如果有存在，将 D 压入那个栈
     - 2. 如果 D 这个 Activity 在 Manifest.xml 中的 Task Affinity 默认没有设置，则会把其压入栈 1，变成：A B C D，这样就和标准模式效果是一样的了。
   - 我想，这篇里的部分结论，很大一部分结论，还是需要小项目代码再验证一下其正确性的 不敢轻信、没有确信!!!
     - 也就是说，设置了这个标志后，新启动的 Activity 并非就一定在新的 Task 中创建，如果 A 和 B 在属于同一个 package，而且都是使用默认的 Task Affinity，那 B 还是会在 A 的 task 中被创建。所以，只有 A 和 B 的 Task Affinity 不同时，设置了这个标志才会使 B 被创建到新的 Task。
     - ！注意如果试图从非 Activity 的非正常途径启动一个 Activity，比如从一个 Receiver 中启动一个 Activity，则 Intent 必须要添加 FLAG_ACTIVITY_NEW_TASK 标记。
     - 我们这里之所以会新建一个栈，因为我们的 APP 和系统 Activity 的 Task Affinity 不同

7.3 Intent flag 标记位进阶：（这个难度比较高一点儿，改天脑袋清醒的时候再好好理解消化一下）

- https://blog.csdn.net/vshuang/article/details/66472338?spm=1001.2101.3001.6661.1&utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.no_search_link

7.3.1 FLAG_ACTIVITY_CLEAR_TOP

如果设置，并且这个 Activity 已经在当前的 Task 中运行，因此，不再是重新启动一个这个 Activity 的实例，而是在这个 Activity 上方的所有 Activity 都将关闭，然后这个 Intent 会作为一个新的 Intent 投递到老的 Activity（现在位于顶端）中。例如，假设一个 Task 中包含这些 Activity：A，B，C，D。如果 D 调用了 startActivity()，并且包含一个指向 Activity B 的 Intent，那么，C 和 D 都将结束，然后 B 接收到这个 Intent，因此，目前 stack 的状况是：A，B。上例中正在运行的

Activity B 既可以在 onNewIntent() 中接收到这个新的 Intent，也可以把自己关闭然后重新启动来接收这个 Intent。如果它的启动模式声明为 “multiple” (默认值)，并且你没有在这个 Intent 中设置 FLAG_ACTIVITY_SINGLE_TOP 标志，那么它将关闭然后重新创建；对于其它的启动模式，或者在这个 Intent 中设置 FLAG_ACTIVITY_SINGLE_TOP 标志，都将把这个 Intent 投递到当前这个实例的 onNewIntent() 中。这个启动模式还可以与 FLAG_ACTIVITY_NEW_TASK 结合起来使用：用于启动一个 Task 中的根 Activity，它会把那个 Task 中任何运行的实例带入前台，然后清除它直到根 Activity。这非常有用，例如，当从 Notification Manager 处启动一个 Activity

7.3.2 FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET

如果设置，这将在 Task 的 Activity stack 中设置一个还原点，当 Task 恢复时，需要清理 Activity。也就是说，下一次 Task 带着 FLAG_ACTIVITY_RESET_TASK_IF_NEEDED 标记进入前台时（典型的操作是用户在主画面重启它），这个 Activity 和它之上的都将关闭，以至于用户不能再返回到它们，但是可以回到之前的 Activity。这在你的程序有分割点的时候很有用。例如，一个 e-mail 应用程序可能有一个操作是查看一个附件，需要启动图片浏览 Activity 来显示。这个 Activity 应该作为 e-mail 应用程序 Task 的一部分，因为这是用户在这个 Task 中触发的操作。然而，当用户离开这个 Task，然后从主画面选择 e-mail app，我们可能希望回到查看的会话中，但不是查看图片附件，因为这让人困惑。通过在启动图片浏览时设定这个标志，浏览及其它启动的 Activity 在下次用户返回到 mail 程序时都将全部清除。

7.3.3 FLAG_ACTIVITY_RESET_TASK_IF_NEEDED

If set, and this activity is either being started in a new task or bringing to the top an existing task, then it will be launched as the front door of the task. This will result in the application of any affinities needed to have that task in the proper state (either moving activities to or from it), or simply resetting that task to its initial state if needed.

7.3.4 FLAG_ACTIVITY_NEW_TASK

如果设置，这个 Activity 会成为历史 stack 中一个新 Task 的开始。一个 Task（从启动它的 Activity 到下一个 Task 中的 Activity）定义了用户可以迁移的 Activity 原子组。Task 可以移动到前台和后台；在某个特定 Task 中的所有 Activity 总是保持相同的次序。这个标志一般用于呈现 “启动” 类型的行为：它们提供用户一系列可以单独完成的事情，与启动它们的 Activity 完全无关。使用这个标志，如果正在启动的 Activity 的 Task 已经在运行的话，那么，新的 Activity 将不会启动；代替的，当前 Task 会简单的移入前台。参考 FLAG_ACTIVITY_MULTIPLE_TASK 标志，可以禁用这一行为。这个标志不能用于调用方对已经启动的 Activity 请求结果。

7.3.5 FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS

- 如果设置，新的 Activity 不会在最近启动的 Activity 的列表中保存。
- 参考一个 stackoverflow 的问答 https://stackoverflow.com/questions/7759556/flag-activit
- I have a Notification which starts an Activity. After a long press on home button and selecting my app, I want to start my main Activity again, and not this Activity started by the Notification. I tried with FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS, but this removed my whole application from the recents, and that’s not what I want to achieve. How can I have my app in the recents, but have the main Activity started?
- Okay, I found the solution to my problem. I started an Activity from a Notification with FLAG_ACTIVITY_NEW_TASK. But it seems to me that this Activity only gets started in an own task if affinity is different from the default affinity. So I had to add a different affinity in the manifest.

- And it seems that FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS does not (as documented) exlucde the Activity from the recents, rather it excludes the whole task (not the whole application) in which the Activity gets started from the recents. And as I hadn't set a different affinity the Activity which I wanted to exclude was started in the same task (although I had set FLAG_ACTIVITY_NEW_TASK) and so my whole application (as it was running in only one task) was excluded from the recents.

- Now I've set a different affinity for the Activity that gets started from the Notification and I start it with FLAG_ACTIVITY_NEW_TASK | FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS. When I leave this Activity and long-press the HOME button I can choose my app and the default task is started or brought to the front.

7.3.6 FLAG_ACTIVITY_FORWARD_RESULT

- 如果设置，并且这个 Intent 用于从一个存在的 Activity 启动一个新的 Activity，那么，这个作为答复目标的 Activity 将会传到这个新的 Activity 中。这种方式下，新的 Activity 可以调用 setResult(int)，并且这个结果值将发送给那个作为答复目标的 Activity。

8 启动模式与 startActivityForResult

8.1 LaunchMode 与 StartActivityForResult

- 我们在开发过程中经常会用到 StartActivityForResult 方法启动一个 Activity，然后在 onActivityResult() 方法中可以接收到上个页面的回传值，但你有可能遇到过拿不到返回值的情况，那有可能是因为 Activity 的 LaunchMode 设置为了 singleTask。5.0 之后，android 的 LaunchMode 与 StartActivityForResult 的关系发生了一些改变。两个 Activity，A 和 B，现在由 A 页面跳转到 B 页面，看一下 LaunchMode 与 StartActivityForResult 之间的关系：

- 这是为什么呢?

- 这是因为 ActivityStackSupervisor 类中的 startActivityUncheckedLocked 方法在 5.0 中进行了修改。

- 在 5.0 之前，当启动一个 Activity 时，系统将首先检查 Activity 的 launchMode，如果为 A 页面设置为 SingleInstance 或者 B 页面设置为 singleTask 或者 singleInstance，则会在 LaunchFlags 中加入 FLAG_ACTIVITY_NEW_TASK 标志，而如果含有 FLAG_ACTIVITY_NEW_TASK 标志的话，onActivityResult 将会立即接收到一个 cancle 的信息。

```
final boolean launchSingleTop = r.launchMode == ActivityInfo.LAUNCH_SINGLE_TOP;
final boolean launchSingleInstance = r.launchMode == ActivityInfo.LAUNCH_SINGLE_INSTANCE;
final boolean launchSingleTask = r.launchMode == ActivityInfo.LAUNCH_SINGLE_TASK;
int launchFlags = intent.getFlags();
if ((launchFlags & Intent.FLAG_ACTIVITY_NEW_DOCUMENT) != 0 &&
        (launchSingleInstance || launchSingleTask)) {
    // We have a conflict between the Intent and the Activity manifest, manifest wins.
    Slog.i(TAG, "Ignoring FLAG_ACTIVITY_NEW_DOCUMENT, launchMode is " +
            "\"singleInstance\" or \"singleTask\"");
    launchFlags &=
            ~(Intent.FLAG_ACTIVITY_NEW_DOCUMENT | Intent.FLAG_ACTIVITY_MULTIPLE_TASK);
} else {
    switch (r.info.documentLaunchMode) {
        case ActivityInfo.DOCUMENT_LAUNCH_NONE:
            break;
        case ActivityInfo.DOCUMENT_LAUNCH_INTO_EXISTING:
            launchFlags |= Intent.FLAG_ACTIVITY_NEW_DOCUMENT;
            break;
        case ActivityInfo.DOCUMENT_LAUNCH_ALWAYS:
            launchFlags |= Intent.FLAG_ACTIVITY_NEW_DOCUMENT;
            break;
```

```
        case ActivityInfo.DOCUMENT_LAUNCH_NEVER:
            launchFlags &= ~Intent.FLAG_ACTIVITY_MULTIPLE_TASK;
            break;
    }
}
final boolean launchTaskBehind = r.mLaunchTaskBehind
        && !launchSingleTask && !launchSingleInstance
        && (launchFlags & Intent.FLAG_ACTIVITY_NEW_DOCUMENT) != 0;
if (r.resultTo != null && (launchFlags & Intent.FLAG_ACTIVITY_NEW_TASK) != 0) {
    // For whatever reason this activity is being launched into a new
    // task...  yet the caller has requested a result back.  Well, that
    // is pretty messed up, so instead immediately send back a cancel
    // and let the new task continue launched as normal without a
    // dependency on its originator.
    Slog.w(TAG, "Activity is launching as a new task, so cancelling activity result.");
    r.resultTo.task.stack.sendActivityResultLocked(-1,
            r.resultTo, r.resultWho, r.requestCode,
            Activity.RESULT_CANCELED, null);
    r.resultTo = null;
}
```

- 而 5.0 之后这个方法做了修改，修改之后即便启动的页面设置 launchMode 为 singleTask 或 singleInstance，onActivityResult 依旧可以正常工作，也就是说无论设置哪种启动方式，StartActivityForResult 和 onActivityResult() 这一组合都是有效的。所以如果你目前正好基于 5.0 做相关开发，不要忘了向下兼容，这里有个坑请注意避让。
- 所以下面的结论可能不对，需要改天代码再好好验证一下

```
if (sourceRecord == null) {
    // This activity is not being started from another...
    // in this case we -ALWAYS- start a new task.
    if ((launchFlags & Intent.FLAG_ACTIVITY_NEW_TASK) == 0) {
        Slog.w(TAG, "startActivity called from non-Activity context; forcing Intent.FLAG_ACTIVITY_NEW_TASK for: "
              + intent);
        launchFlags |= Intent.FLAG_ACTIVITY_NEW_TASK;
    }
} else if (sourceRecord.launchMode == ActivityInfo.LAUNCH_SINGLE_INSTANCE) {
    // The original activity who is starting us is running as a single
    // instance...  this new activity it is starting must go on its
    // own task.
    launchFlags |= Intent.FLAG_ACTIVITY_NEW_TASK;
} 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;
}
if (r.resultTo != null && (launchFlags&Intent.FLAG_ACTIVITY_NEW_TASK) != 0) {
    // For whatever reason this activity is being launched into a new
    // task...  yet the caller has requested a result back.  Well, that
    // is pretty messed up, so instead immediately send back a cancel
    // and let the new task continue launched as normal without a
    // dependency on its originator.
    Slog.w(TAG, "Activity is launching as a new task, so cancelling activity result.");
    sendActivityResultLocked(-1,
            r.resultTo, r.resultWho, r.requestCode,
        Activity.RESULT_CANCELED, null);
    r.resultTo = null;
}
```

- 也就是说 startActivityForResult 启动的 activity 有 FLAG_ACTIVITY_NEW_TASK，那么就不能返回结果。(这个结论可能太古老了吧?!!!)

8.2 启动任务（Task）：这里要再消化一下！

- Intent filter 中有”android.intent.action.MAIN ”action 和”android.intent.category.LAUNCHER ” category 的 activity 将被标记为 task 的入口。带有这两个标记的 activity 将会显示在应用程序启动器（application launcher）中。

- 第二个比较重要的点是，用户必须能够离开 task 并在之后返回 。因为这个原因，singleTask 和 singleInstance 这两种运行模式只能应用于含有 MAIN 和 LAUNCHER 过滤器的 activity 。打个比方，如果不包含带 MAIN 和 LAUNCHER 过滤器，某个 activity 运行了一个 singleTask 模式的 activity，初始化了一个新的 task，当用户按下 HOME 键时，那个 activity 就被主屏幕“挡住”了，用户再也无法返回到那个 activity。(这里读得昏昏乎乎!!!)
- 类似的情况在 FLAG_ACTIVITY_NEW_TASK 标记上也会出现。如果这个标记会新建一个 task，当用户按下 HOME 键时，必须有一种方式能够让用户返回到那个 activity。有些东西（比如 notification manager）总是要求在外部 task 中启动 activity，在传递给 startActivity 的 intent 中总是包含 FLAG_ACTIVITY_NEW_TASK 标记。
- 对于那种不希望用户离开之后再返回 activity 的情况，可将 finishOnTaskLaunch 属性设置为 true。

8.3 Activity Stack

- 可以通过 adb shell dumpsys | grep ActivityRecord 来查看 TASKS 的 ActivityStacks
- 可以通过 adb shell dumpsys activity activities | grep packageName| grep Run 来查看某个 packageName 的 ActivityStatcks

9 onNewIntent

- 当通过 singleTop/singleTask 启动 activity 时，如果满足复用条件，则不会创建新的 activity 实例，生命周期就变为 onNewIntent()—->onResart()——>onStart()—->onResume()。
  - Activity 第一启动的时候执行 onCreate()—->onStart()—->onResume() 等后续生命周期函数，也就时说第一次启动 Activity 并不会执行到 onNewIntent().;
  - 而后面如果再有想启动 Activity 的时候，那就是执行 onNewIntent()—->onResart()——>onStart()—->onResume();
  - 如果 android 系统由于内存不足把已存在 Activity 释放掉了，那么再次调用的时候会重新启动 Activity 即执行 onCreate()—->onStart()—->onResume() 等。
- 注意:当调用到 onNewIntent(intent) 的时候,需要在 onNewIntent() 中使用 setIntent(intent) 赋值给 Activity 的 Intent. 否则，后续的 getIntent() 都是得到老的 Intent。

10 Activity 所需的任务栈与 TaskAffinity

- 这要从一个参数说起：TaskAffinity(任务相关性)。这个参数标识了一个 activity 所需的任务栈的名字，默认情况下，所有 activity 所需的任务栈的名字为应用的包名。
- TaskAffinity 参数标识着 Activity 所需要的任务栈的名称，默认情况下，一个应用中所有 Activity 所需要的任务栈名称都为该应用的包名。
- TaskAffinity 属性一般跟 singleTask 模式或者跟 allowTaskReparenting 属性结合使用，在其他情况下没有实际意义。

10.1 TaskAffinity 和 singleTask 启动模式结合使用

- 当 TaskAffinity 和 singleTask 启动模式结合使用时，当前 Activity 的任务栈名称将与 TaskAffinity 属性指定的值相同，下面我们通过代码来验证，我们同过 MainActivity 来启动 ActivityA，其中 MainActivity 启动模式为默认模式，ActivityA 启动模式为 singleTask，而 TaskAffinity 属性值为 android:taskAffinity="com.zejian.singleTask.affinity"

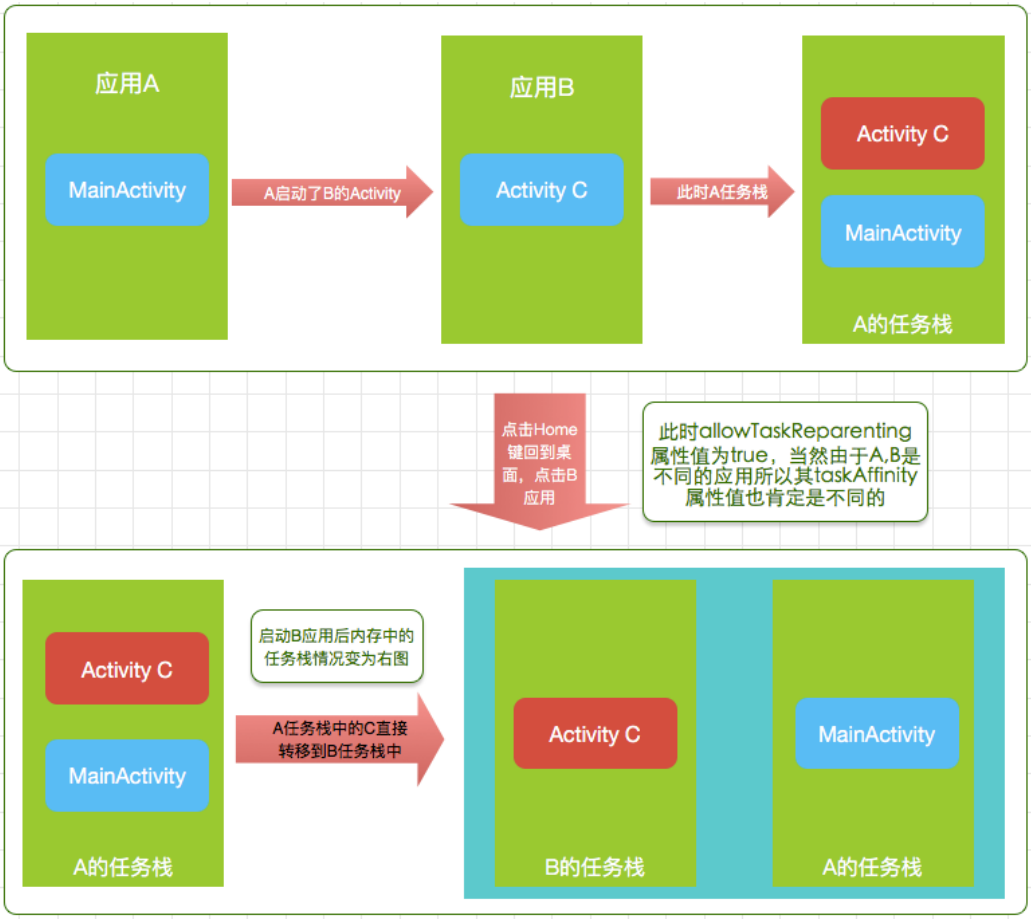
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="comzejian.myapplication">
  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">

    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <activity android:name=".ActivityA"
              android:launchMode="singleTask"
              android:taskAffinity="com.zejian.singleTask.affinity"
              />

  </application>
</manifest>
```

- 可以通过 singleTask 与 android:taskAffinity 属性相结合的方式来指定我们 Activity 所需要的栈名称，使相应的 Activity 存在于不同的栈中

10.2 当 TaskAffinity 和 allowTaskReparenting 结合使用

10.2.1 allowTaskReparenting 属性

- 它的主要作用是 activity 的迁移，即从一个 task 迁移到另一个 task，这个迁移跟 activity 的 taskAffinity 有关。
  - 当 allowTaskReparenting 的值为“true”时，则表示 Activity 能从启动的 Task 移动到有着 affinity 的 Task（当这个 Task 进入到前台时），
  - 当 allowTaskReparenting 的值为“false”，表示它必须呆在启动时呆在的那个 Task 里。如果这个特性没有被设定，元素 (当然也可以作用在每次 activity 元素上) 上的 allowTaskReparenting 属性的值会应用到 Activity 上。默认值为“false”。



- 举个例子，比如现在有两个应用 A 和 B，A 启动了 B 的一个 ActivityC，然后按 Home 键回到桌面，再单击 B 应用时，如果此时，allowTaskReparenting 的值为 “true”，那么这个时候并不会启动 B 的主 Activity，而是直接显示已被应用 A 启动的 ActivityC，我们也可以认为 ActivityC 从 A 的任务栈转移到了 B 的任务栈中。

1. 用代码来码证一下

    - ActivityA

```
public class ActivityA extends Activity {
    private Button btnC;
    @Override
        protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_a);
        btnC = (Button) findViewById(R.id.mainC);
        btnC.setOnClickListener(new View.OnClickListener() {
                @Override public void onClick(View v) {
                    Intent intent = new Intent(Intent.ACTION_MAIN);
                    intent.addCategory(Intent.CATEGORY_LAUNCHER); // 去打开 B 应用中的 activity
                    ComponentName cn = new ComponentName("com.cmcm.activitytask2", "com.cmcm.activitytask2.ActivityC"
                    intent.setComponent(cn);
                    startActivity(intent);
                }
            });
    }
}
```

    - A 应用的 manifest.xml

```
<activity android:name=".ActivityA">
     <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
</activity>
```

- B 应用中的启动模式以及标志位的设置

```
public class ActivityC extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_c);
    }
}
```

- B 应用的 manifest.xml

```
<activity android:name=".ActivityC" android:exported="true"
      android:allowTaskReparenting="true">
</activity>
```

2. 查看 Activity 的返回栈
   - adb shell dumpsys activity // 找
   - ACTIVITY MANAGER RECENT TASKS (dumpsys activity recents)
   - ACTIVITY MANAGER ACTIVITIES (dumpsys activity activities)

10.2.2 注意点

- 有点需要说明的是 allowTaskReparenting 仅限于 singleTop 和 standard 模式，这是因为一个 activity 的 affinity 属性由它的 taskAffinity 属性定义（代表栈名），而一个 task 的 affinity 由它的 root activity 定义。所以，一个 task 的 root activity 总是拥有和它所在 task 相同的 affinity。
- 由于以 singleTask 和 singleInstance 启动的 activity 只能是一个 task 的 root activity，因此 allowTaskReparenting 仅限于以 standard 和 singleTop 启动的 activity
- 列一下清单文件中 <activity> 元素的几个关键属性
  - launchMode
  - taskAffinity
  - allowTaskReparenting
  - clearTaskOnLaunch
  - alwaysRetainTaskState
  - finishOnTaskLaunch

10.2.3 taskAffinity 在两种情况下起作用：

1. 当启动 Activity 的 Intent 中带有 FLAG_ACTIVITY_NEW_TASK 标志时。
   - 在默认情况下，目标 Activity 将与 startActivity 的调用者处于同一 task 中。但如果用户特别指定了 FLAG_ACTIVITY_NEW_TASK，表明它希望为 Activity 重新开设一个 Task。这时就有两种情况：
     - 假如当前已经有一个 Task，它的 affinity 与新 Activity 是一样的，那么系统会直接用此 Task 来完成操作，而不是另外创建一个 Task；

- 否则系统需要创建一个 Task。

2. 当 Activity 中的 allowTaskReparenting 属性设置为 true 时。

   - 在这种情况下，Activity 具有" 动态转移" 的能力。举个前面的" 短信" 例子，在默认情况下，该应用程序中的所有 Activity 具有相同的 affinity。
   - 当另一个程序启动了" 短信编辑" 时，一开始这个 Activity 和启动它的 Activity 处于同样的 Task 中。但如果" 短信编辑"Activity 指定了 allowTaskReparenting，且后期" 短信" 程序的 Task 转为前台，此时" 短信编辑" 这一 Activity 会被" 挪" 到与它更亲近的" 短信"Task 中。

10.3 清空任务栈

- Android 系统除了给我提供了 TaskAffinity 来指定任务栈名称外，还给我提供了清空任务栈的方法，在一般情况下我们只需要在 <activity> 标签中指明相应的属性值即可。
- 如果用户将任务切换到后台之后过了很长一段时间，系统会将这个任务中除了最底层的那个 Activity 之外的其它所有 Activity 全部清除掉。当用户重新回到这个任务的时候，最底层的那个 Activity 将得到恢复。这个是系统默认的行为，因为既然过了这么长的一段时间，用户很有可能早就忘记了当时正在做什么，那么重新回到这个任务的时候，基本上应该是要去做点新的事情了。
- 当然，既然说是默认的行为，那就说明我们肯定是有办法来改变的，在元素中设置以下几种属性就可以改变系统这一默认行为：

10.3.1 android:alwaysRetainTaskState

- **alwaysRetainTaskState 实际上是给了当前 Activity 所在的任务栈一个 “免死金牌”**，如果当前 Activity 的 android:alwaysRetainTaskState 设置为 true 时，那么该 Activity 所在的任务栈将不会受到任何清理命令的影响，一直保持当前任务栈的状态。
- 如果将最底层的那个 Activity 的这个属性设置为 true，那么上面所描述的默认行为就将不会发生，任务中所有的 Activity 即使过了很长一段时间之后仍然会被继续保留。

10.3.2 clearTaskOnLaunch

- 如果将最底层的那个 Activity 的这个属性设置为 true，那么只要用户离开了当前任务，再次返回的时候就会将最底层 Activity 之上的所有其它 Activity 全部清除掉。简单来讲，**就是一种和 alwaysRetainTaskState 完全相反的工作模式，它保证每次返回任务的时候都会是一种初始化状态，即使用户仅仅离开了很短的一段时间**。
- 这个属性用来标记是否从 task 清除除根 Activity 之外的所有的 Activity，“true” 表示清除，“false” 表示不清除，默认为 “false” 。这里有点我们必须要注意的，这个属性只对任务栈内的 root Activity 起作用，任务栈内其他的 Activity 都会被忽略。如果 android:clearTaskOnLaunch 属性为 “true”，每次我们重新进入这个应用时，我们只会看到根 Activity，任务栈中的其他 Activity 都会被清除出栈。
- 比如一个应用的 Activity A,B,C，其中 clearTaskOnLaunch 设置为 true，C 为默认值，我们依次启动 A,B,C，点击 HOME, 再在桌面点击图标。启动的是 A，而 B，C 将都被移除当前任务栈。也就是说，当 Activity 的属性 clearTaskOnLaunch 为 true 时将被优先启动，其余的 Activity(B、C) 都被移除任务栈并销毁，除非前面 A 已经 finish 销毁，后面的已注册 clearTaskOnLaunch 为 true 的 activity(B) 才会生效。

-特别地,如果我们的应用中引用到了其他应用的 Activity,这些 Activity 设置了 android:allowTaskReparent 属性为“true”，则它们会被重新宿主到有共同 affinity 的 task 中。

- 特别地，如果一个 activity 的 allowTaskReparenting 属性为 true，那么它可以从一个 task（TASK1）移到另外一个有相同 affinity 的 task（TASK2）中（TASK2 带到前台时）。

如果一个.apk 文件从用户角度来看包含了多个“应用程序”，你可能需要对那些 activity 赋不同的 affinity 值。

10.3.3 android:finishOnTaskLaunch

- **这个属性和 clearTaskOnLaunch 是比较类似的，不过它不是作用于整个任务上的，而是作用于单个 Activity 上**。如果某个 Activity 将这个属性设置成 true，那么用户一旦离开了当前任务，再次返回时这个 Activity 就会被清除掉。
- finishOnTaskLaunch 属性与 clearTaskOnLaunch 有些类似,它们的区别是 finishOnTaskLaunch 是作用在自己身上 (把自己移除任务栈，不影响别的 Activity)，而 clearTaskOnLaunch 则是作用在别人身上 (把别的 Activity 移除任务栈),如果我们把 Activity 的 android:finishOnTaskLaunch 属性值设置为 true 时，离开这个 Activity 所依赖的任务栈后，当我们重新返回时，该 Activity 将会被 finish 掉，而且其他 Activity 不会受到影响。

11 启动模式的应用场景

11.1 SingleTask 模式的运用场景

- 最常见的应用场景就是保持我们应用开启后仅仅有一个 Activity 的实例。最典型的样例就是应用中展示的主页 (Home 页)。
- 假设用户在主页跳转到其他页面，运行多次操作后想返回到主页，假设不使用 SingleTask 模式，在点击返回的过程中会多次看到主页，这明显就是设计不合理了。

11.2 SingleTop 模式的运用场景

- 假设你在当前的 Activity 中又要启动同类型的 Activity，此时建议将此类型 Activity 的启动模式指定为 SingleTop，能够降低 Activity 的创建，节省内存!

11.3 SingleInstance 模式的运用场景

- SingleInstance 是 activity 启动的一种模式，一般做应用层开发很少用到，我一般用到的 app 定时提醒会用到这个模式吧。这个模式使用起来有很多坑，假设有 activityA，activityB，activityC 这三个 activity，我们将 activityB 设置为 SingleInstance

11.3.1 第一种情况

- A 开启 B, B 开启 C, 如果 finish activityC, 那么 activityA 会显示而不是我们想要的 activityB，这是因为 activityB 和 activityA、activityC 所处的栈不同，C 关闭了，就要显示 C 所处栈的下一个 activity，解决这个问题办法很多，我自己用的方法是通过记录开启 activity，在被关闭的 activity 的 finish 方法中重新开启 activityB。

11.3.2 第二种情况

A 开启 B，然后按 home 键，再从左面点开应用，显示的是 A，这是因为 launch 启动我们应用的时候会从默认的栈找到栈顶的 activity 显示，这个解决办法的思路跟第一种差不多，也就不献丑了

11.3.3 第三种情况

- A 开启 C，C 开启 B，B 开启 A，结果显示的是 C，这还是两个栈造成的，B 开启 A 的时候，其实是到达 A 所处的栈，栈顶是 C，所以就显示 C 了，解决办法是用 flag 把默认栈 activity 清理了，重新开启 A，或者回退到 C 时再开启 A。

11.3.4 总结

- 三种情况的解决方法都是基于页面少的情况，如果页面多了会产生更多的问题
- 为了必避免这个问题，最好不用在中间层使用 SingleInstance
- TIPS:
  - (1) 如果想让 C 和 B 同一个栈，那就使用 taskinfinity，给他俩设置同样的栈名
  - (2)onActivityResult 不能与 SingleInstance 不能一起使用，因为不同栈

11.4 standard 运用场景

- Activity 的启动默认就是这种模式。在 standard 模式下，每次启动一个 Activity 都会创建一个新的实例;
- 在正常应用中正常打开和关闭页面就可以了，退出整个 app 就关闭所有的页面

12 Activity 时的生命周期不同

- 由于当一个 Activity 设置了 SingleTop 或者 SingleTask 模式或者 SingleInstance 模式后，跳转此 Activity 出现复用原有 Activity 的情况时，此 Activity 的 onCreate 方法将不会再次运行。onCreate 方法仅仅会在第一次创建 Activity 时被运行。
- 而一般 onCreate 方法中会进行该页面的数据初始化、UI 初始化，假设页面的展示数据无关页面跳转传递的参数，则不必操心此问题，若页面展示的数据就是通过 getInten() 方法来获取，那么问题就会出现：getInten() 获取的一直都是老数据，根本无法接收跳转时传送的新数据!
- 这时我们须要另外一个回调 onNewIntent(Intent intent) 方法。此方法会传入最新的 intent，这样我们就能够解决上述问题。这里建议的方法是又一次去 setIntent。然后又一次去初始化数据和 UI

```
/** 复用 Activity 时的生命周期回调 */
@Override
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);
    setIntent(intent);
    initData();
    initView();
}
```

13 实际中的栈管理类

- 管理 Activity 的类，一般在 BaseActivity 会调用这个类，然后所有的 Activity 继承 BaseActivity，这样管理好整个项目的 Activity

```java
public class ActivityStackManager { // activity 堆栈管理
    private static ActivityStackManager mInstance;
    private static Stack<Activity> mActivityStack;
    public static ActivityStackManager getInstance() {
        if (null == mInstance)
            mInstance = new ActivityStackManager();
        return mInstance;
    }
    private ActivityStackManager() {
        mActivityStack = new Stack<Activity>();
    }
    public void addActivity(Activity activity) {  // 入栈
        mActivityStack.push(activity);
    }
    public void removeActivity(Activity activity) {  // 出栈
        mActivityStack.remove(activity);
    }
    public void finishAllActivity() {  // 彻底退出
        Activity activity;
        while (!mActivityStack.empty()) {
            activity = mActivityStack.pop();
            if (activity != null)
                activity.finish();
        }
    }
    public void finishActivity(Class<?> cls) {  // 结束指定类名的 Activity
        for (Activity activity : mActivityStack) {
            if (activity.getClass().equals(cls)) {
                finishActivity(activity);
            }
        }
    }
    public boolean checkActivity(Class<?> cls) {  // 查找栈中是否存在指定的 activity
        for (Activity activity : mActivityStack) {
            if (activity.getClass().equals(cls)) {
                return true;
            }
        }
        return false;
    }
    public void finishActivity(Activity activity) {  // 结束指定的 Activity
        if (activity != null) {
            mActivityStack.remove(activity);
            activity.finish();
            activity = null;
        }
    }
    public boolean finishToActivity(Class<? extends Activity> actCls, boolean isIncludeSelf) { // finish 指定的 activity 之_
        List<Activity> buf = new ArrayList<Activity>();
        int size = mActivityStack.size();
        Activity activity = null;
        for (int i = size - 1; i >= 0; i--) {
            activity = mActivityStack.get(i);
            if (activity.getClass().isAssignableFrom(actCls)) {
                for (Activity a : buf)
                    a.finish();
                return true;
            } else if (i == size - 1 && isIncludeSelf)
                buf.add(activity);
            else if (i != size - 1)
                buf.add(activity);
        }
        return false;
    }
}
```

14 AndroidManifest 配置相关参数的说明详解

14.1 android:allowEmbedded

- 表示该 activity 可作为其他 activity 的嵌入式子项启动。此属性尤其适用于子项位于其他 Activity 所拥有容器（如 Display）中的情况。
- 比如 CarLauncher 中三个应用：车控/Hvaa/CarSettings 都分别设置了这个属性

14.2 android:stateNotNeeded="true" 的作用

- 这个属性默认情况为 false，若设为 true，则当 Activity 重新启动时不会调用 onSaveInstanceState () 方法，同样，onCreate () 方法中的 Bundle 参数将会用 null 值传进去，也就是说，Activity 每次启动都跟第一次启动一样。这样，在某种特殊场合下，由于用户按了 Home 键，该属性设置为 true 时，可以保证不用保存原先的状态引用，节省了空间资源，从而可以让 Activity 不会像默认设置那样 Crash 掉。

15 ActivityStack

- 任务是一个 Activity 的集合，它使用栈的方式来管理其中的 Activity，这个栈又被称为返回栈 (back stack)，栈中 Activity 的顺序就是按照它们被打开的顺序依次存放的。

16 安卓如何管理多个任务 (没被称谓多个任务栈，虽然本质上仍是多个任务栈，每个任务都是一个栈结构)

16.1 Android 多任务运行机制

- 任务（Task1）是一个有机整体，当用户开始新任务（Task2）或通过“主页”按钮（Home 键）转到主屏幕时，可以将该任务（Task1）移动到“后台”。尽管在后台时，该任务（Task1）中的所有 Activity 全部停止，但是任务（Task1）的返回栈仍旧不变，也就是说，当另一个任务（Task2）发生时，该任务（Task1）仅仅失去焦点而已，如图 2 中所示。然后，任务（Task1）可以返回到“前台”，用户就能够回到离开时的状态。
- 例如，假设当前任务（任务 A）的堆栈中有三个 Activity，即当前 Activity 下方还有两个 Activity。用户先按“主页”按钮（Home 键），然后从应用启动器启动新应用。显示主屏幕时，任务 A 进入后台。新应用启动时，系统会使用自己的 Activity 堆栈为该应用启动一个任务（任务 B）。与该应用交互之后，用户再次返回主屏幕并选择最初启动任务 A 的应用。现在，任务 A 出现在前台，其堆栈中的所有三个 Activity 保持不变，而位于堆栈顶部的 Activity 则会恢复执行。此时，用户还可以通过转到主屏幕并选择启动该任务的应用图标（或者，通过从概览屏幕选择该应用的任务）切换回任务 B。这是 Android 系统中的一个多任务示例。
- 注意：后台可以同时运行多个任务。但是，如果用户同时运行多个后台任务，则系统可能会开始销毁后台 Activity，以回收内存资源，从而导致 Activity 状态丢失。请参阅下面有关 Activity 状态的部分。
- Activity 和任务的默认行为总结如下：
  - 当 Activity A 启动 Activity B 时，Activity A 将会停止，但系统会保留其状态（例如，滚动位置和已输入表单中的文本）。如果用户在处于 Activity B 时按“返回”按钮，则 Activity A 将恢复其状态，继续执行。
  - 用户通过按“主页”按钮（Home 键）离开任务时，当前 Activity 将停止且其任务会进入后台。系统将保留任务中每个 Activity 的状态。如果用户稍后通过选择开始任务的启动器图标来恢复任务，则任务将出现在前台并恢复执行堆栈顶部的 Activity。

  - 如果用户按“返回”按钮，则当前 Activity（处于栈顶的那个 Activity 即上面 1 中的 ActivityB）会从堆栈弹出并被销毁。堆栈中的前一个 Activity（Activity A）恢复执行。销毁 Activity 时，系统不会保留该 Activity 的状态。
  - 即使来自其他任务，Activity 也可以多次实例化。（但是多次实例化，会造成内存资源浪费，具体可参考下面）

- 一个 Activity 将多次实例化
  - 由于返回栈中的 Activity 永远不会重新排列，因此如果应用允许用户从多个 Activity 中启动特定 Activity，则会创建该 Activity 的新实例并推入堆栈中（而不是将 Activity 的任一先前实例置于顶部）。因此，应用中的一个 Activity 可能会多次实例化（即使 Activity 来自不同的任务），如图 3 所示。因此，如果用户使用“返回”按钮向后导航，则会按 Activity 每个实例的打开顺序显示这些实例（每个实例的 UI 状态各不相同）。但是，如果您不希望 Activity 多次实例化，则可修改此行为。具体操作方法将在后面的管理任务部分中讨论。

- 保存 Activity 状态
  - 正如上文所述，当 Activity 停止时，系统的默认行为会保留其状态。这样一来，当用户导航回到上一个 Activity 时，其用户界面与用户离开时一样。但是，在 Activity 被销毁且必须重建时，您可以而且应当主动使用回调方法保留 Activity 的状态。
  - 系统停止您的一个 Activity 时（例如，新 Activity 启动或任务转到前台（即重新开始一个新的任务）），如果系统需要回收系统内存资源，则可能会完全销毁该 Activity。发生这种情况时，有关该 Activity 状态的信息将会丢失。如果发生这种情况，系统仍会知道该 Activity 存在于返回栈中，但是当该 Activity 被置于堆栈顶部时，系统一定会重建 Activity（而不是恢复 Activity）。为了避免用户的工作丢失，您应主动通过在 Activity 中实现 onSaveInstanceState() 回调方法来保留工作。

16.2 定义启动模式

- 启动模式允许您定义 Activity 的新实例如何与当前任务关联。您可以通过两种方法定义不同的启动模式：
  - 使用清单文件
    * 在清单文件中声明 Activity 时，您可以指定 Activity 在启动时应该如何与任务关联。
  - 使用 Intent 标志
    * 调用 startActivity() 时，可以在 Intent 中加入一个标志（上面管理任务章节中的 3 种 Intent 的 flag），用于声明新 Activity 如何（或是否）与当前任务关联。

- 注：某些适用于清单文件的启动模式不可用作 Intent 标志，同样，某些可用作 Intent 标志的启动模式无法在清单文件中定义。

- 因此，如果 Activity A 启动 Activity B，则 Activity B 可以在其清单文件中定义它应该如何与当前任务关联（如果可能），并且 Activity A 还可以请求 Activity B 应该如何与当前任务关联。如果这两个 Activity（Activity A 和 Activity B）均定义 Activity B 应该如何与任务关联（可以一个在 intent 中，一个在清单文件中），**则 Activity A 的请求（如 Intent 中所定义）优先级要高于 Activity B 的请求（如其清单文件中所定义）**。