Activity 状态更改

原创 xyzso1z 2020-06-01 20:09:21 111 收藏 1 原力计划 编辑 版权
分类专栏： Android 文章标签： Activity Activity状态更改 配置发生更改生命周期 Activity弹窗生命周期 onDestroy不回调

> 前言

- 用户触发和系统触发的不同事件会导致`Activity`从一个状态转换到另一个状态。本文主要介绍发上此类转换的一些常见情况，以及如何处理这些转换。
- 在交接本文之前需要知道`Activity`状态的情况，可以查看上一篇文章 《Activity生命周期》

1. 配置发生了更改

有很多事件会触发配置更改。最显著的例子就是横竖屏之间的切换。其他情况，如语言或输入设备的改变等，也可能导致配置更改。

当配置发生更改时，`Activity`会销毁并重新创建。原始`Activity`实例将触发`onPause()`、`onStop()`、`onSaveInstanceState()`、`onDestroy()`回调。系统将创建新的`Activity`实例，并触发`onCreate()`、`onStart()`、`onRestoreInstanceState()`、`onResume()`回调。

- 进入Activity:

```
2020-06-01 16:12:11.218 6286-6286/com.xyz.activity E/xx: the Activity of 14451054
2020-06-01 16:12:11.218 6286-6286/com.xyz.activity E/xx: onCreate
2020-06-01 16:12:11.220 6286-6286/com.xyz.activity E/xx: onStart
2020-06-01 16:12:11.220 6286-6286/com.xyz.activity E/xx: onResume
```

- 旋转屏幕：

```
2020-06-01 16:13:52.591 6286-6286/com.xyz.activity E/xx: onPause
2020-06-01 16:13:52.592 6286-6286/com.xyz.activity E/xx: onStop
2020-06-01 16:13:52.592 6286-6286/com.xyz.activity E/xx: onSaveInstanceState
2020-06-01 16:13:52.592 6286-6286/com.xyz.activity E/xx: onDestroy
2020-06-01 16:13:52.593 6286-6286/com.xyz.activity E/xx: the Activity of 14451054
2020-06-01 16:13:52.618 6286-6286/com.xyz.activity E/xx: the Activity of 29845855
2020-06-01 16:13:52.618 6286-6286/com.xyz.activity E/xx: onCreate
2020-06-01 16:13:52.620 6286-6286/com.xyz.activity E/xx: onStart
2020-06-01 16:13:52.622 6286-6286/com.xyz.activity E/xx: onRestoreInstanceState
2020-06-01 16:13:52.622 6286-6286/com.xyz.activity E/xx: key get :hello world!
2020-06-01 16:13:52.623 6286-6286/com.xyz.activity E/xx: onResume
```

> 拓展：如何防止旋转屏幕导致Activity重新创建

- 方法1：禁止旋转屏幕(简单直接，对于没有旋转屏幕需求的app可使用)

```
<activity
        android:name=".MyActivity"
       android:screenOrientation="portrait" />
```

- 方法2：修改 `AndroidManifest.xml` 配置
  在activity属性中加入：

```
<activity
        android:name=".ConfigChangeActivity"
       android:configChanges="orientation|screenSize" />
```

`android:configChanges` ：这个方法主要是负责列出清单，当清单上用户指定的设置改变时，Activity会自己处理这些变化。
`orientation` ：屏幕界面旋转（可能是用户手动旋转的），**【注意：如果你的开发API等级等于或高于13，你还需要设置 `screenSize` ，因为screenSize会在屏幕旋转时改变】**

2. Activity 或对话框显示在前台

如果有新的 `Activity` 或对话框出现在前台，并且**局部覆盖**了正在进行的 `Activity` ，则被覆盖的 `Activity` 会失去焦点并进入“已暂停”状态。然后，系统会调用 `onPause()` 。

当被覆盖的 `Activity` 返回到前台并重新获得焦点时，会调用 `onResume()` 。

如果有新的 `Activity` 或对话框出现在前台，夺取了焦点且**完全覆盖**了正在进行的 `Activity` ，则被覆盖的 `Activity` 会失去焦点并进入“Stoped”状态。然后，系统会快速地接连调用 `onPause()` 和 `onStop()` 。

当被覆盖的 `Activity` 的同一实例返回到前台时，系统会对该 `Activity` 调用 `onRestart()` 、`onStart()` 和 `onResume()` 。如果被覆盖的 Activity 的新实例进入后台，则系统不会调用 `onRestart()` ，而只会调用 `onStart()` 和 `onResume()` 。

3. 用户点按“返回”按钮

如果 `Activity` 位于前台，并且用户点按了返回按钮， `Activity` 将依次经历 `onPause()` 、`onStop()` 、 `onDestroy()` 回调。活动不仅会被销毁，还会从返回堆栈中移除。

需要注意的是，在这种情况下，默认不会触发 `onSaveInstanceState()` 回调。此行为基于的假设是，用户点按返回按钮时不期望返回 `Activity` 的同一实例。不过，您可以通过替换 `onBackPressed()` 方法实现某种自定义行为，例如“confirm-quit”对话框。

如果您替换 `onBackPressed()` 方法， 建议被替换的方法调用 super.onBackPressed()。否则，返回按钮的行为可能会让用户感觉突兀。

4. 杀死后台应用

该操作分为三步，分别进行讲解：

1. 用户进入Activity,Activity将依次经历`onCreate()`、`onStart()`、`onResume()`回调。

```
2020-06-01 19:32:39.666 5840-5840/com.xyz.activity E/xx: the Activity of 4362927
2020-06-01 19:32:39.666 5840-5840/com.xyz.activity E/xx: onCreate
2020-06-01 19:32:39.668 5840-5840/com.xyz.activity E/xx: onStart
2020-06-01 19:32:39.668 5840-5840/com.xyz.activity E/xx: onResume
```

2. 用户按Home键，使应用退到后台,`Activity`将依次经历`onPause()`、`onStop()`、`onSaveInstanceState()`回调。

```
2020-06-01 19:37:32.071 6168-6168/com.xyz.activity E/xx: onPause
2020-06-01 19:37:32.104 6168-6168/com.xyz.activity E/xx: onStop
2020-06-01 19:37:32.104 6168-6168/com.xyz.activity E/xx: onSaveInstanceState
```

3. 如果用户再次进入应用，`Activity`将会依次经历onRestart()、onStart()、onResume()回调。

```
2020-06-01 19:40:08.292 6168-6168/com.xyz.activity E/xx: onRestart
2020-06-01 19:40:08.295 6168-6168/com.xyz.activity E/xx: onStart
2020-06-01 19:40:08.296 6168-6168/com.xyz.activity E/xx: onResume
```

4. 如果用户杀死该后台应用，Activity将会可能会经历`onDestroy()`，但很有可能不会执行`onDestroy()`所以尽量不要在`onDestroy()`中进行资源释放。

关于第4点进行拓展：
从后台强杀分两种情况：

第一种：当前仅有一个activity，这时候，强杀，是会执行onDestroy方法的；

第二种：栈里面的第一个没有销毁的activity会执行ondestroy方法，其他的不会执行。

比如说：从`Mainactivity`跳转到`activityA`（或者继续从`activityA`再跳转到`activityB`），这时候，从后台强杀，只会执行`Mainactivity`的`onDestroy`方法，`ActivityA`（以及`activityB`）的`onDestroy`方法都不会执行；（原文）

试了一下 该分析也不一定有效：

```
2020-06-01 20:05:55.912 10129-10129/com.xyz.activity E/xx: the Activity of 256450186 i
2020-06-01 20:05:55.912 10129-10129/com.xyz.activity E/xx: MainActivity onCreate
2020-06-01 20:05:55.920 10129-10129/com.xyz.activity E/xx: MainActivity onStart
2020-06-01 20:05:55.921 10129-10129/com.xyz.activity E/xx: MainActivity onResume
2020-06-01 20:05:58.622 10129-10129/com.xyz.activity E/xx: MainActivity onPause
2020-06-01 20:05:58.636 10129-10129/com.xyz.activity E/xx: the Activity of 244799732 i
2020-06-01 20:05:58.636 10129-10129/com.xyz.activity E/xx: NormalActivity onCreate
2020-06-01 20:05:58.638 10129-10129/com.xyz.activity E/xx: NormalActivity onStart
2020-06-01 20:05:58.639 10129-10129/com.xyz.activity E/xx: NormalActivity onResume
```

```
2020-06-01 20:05:59.088 10129-10129/com.xyz.activity E/xx: MainActivity onStop
2020-06-01 20:05:59.088 10129-10129/com.xyz.activity E/xx: MainActivity onSaveInstance
2020-06-01 20:06:01.008 10129-10129/com.xyz.activity E/xx: NormalActivity onPause
2020-06-01 20:06:01.042 10129-10129/com.xyz.activity E/xx: NormalActivity onStop
2020-06-01 20:06:01.042 10129-10129/com.xyz.activity E/xx: NormalActivity onSaveInstan
2020-06-01 20:06:01.665 10129-10129/com.xyz.activity E/xx: MainActivity onDestroy
2020-06-01 20:06:01.666 10129-10129/com.xyz.activity E/xx: the Activity of 256450186 i
2020-06-01 20:06:01.673 10129-10129/com.xyz.activity E/xx: NormalActivity onDestroy
2020-06-01 20:06:01.674 10129-10129/com.xyz.activity E/xx: the Activity of 244799732 i
```

代码