1. Activity生命周期

官方文档

https://developer.android.com/guide/components/activities.html#Lifecycle



* *被系统回收时，不会调用任何其他生命周期，但在再次启动时，会在onStart之后调用onRestoreInstanceState*
* *按下HOME键时，onSaveInstanceState在onPause之后调用*
* *按返回键退出时，会先调用onPause，再调用上层Activity的onRestart、onStart、onResume，再调用onStop、onDestroy。*
* *启动其他Activity时，会先调用onPause，再调用被启动的Activity的onCreate、onStart、onResume，再调用onSaveInstanceState、onStop*
* *singleTop、singleTask、singleInstance启动模式，onStop状态时，onNewIntent在onRestart之前调用*
* *singleTop、singleTask、singleInstance启动模式，Activity在顶部时，会先调用onPause，再调用onNewIntent、onResume*

1. 四中启动模式

官方文档

<https://developer.android.com/guide/topics/manifest/activity-element.html>

Activity在AndroidManifest文件的对应Activity标签设置android:launchMode属性。

**standard**默认模式，可以不用写配置。在这个模式下，都会默认创建一个新的实例。因此，在这种模式下，可以有多个相同的实例，也允许多个相同Activity叠加。

**singleTop**  
可以有多个实例，但是不允许多个相同Activity叠加。即，如果Activity在栈顶的时候，启动相同的Activity，不会创建新的实例，而会调用其onNewIntent方法。

**singleTask**  
**只有一个实例**。在同一个应用程序中启动他的时候，若Activity不存在，则会在当前task创建一个新的实例，若存在，则会把task中在其之上的其它Activity destory掉并调用它的onNewIntent方法。  
系统在新任务的根位置创建 Activity 并向其传送 Intent。 不过，如果已存在一个 Activity 实例，则系统会通过调用该实例的 [onNewIntent()](https://developer.android.com/reference/android/app/Activity.html" \l "onNewIntent(android.content.Intent)) 方法向其传送 Intent，而不是创建新的 Activity 实例。即不同程序公用一个Activity。

**singleInstance**  
**只有一个实例**，并且这个实例独立运行在一个task中，这个task只有这个实例，不允许有别的Activity存在。如果已经存在，再次启动会调用onNewIntent。

在一个新栈中创建该Activity实例，并让多个应用共享该栈中的该Activity实例。 一旦改模式的Activity的实例存在于某个栈中，任何应用再激活改Activity时都会重用该栈中的实例，其效果相当于多个应用程序共享一个应用，不管谁激活该Activity都会进入同一个应用中。

1. Activity重启时大量数据的恢复方法

如果在重启Activity时，要求恢复大数据集、重建网络连接、或执行其他的敏感操作，那么由于配置的改变完全重启会降低用户体验。此外，系统也 不可能用onSaveInstanceState()回调方法的Bundle对象来完整的保存要恢复的数据，因为Bundle对象没有被设计用来携带大对 象(如位图)和大数据，这样导致Bundle对象在系列化和随后的反系列化时要占用大量的内存，从而导致配置改变缓慢。这种情况下，可以选择保留一个状态 对象来减轻恢复Activity重启时的负担。

保留运行期间配置改变对象的方法如下：

1. 重写onRetainNonConfigurationInstance()回调方法，它会返回希望保留的对象。

2. 在Activity被重建时，调用getLastNonConfigurationInstance()方法来恢复这个对象。

当Android系统由于配置的变化关掉Activity时，它会在onStop()回调和onDestroy()之间调用 onRetainNonConfigurationInstance()回调方法。在 onRetainNonConfigurationInstance()回调的实现中，为了在配置变化之后能够有效的恢复状态，它可以返回任意任何需要的 对象。

如果应用程序要从网络上加载数据，这样做就很有价值。如果用户改变了设备的方向，并且Activity重启了，应用程序就必须重新获取数据，这样会 很慢。因此可以实现onRetainNonConfigurationInstance()方法，让它返回一个带有数据的对象，然后再Activity被 重启时，再用getLastNonConfigurationInstance()方法来获取这个被保留的对象，例如：

@Override

public Object onRetainNonConfigurationInstance() {

final MyDataObject data = collectMyLoadedData();

return data;

}

这个方法最大的好处是：

\* 当Activity曾经通过某个资源得到一些图片或者信息，那么当再次恢复后，无需重新通过原始资源地址获取，可以快速的加载整个Activity状态信息。

\* 当Activity包含有许多线程时，在变化后依然可以持有原有线程，无需通过重新创建进程恢复原有状态。

\* 当Activity包含某些Connection Instance时，同样可以在整个变化过程中保持连接状态。

下边是需要特别注意的几点：

\* onRetainNonConfigurationInstance()在onSaveInstanceState()之后被调用。

\* 调用顺序同样介于onStop() 和 onDestroy()之间。

警告：在返回对象时，不要带有跟Activity绑定的对象，如Drawable、Adapter、View或其他跟Context关联的对象。如 果这样做了，会导致最初的Activity实例的所有View对象和资源泄漏（泄漏资源意味着应用程序占用了这些资源，因而不能被作为垃圾来回收，这样就 会导致大量的内存占用）。

Activity重启时，恢复数据的方法：

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

final MyDataObject data = (MyDataObject) getLastNonConfigurationInstance();

if (data == null) {

data = loadMyData();

}

...

}

这个例子中，getLastNonConfigurationInstance()方法返回被 onRetainNonConfigurationInstance()方法保存的数据。如果data对象是null（在由于配置改变以外的其他原因而导 致的Activity重启时，会发生这种情况），那么代码就会从初始资源中装载数据对象。

1. 屏幕方向变化时