Android：Activity——生命周期深入详解

博乐推荐

标签： android 生命周期 内存 数据存储与恢复 2016-03-23 13:34 263人阅读 评论(2) 收藏 举报

分类： Android Coding（14） 编程体会（10） 内存管理（1）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?) [+]

一、生命周期全面分析

Android活动默认运行在当前进程所拥有的栈中，前台可见的活动则在活动栈的最顶部。其他后台活动则在栈的里面，在正常的情况下（内存充足）其他的活动并没有被回收或者杀死，它们仍然存在于栈中保持着原来的状态。当前面的活动退出后，后面的活动就会搬到前台使得被用户可见。如果在非正常情况下（内存紧张、按下Home键后右启动其他应用）那么栈内的非前台Activity就可能被回收，但是当我们返回到该Activity时它又会被重新构造，并且会通过onSaveInstance和onRestoreInstance加载原来的数据使得它保持之前的状态呈现给用户。无论是正常情况，还是非正常情况下，这样的实现都是基于一个非常重要的机制——生命周期。

1.什么是生命周期

周期即活动从开始到结束所经历的各种状态。生命周期即活动从开始到结束所经历的各个状态。从一个状态到另一个状态的转变，从无到有再到无，这样一个过程中所经历的状态就叫做生命周期。

- **Acitivity本质上有四种状态**：
  1. **运行：**如果一个活动被移到了前台（活动栈顶部）。
  2. **暂停：**如果一个活动被另一个非全屏的活动所覆盖（比如一个Dialog），那么该活动就失去了焦点，它将会暂停（但它仍然保留所有的状态和成员信息，并且仍然是依附在WindowsManager上），在系统内存积极缺乏的时候会将它杀死。
  3. **停止：**如果一个活动被另一个全屏活动完全覆盖，那么该活动处于停止状态（状态和成员信息会保留，但是Activity已经不再依附于WindowManager了）。同时，在系统缺乏资源的时候会将它杀死（它会比暂停状态的活动先杀死）。
  4. **重启：**如果一个活动在处于停止或者暂停的状态下，系统内存缺乏时会将其结束（finish）或者杀死（kill）。这种非正常情况下，系统在杀死或者结束之前会调用onSaveInstance（）方法来保存信息，同时，当Activity被移动到前台时，重新启动该Activity并调用onRestoreInstance（）方法加载保留的信息，以保持原有的状态。

在上面的四中常有的状态之间，还有着其他的生命周期来作为不同状态之间的过度，用于在不同的状态之间进行转换，生命周期的具体说明见下。

二、各个生命周期状态的说明

1.正常情况下的生命周期

1. **onCreate：**与onDestroy配对，表示Activity正在被创建，这是生命周期的第一个方法。在这个方法中可以做一些初始化的工作（加载布局资源、初始化Activity所需要的数据等），耗时的工作在异步线程上完成。

2. **onRestart：**表示Activity正在重新启动。一般情况下，在当前Activity从不可见重新变为可见的状态时onRestart就会被调用。这种情形一般是由于用户的行为所导致的，比如用户按下Home键切换到桌面或者打开了一个新的Activity（这时当前Activity会暂停，也就是onPause和onStop被执行），接着用户有回到了这个Activity，就会出现这种情况。

3. **onStart：**与onStop配对，表示Activity正在被启动，并且即将开始。但是这个时候要注意它与onResume的区别。两者都表示Activity可见，但是onStart时Activity还正在加载其他内容，正在向我们展示，用户还无法看到，即无法交互。

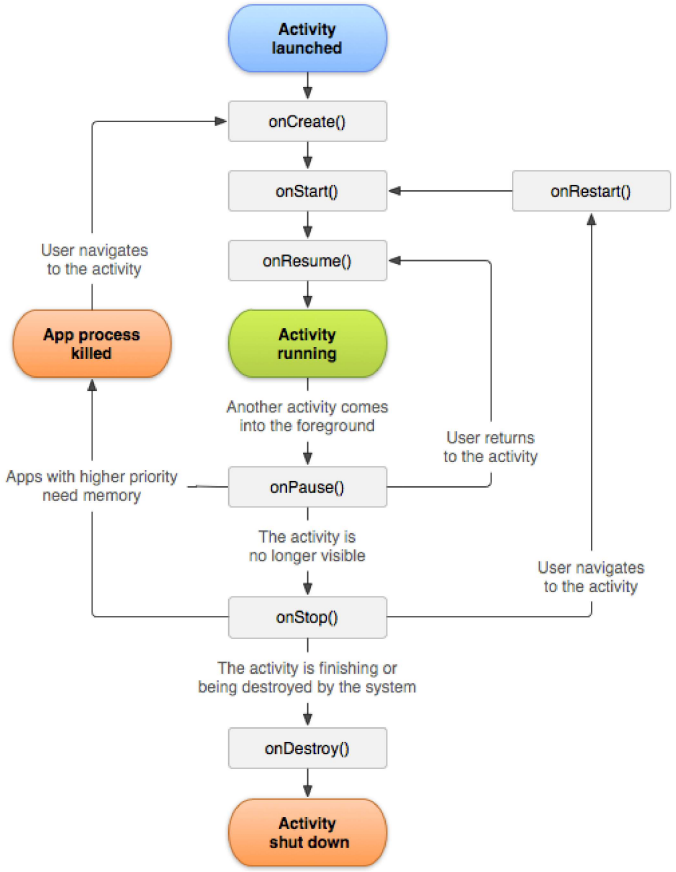
4. **onResume：**与onPause配对，表示Activity已经创建完成，并且可以开始活动了，这个时候用户已经可以看到界面了，并且即将与用户交互（完成该周期之后便可以响应用户的交互事件了）。

5. **onPause：**与onResume配对，表示Activity正在暂停，正常情况下，onStop接着就会被调用。在特殊情况下，如果这个时候用户快速地再回到当前的Activity,那么onResume会被调用（极端情况）。一般来说，在这个生命周期状态下，可以做一些存储数据、停止动画的工作，但是不能太耗时，如果是由于启动新的Activity而唤醒的该状态，那会影响到新Activity的显示，原因是onPause必须执行完，新的Activity的onResume才会执行。

6. **onStop：**与onStart配对，表示Activity即将停止，可以做一些稍微重量级的回收工作，同样也不能太耗时（可以比onPause稍微好一点）。

7. **onDestroy：**与onCreate配对，表示Activity即将被销毁，这是Activity生命周期的最后一个回调，我们可以做一些回收工作和最终的资源释放（如Service、BroadReceiver、Map等）。

   正常情况下，Activity的常用生命周期就是上面的7个，下图更加详细的描述的各种生命周期的切换过程：



这里要说的是，从上图我们可以看到一个现象：

onStart与onStop、onResume与onPause是配对的。两种Activity回到前台的方式，从onPause状态回到前台会走到onResume状态，从onStop状态回到前台会到onStart状态，这是从是否可见和是否在前台来说的。从是否可见来说，onStart和onStop是配对的；从是否在前台来说，onResume和onPause是配对的。至于为什么会有他们，在第三点生命周期的使用会说到。

我们来看看正常情况下生命周期的系统日志：

```
03-23 00:15:52.970 32278-32278/com.example.david.lifecircle E/TAG: onCreate() is invoked!
03-23 00:15:52.971 32278-32278/com.example.david.lifecircle E/TAG: onStart() is invoked!
03-23 00:15:52.971 32278-32278/com.example.david.lifecircle E/TAG: onResume() is invoked!
03-23 00:15:55.858 32278-32278/com.example.david.lifecircle E/TAG: onPause() is invoked!
03-23 00:16:02.573 32278-32278/com.example.david.lifecircle E/TAG: onRestart() is invoked!
03-23 00:16:02.573 32278-32278/com.example.david.lifecircle E/TAG: onStart() is invoked!
03-23 00:16:02.573 32278-32278/com.example.david.lifecircle E/TAG: onResume() is invoked!
```

2.异常情况下的生命周期

一般正常情况的声明周期就像上面所说的一样，但是因为Android本身内存或者其他的一些情况会使得Activity不按照正常的生命周期。比如当资源配置发生改变、系统内存

- **情况1：资源相关的系统配置发生改变导致Activity被杀死并重新创建**

  理解这个问题，我们首先要对系统的资源加载机制有一定了解，不过这里我不分析系统资源加载机制了（因为我也不怎么懂）。简单说明一下，就像是我们把一张图片放在drawable目录之后，就可以通过Resources去获取这张图片。同时为了兼容不同的设备，我们还可能需要在其他的一些目录放置不同的图片，比如 drawable-mdpi、drawable-hdpi等。这样，当应用程序启动时，系统就会根据当前设备的情况去加载合适的Resource资源，比如说横屏和竖屏的手机会拿到两张不同的图片（设定了landscape或portrait状态下的图片）。
  如果说，当前Activity处于竖屏状态，如果突然旋转屏幕，由于系统配置发生了改变，在默认情况下，Activity就会被销毁并重新创建（当然我们也可以组织系统重新创建，具体就在Mainfest中申明android:Configchanges=属性即可）。

  异常情况下的调用流程：

  1. 调用onSaveInstance保存当前Activity状态。注意，它与onPause方法没有先后之分。
  2. 调用onStop方法做后续处理。
  3. 调用onDestroy方法销毁当前活动。
  4. 重新onCreate该活动。
  5. 调用onStart方法之后，再调用onRestoreInstance方法加载保存的数据。
  6. 接下来就与正常的一样了，调用onResume，然后运行。

  我们来看一下生命周期异常运行的系统日志：

```
03-23 00:19:23.480 26457-26457/com.example.david.lifecircle E/TAG: onCreate() is invoked!
03-23 00:19:23.481 26457-26457/com.example.david.lifecircle E/TAG: onStart() is invoked!
03-23 00:19:23.481 26457-26457/com.example.david.lifecircle E/TAG: onResume() is invoked!
03-23 00:19:51.323 26457-26457/com.example.david.lifecircle E/TAG: onPause() is invoked!
03-23 00:19:51.324 26457-26457/com.example.david.lifecircle E/TAG: onSaveInstanceState() is invoked!   Save Text = Save Data
03-23 00:19:51.478 26457-26457/com.example.david.lifecircle E/TAG: onCreate() is invoked!
03-23 00:19:51.488 26457-26457/com.example.david.lifecircle E/TAG: onStart() is invoked!
03-23 00:19:51.490 26457-26457/com.example.david.lifecircle E/TAG: onRestoreInstanceState() is invoked!   Recover Text = Save Data
03-23 00:19:51.490 26457-26457/com.example.david.lifecircle E/TAG: onResume() is invoked!
```

- **情况2：资源内存不足导致低优先级的Activity被杀死**

  这种情况不好模拟，但是其数据存储和恢复过程和情况1完全一致，这里简单的描述一下Activity的优先级情况。Activity的优先级从高到低可以大致分为一下三种：

  （1）前台Activity——正在和用户交互的Activity，优先级最高。

  （2）可见但非前台Activity——比如Activity中弹出了一个对话框，导致Activity可见但无法和用户直接交互。

  （3）后台Activity——已经被暂停或者停止的Activity，优先级最底。

当系统内存不足的时候，系统就会按照上述优先级从低到高来杀死目标Activity。并在后续通过onSaveInstance和onRestoreInstance来存储和恢复数据。

特别提醒的是：如果一个进程中没有四大组件（Activity、Service、ContentProvider、BroadCastReceiver）。那么这个进程就会很快被杀死，因此一些后台工作不适合脱离四大组件而独立运行在后台中，否则很容易被杀死。一般是将后台工作放入Service中从而保证进程有一定的优先级，这样才不会被系统轻易杀死。

三、生命周期的使用

1.常见的生命周期有关问题：

1. onStart和onResume、onPause和onStop从描述上看来差不多，但是他们为什么会分开呢？有什么不同？
2. 两个Activity A和B，从A中启动B，那么B的onResume与A的onPause哪个会先执行呢？
3. onSaveInstance与onRestoreInstance是任何情况下都可以使用的嘛?所有的保存数据和恢复的操作都可以在这对方法中执行？
4. 如上面所说，如何使得在系统配置放生改变后，Activity不被重新创建呢？

2.解析

1、 onStart和onResume、onPause和onStop这两对看起来是差不多，而且很多时候都会同时调用onPause、onStop，然后回到onStart、onResume。但是在一些比较特殊的情况下就不一样了。我们举两种情况，

第一种：前台弹出了一个Dialog，那么这个Dialog的作用只是提醒用户或者让用户输入一个信息等就完毕了，这是一个比较轻量级的任务；

第二种：重新启动另一个Activity界面，转到另一个模块。这时新启动的Activity就不是一个临时或者轻量级的任务了。

这两种情况，第一种一般很快就会返回当前Activity，不会太耗时；第二种可能会很久，在这段时间内系统可能需要启动其他的应用，那么就会产生内存紧张的问题。所以，我认为是要区分这两种情况，才会加入这两对方法。在第一种情况下，可以在onPause中做一些较轻微的数据存储，因为一般很快就会回到当前Activity；第二种情况下，适合在onStop中做一些较重量级的存储。除此之外，我想不到其他的使用了。

2、这个问题可以从源码中得到解答。不过源码太复杂，涉及底层太多（AMS、Binder、ActivityStack、ActivityThread等）。不过可以直接调用生命周期，输出系统日志来得到解答。从下面的日志我们可以看出，的确是要等到A活动的onPause方法之后B才能执行（这里onCreate没有输出日志）：

```
03-23 01:02:31.339 32382-32382/com.example.david.lifecircle E/MainActivity: onCreate() is invoked!
03-23 01:02:31.341 32382-32382/com.example.david.lifecircle E/MainActivity: onStart() is invoked!
03-23 01:02:31.341 32382-32382/com.example.david.lifecircle E/MainActivity: onResume() is invoked!
03-23 01:04:04.005 32382-32382/com.example.david.lifecircle E/MainActivity: onPause() is invoked!
03-23 01:04:04.047 32382-32382/com.example.david.lifecircle E/SecondActivity: onStart() is invoked!
03-23 01:04:04.047 32382-32382/com.example.david.lifecircle E/SecondActivity: onResume() is invoked!
```

3、 onSaveInstance和onRestoreInstance是只有Activity异常销毁的时候才会调用的，所以这里一般执行的是Activity异常销毁时需要保存和恢复的数据；onSaveInstance和onRestoreInstance方法还可以判断Activity是否被重建，但正常情况下是不会调用的。所以正常情况下，还是应该在onPause和onStop方法中保存数据。

4、上面提到，我们可以在AndroidMainfest.xml里，对< activity />增加一个android:configChanges属性，来指定在哪些配置改变的情况下Activity不需要重建。如下所示：

```
android:configChanges="orientation|screenSize"//界面方向以及大小的改变不需要重建
```

我们在AndroidMainfest.xml做如下申明：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.david.lifecircle">

    <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"
            android:configChanges="orientation|screenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

```
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"></activity>
    </application>

</manifest>
```

MainActivity中的部分代码：

```
@Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        Log.e("MainActivity","onConfigurationChanged() is invoked! "+newConfig.orientation);
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.e("MainActivity","onPause() is invoked! ");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.e("MainActivity","onResume() is invoked! ");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.e("MainActivity","onStart() is invoked! ");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.e("MainActivity","onRestart() is invoked! ");
    }
```

点击屏幕旋转，然后来看一下系统日志输出：

```
03-23 01:14:11.357 10361-10361/com.example.david.lifecircle E/MainActivity: onCreate() is invoked!
03-23 01:14:11.359 10361-10361/com.example.david.lifecircle E/MainActivity: onStart() is invoked!
03-23 01:14:11.359 10361-10361/com.example.david.lifecircle E/MainActivity: onResume() is invoked!
03-23 01:14:28.140 10361-10361/com.example.david.lifecircle E/MainActivity: onConfigurationChanged() is invoked! 2
03-23 01:14:38.294 10361-10361/com.example.david.lifecircle E/MainActivity: onConfigurationChanged() is invoked! 1
03-23 01:14:47.531 10361-10361/com.example.david.lifecircle E/MainActivity: onConfigurationChanged() is invoked! 2
```

我们发现，屏幕旋转之后，并没有重新调用生命周期，说明活动并没有被重建。configChanges属性还有许多的值，如：mcc\mnc\local\touchscreen\keyboard等等。