Android 软件开发

认识 Activity

秦兴国

xgqin@guet.edu.cn

计算机与信息安全学院
桂林电子科技大学

2017 年 10 月 27 日

Outline

关于 Activity

Activity 是 Android 四大组件之一，它作为应用与用户交互的入口。熟练的使用 Activity 可以让应用正确应对如下情况：

关于 Activity

Activity 是 Android 四大组件之一，它作为应用与用户交互的入口。熟练的使用 Activity 可以让应用正确应对如下情况：

- 设备屏幕方向变化时，可平滑的进行处理，从而不影响用户使用体验；
- 用户数据在 Activity 间进行跳转时确保不丢失；
- 系统合理的关闭应用进程并释放资源；

Activity 简介

Activity 作为应用的核心组件，Activity 的启动方式、Activity 之间的交互跳转方式是 Android 平台中应用模型的基础。

[1] 一个 Activity 实现应用的一个屏幕

Activity 简介

Activity 作为应用的核心组件，Activity 的启动方式、Activity 之间的交互跳转方式是 Android 平台中应用模型的基础。

Android 系统通过一些列针对 Activity 不同生命周期中对应的回调方法 (callback methods) 的方式，初始化 Activity 实例。

[1] 一个 Activity 实现应用的一个屏幕

Activity 简介

Activity 作为应用的核心组件，Activity 的启动方式、Activity 之间的交互跳转方式是 Android 平台中应用模型的基础。

Android 系统通过一些列针对 Activity 不同生命周期中对应的回调方法 (callback methods) 的方式，初始化 Activity 实例。

Activity 作为应用与用户交互的入口，提供用户绘制 UI 组件的窗口[1]，该窗口通常填充整个设备屏幕 (也可小于屏幕或浮动在其他窗口之上)。用户通常通过子类化 Activity 的方式定义自己的 Activity。

在应用中要使用自定义的 Activity，需要在应用清单文件 (AndroidManifest.xml) 中注册该 Activity 信息，并合理管理 Activity 生命周期。

[1] 一个 Activity 实现应用的一个屏幕

在清单文件中配置 Activity

Activity 对于应用可见，需要在清单文件中注册 Activity 及其相关属性。

在清单文件中配置 Activity

Activity 对于应用可见，需要在清单文件中注册 Activity 及其相关属性。

```
1  <manifest ... >
2  <application ... >
3      <activity android:name=".ExampleActivity" />
4      ...
5      </application ... >
6   ...
7  </manifest >
```

在配置中，仅 *android:name* 属性是必须的，用于指明 Activity 的类名；还可设置 Activity 的其他属性，例如标签 (label)、图标 (icon)、UI 主题 (UI theme) 等。

在清单文件中配置 Activity

声明 Intent filter

Intent filter 是 Android 平台提供的一个强大功能。通过 Intent filter 可以隐式的启动 Activity。

在清单文件中配置 Activity

声明 Intent filter

Intent filter 是 Android 平台提供的一个强大功能。通过 Intent filter 可以隐式的启动 Activity。

```
1  <activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
2    <intent-filter>
3        <action android:name="android.intent.action.SEND" />
4        <category android:name="android.intent.category.DEFAULT" />
5        <data android:mimeType="text/plain" />
6    </intent-filter>
7  <activity>
```

在清单文件中配置 Activity

声明 Intent filter

Intent filter 是 Android 平台提供的一个强大功能。通过 Intent filter 可以隐式的启动 Activity。

```
1  <activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
2    <intent-filter>
3        <action android:name="android.intent.action.SEND" />
4        <category android:name="android.intent.category.DEFAULT" />
5        <data android:mimeType="text/plain" />
6    </intent-filter>
7  <activity>
```

```
1    // Create the text message with a string
2    Intent sendIntent = new Intent();
3    sendIntent.setAction(Intent.ACTION_SEND);
4    sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
5    sendIntent.setType("text/plain");
6    // Start the activity
7    startActivity(sendIntent);
```

在清单文件中配置 Activity

声明 Activity 权限配置

可通过清单文件的 *<activity>* 元素配置哪些应用可以访问该 Activity。当要一个父 Activity 启动子 Activity 时，需要确保两个 Activity 拥有相同的权限配置[2]。

[2]如果在指定的 Activity 中使用了 *<uses-permission>* 元素指定该 Activity 的权限，则调用该 Activity 的父 Activity 也必须拥有与之匹配的权限

在清单文件中配置 Activity

声明 Activity 权限配置

可通过清单文件的 *<activity>* 元素配置哪些应用可以访问该 Activity。当要一个父 Activity 启动子 Activity 时，需要确保两个 Activity 拥有相同的权限配置[2]。

```
1  <manifest>
2    <activity android:name="....."
3    android:permission="com.google.socialapp.permission.SHARE_POST"
4     />
5  </mainfest>
```

[2]如果在指定的 Activity 中使用了 *<uses-permission>* 元素指定该 Activity 的权限，则调用该 Activity 的父 Activity 也必须拥有与之匹配的权限

在清单文件中配置 Activity

声明 Activity 权限配置

可通过清单文件的 *<activity>* 元素配置哪些应用可以访问该 Activity。当要一个父 Activity 启动子 Activity 时，需要确保两个 Activity 拥有相同的权限配置[2]。

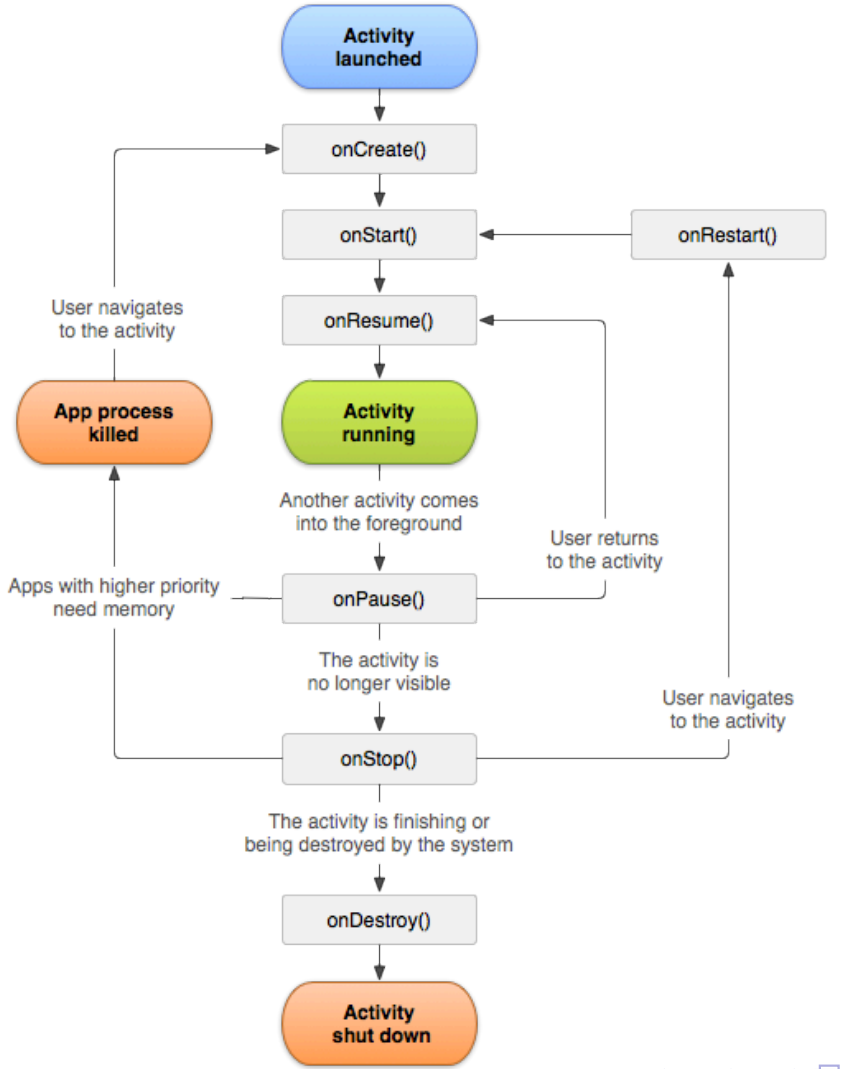
```
1  <manifest>
2    <activity android:name="....."
3    android:permission="com.google.socialapp.permission.SHARE_POST"
4     />
5  </mainfest>
```

```
1  <manifest>
2   <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
3  </manifest>
```

[2]如果在指定的 Activity 中使用了 *<uses-permission>* 元素指定该 Activity 的权限，则调用该 Activity 的父 Activity 也必须拥有与之匹配的权限

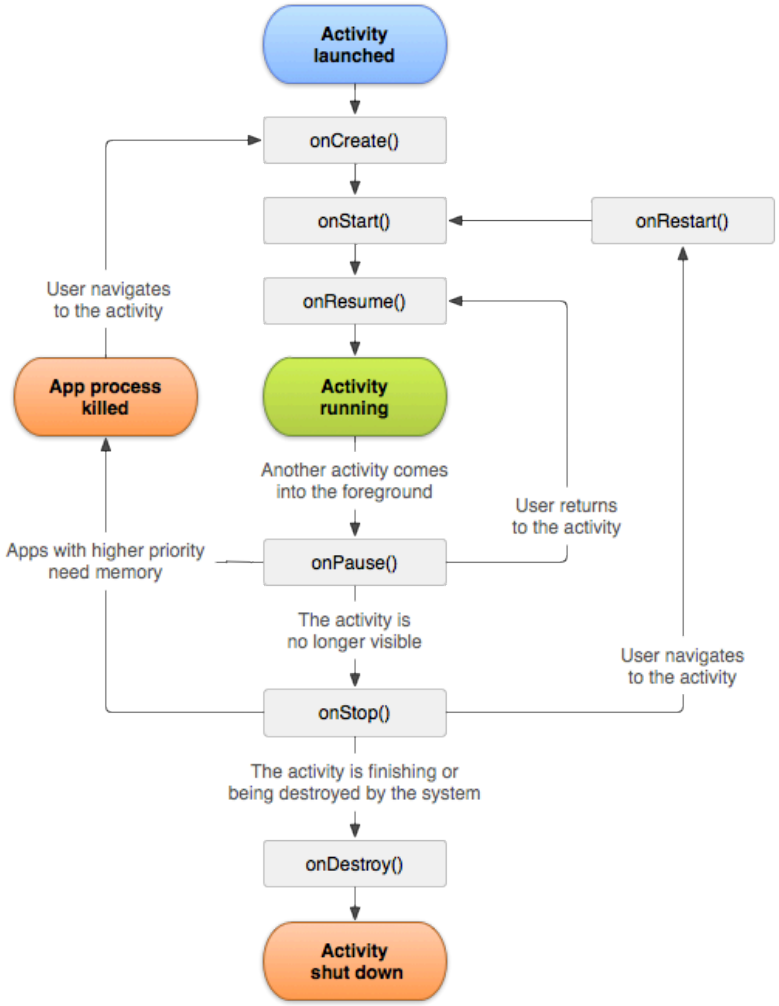
Activity 生命周期与状态

在 Activity 整个生命周期中，当用户在不同应用间切换、在同一应用的不同 Activity 之间浏览时，Activity 状态将发生转换。



Activity 生命周期与状态

在 Activity 整个生命周期中，当用户在不同应用间切换、在同一应用的不同 Activity 之间浏览时，Activity 状态将发生转换。



Activity 提供一系列的回调方法供开发者处理 Activity 状态之间的转换。

- onCreate
- onStart
- onResume
- onPause
- onStop
- onDestroy
- onRestart

Activity 生命周期与状态

通过 Activity 提供的一系列回调方法，可以指定当用户离开或重新进入 Activity 时的行为。合理的使用这些回调函数，可以避免应用出现以下情况：

Activity 生命周期与状态

通过 Activity 提供的一系列回调方法，可以指定当用户离开或重新进入 Activity 时的行为。合理的使用这些回调函数，可以避免应用出现以下情况：

- 当用户在使用你的应用时接听电话，或切换到其他应用时，造成应用崩溃；

Activity 生命周期与状态

通过 Activity 提供的一系列回调方法，可以指定当用户离开或重新进入 Activity 时的行为。合理的使用这些回调函数，可以避免应用出现以下情况：

- 当用户在使用你的应用时接听电话，或切换到其他应用时，造成应用崩溃；
- 当用户不再使用应用时，该应用还占据额外的系统资源；

Activity 生命周期与状态

通过 Activity 提供的一系列回调方法，可以指定当用户离开或重新进入 Activity 时的行为。合理的使用这些回调函数，可以避免应用出现以下情况：

- 当用户在使用你的应用时接听电话，或切换到其他应用时，造成应用崩溃；
- 当用户不再使用应用时，该应用还占据额外的系统资源；
- 当用户离开应用并再返回应用后，丢失用户在该应用的状态信息；

Activity 生命周期与状态

通过 Activity 提供的一系列回调方法，可以指定当用户离开或重新进入 Activity 时的行为。合理的使用这些回调函数，可以避免应用出现以下情况：

- 当用户在使用你的应用时接听电话，或切换到其他应用时，造成应用崩溃；
- 当用户不再使用应用时，该应用还占据额外的系统资源；
- 当用户离开应用并再返回应用后，丢失用户在该应用的状态信息；
- 当屏幕旋转时，丢失用户使用应用的状态信息或应用崩溃；

Activity 生命周期与状态

当用户通过导航栏，返回按钮等方式离开一个 Activity 时，Android 系统会拆解 (dismantle) 该 Activity。在某种情况下，做部分拆解。Activity 还驻留在内存中 (例如，当用户切换到其他应用时)，并可再次回到前台 (foreground) 运行。

Activity 生命周期与状态

当用户通过导航栏，返回按钮等方式离开一个 Activity 时，Android 系统会拆解 (dismantle) 该 Activity。在某种情况下，做部分拆解。Activity 还驻留在内存中 (例如，当用户切换到其他应用时)，并可再次回到前台 (foreground) 运行。

当用户返回到该 Activity 时，Activity 应该能够从用户离开该 Activity 时的状态中恢复运行。

Activity 生命周期与状态

当用户通过导航栏，返回按钮等方式离开一个 Activity 时，Android 系统会拆解 (dismantle) 该 Activity。在某种情况下，做部分拆解。Activity 还驻留在内存中 (例如，当用户切换到其他应用时)，并可再次回到前台 (foreground) 运行。

当用户返回到该 Activity 时，Activity 应该能够从用户离开该 Activity 时的状态中恢复运行。

Android 系统结束某一应用进程时，也会考虑到在该进程当中的 Activity 实例的状态。

Activity 生命周期与状态

当用户通过导航栏，返回按钮等方式离开一个 Activity 时，Android 系统会拆解 (dismantle) 该 Activity。在某种情况下，做部分拆解。Activity 还驻留在内存中 (例如，当用户切换到其他应用时)，并可再次回到前台 (foreground) 运行。

当用户返回到该 Activity 时，Activity 应该能够从用户离开该 Activity 时的状态中恢复运行。

Android 系统结束某一应用进程时，也会考虑到在该进程当中的 Activity 实例的状态。

是否实现 Activity 所提供的与生命周期相关的所有回调方法，取决于这个 Activity 的业务逻辑复杂程度。

Activity 生命周期与状态

onCreate() 方法

onCreate() 回调方法是在子类化 Activity 时必须实现的方法。当一个 Activity 实例被创建时，该 Activity 实例处于*Created*状态，从而执行 **onCreate()** 方法[3]。

[3] **onCreate()** 方法接收一个 Bundle 类型的 savedInstanceState 参数，该参数包含 Activity 前一次被保存时的状态；如果该 Activity 未被初始化过，则该参数为空。

Activity 生命周期与状态

onCreate() 方法

onCreate() 回调方法是在子类化 Activity 时必须实现的方法。当一个 Activity 实例被创建时，该 Activity 实例处于*Created*状态，从而执行 **onCreate()** 方法[3]。

在该方法中，需要执行与该 Activity 相关的一些初始化工作，这些初始化工作在整个 Activity 实例生命周期内应仅执行一次。

- 绑定数据；
- 初始化后台线程；
- 实例化某些类成员变量；

[3] **onCreate()** 方法接收一个 Bundle 类型的 savedInstanceState 参数，该参数包含 Activity 前一次被保存时的状态；如果该 Activity 未被初始化过，则该参数为空。

Activity 生命周期与状态 I

onCreate() 方法举例

```
1   TextView mTextView;
2
3   // some transient state for the activity instance
4   String mGameState;
5
6   @Override
7   public void onCreate(Bundle savedInstanceState) {
8     // call the super class onCreate to complete the creation of activity like
9     // the view hierarchy
10    super.onCreate(savedInstanceState);
11
12    // recovering the instance state
13    if (savedInstanceState != null) {
14        mGameState = savedInstanceState.getString(GAME_STATE_KEY);
15    }
16
17    setContentView(R.layout.main_activity);
18
19    mTextView = (TextView) findViewById(R.id.text_view);
20  }
```

Activity 生命周期与状态 II

onCreate() 方法举例

```
21
22    // This callback is called only when there is a saved instance previously saved using
23    // onSaveInstanceState(). We restore some state in onCreate() while we can optionally resto
24    // other state here, possibly usable after onStart() has completed.
25    // The savedInstanceState Bundle is same as the one used in onCreate().
26    @Override
27    public void onRestoreInstanceState(Bundle savedInstanceState) {
28      mTextView.setText(savedInstanceState.getString(TEXT_VIEW_KEY));
29    }
30
31    // invoked when the activity may be temporarily destroyed, save the instance state here
32    @Override
33    public void onSaveInstanceState(Bundle outState) {
34      outState.putString(GAME_STATE_KEY, mGameState);
35      outState.putString(TEXT_VIEW_KEY, mTextView.getText());
36
37      // call superclass to save any view hierarchy
38      super.onSaveInstanceState(outState);
39    }
```

Activity 生命周期与状态

onStart() 方法

当 **onCreate()** 方法执行结束后，Activity 将进入*Started*状态，Android 系统将调用 **onStart()**、**onResume()** 方法。

Activity 生命周期与状态

onStart() 方法

当 **onCreate()** 方法执行结束后，Activity 将进入*Started*状态，Android 系统将调用 **onStart()**、**onResume()** 方法。

Android 系统在完成 **onStart()** 方法后，Activity 实例将对用户可见，也就是说应用已经做好了将该 Activity 实例放入前台运行，并相应用户交互。

在该方法中，通常处理与 UI 相关的逻辑代码，以及注册 Broadcast receiver 用于监听广播并通过 ActivityUI 反馈这类广播。

Activity 生命周期与状态

onStart() 方法

当 **onCreate()** 方法执行结束后，Activity 将进入*Started*状态，Android 系统将调用 **onStart()**、**onResume()** 方法。

Android 系统在完成 **onStart()** 方法后，Activity 实例将对用户可见，也就是说应用已经做好了将该 Activity 实例放入前台运行，并相应用户交互。

在该方法中，通常处理与 UI 相关的逻辑代码，以及注册 Broadcast receiver 用于监听广播并通过 ActivityUI 反馈这类广播。

与 **onCreate()** 类似，**onStart()** 方法也需要被快速执行，Activity 无法驻留在*Started*状态中。一旦完成该方法调用，Activity 进入*Resumed*状态，并调用 **onResume()** 方法。

Activity 生命周期与状态

onResume() 方法

当 Activity 进入*Resumed*状态时，表示 Activity 进入前台运行，Android 系统将调用 **onResume()** 回调函数。

[4] 例如用户接听电话、用户切换至其他 Activity、设备屏幕关闭等

Activity 生命周期与状态

onResume() 方法

当 Activity 进入*Resumed*状态时，表示 Activity 进入前台运行，Android 系统将调用 **onResume()** 回调函数。

Activity 与用户进行交互式将处于该状态中，直到其他事件[4]发生导致用户切换至其他应用。

当发生这类事件时，当前的 Activity 进入*Paused*状态，Android 系统调用 **onPause()** 回调函数。

[4]例如用户接听电话、用户切换至其他 Activity、设备屏幕关闭等

Activity 生命周期与状态

onResume() 方法

当 Activity 从*Paused*状态，返回至*Resumed*状态是，**onResume() 方法将再次被调用。**

Activity 生命周期与状态

onResume() 方法

当 Activity 从*Paused*状态，返回至*Resumed*状态是，**onResume() 方法将再次被调用。**

因此如果应用的组件在 onPause() 方法中被释放时，你应该在 onResume() 方法中再次进行初始化：

Activity 生命周期与状态

onResume() 方法

当 Activity 从*Paused*状态，返回至*Resumed*状态是，**onResume() 方法将再次被调用。**

因此如果应用的组件在 onPause() 方法中被释放时，你应该在 onResume() 方法中再次进行初始化：

```
1   @Override
2   public void onResume() {
3     super.onResume();  // Always call the superclass method first
4
5
6     // Get the Camera instance as the activity achieves full user focus
7     if (mCamera == null) {
8         initializeCamera(); // Local method to handle camera init
9     }
10  }
```

Activity 生命周期与状态

onPause() 方法

当用户从一个 Activity 离开时，**onPause() 方法将被调用**。

Activity 进入*Paused*状态的原因有多种：

Activity 生命周期与状态

onPause() 方法

当用户从一个 Activity 离开时，**onPause() 方法将被调用**。Activity 进入*Paused*状态的原因有多种：

- 某些事件发生，从而打断应用的执行；
- Android 7.0 以上版本，多个应用可以运行于多窗口模式，但仅有一个应用可获得焦点，系统将其他应用置为 *Paused* 状态；
- 半透明 (semi-transparent) 的 Activity(例如，对话框) 覆盖了当前 Activity 时，当前 Activity 处于半可见，但并未获得焦点，则该 Activity 处于*Paused*状态；

Activity 生命周期与状态

onPause() 方法

当 Activity 处于*Paused*状态时，应当将对电池电量影响较大的一些资源进行释放，并在 **onResume()** 方法中再次进行初始化。

Activity 生命周期与状态

onPause() 方法

当 Activity 处于*Paused*状态时，应当将对电池电量影响较大的一些资源进行释放，并在 **onResume()** 方法中再次进行初始化。

```
1   @Override
2   public void onPause() {
3     super.onPause();  // Always call the superclass method first
4
5
6     // Release the Camera because we don't need it when paused
7     // and other activities might need to use it.
8     if (mCamera != null) {
9         mCamera.release();
10        mCamera = null;
11    }
12  }
```

Activity 生命周期与状态

onStop() 方法

当 Activity 对于用户而言不可见时，其进入*Stopped*状态，系统调用 **onStop()** 方法。

Activity 生命周期与状态

onStop() 方法

当 Activity 对于用户而言不可见时，其进入*Stopped*状态，系统调用 **onStop()** 方法。

Activity 进入*Stopped*状态有多种原因，例如：

- 新的 Activity 启动，并进入前台覆盖设备屏幕；
- Activity 自身完成运行，系统将调用 **onStop()**；

Activity 生命周期与状态

onStop() 方法

当 Activity 对于用户而言不可见时，其进入*Stopped*状态，系统调用 **onStop()** 方法。

Activity 进入*Stopped*状态有多种原因，例如：

- 新的 Activity 启动，并进入前台覆盖设备屏幕；
- Activity 自身完成运行，系统将调用 **onStop()**；

当 Activity 进入*Stopped*状态时，Activity 对象实例还驻留在内存当中，用于维护 Activity 的状态及成员信息 (UI 控件信息等)。

Activity 生命周期与状态

onStop() 方法

当 Activity 对于用户而言不可见时，其进入*Stopped*状态，系统调用 **onStop()** 方法。

Activity 进入*Stopped*状态有多种原因，例如：

- 新的 Activity 启动，并进入前台覆盖设备屏幕；
- Activity 自身完成运行，系统将调用 **onStop()**；

当 Activity 进入*Stopped*状态时，Activity 对象实例还驻留在内存当中，用于维护 Activity 的状态及成员信息 (UI 控件信息等)。

当 Activity 重新进入*Resumed*状态时，Activity 将恢复其状态及成员信息；应用不需要重新初始化从其他状态进入到*Resumed*状态所对应的回调函数中创建的组件资源。

Activity 生命周期与状态

onStop() 方法

在 **onStop()** 方法中，应用应该释放掉用户不再使用的绝大部分资源；在该方法中，也经常执行一些与 CPU 资源消耗相对较多的一些操作，例如数据保存等。

```
1   @Override
2   protected void onStop() {
3     super.onStop();
4
5     // save the note's current draft, because the activity is stopping
6     // and we want to be sure the current note progress isn't lost.
7     ContentValues values = new ContentValues();
8     values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
9     values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());
10
11    // do this update in background on an AsyncQueryHandler or equivalent
12    mAsyncQueryHandler.startUpdate (
13            mToken,   // int token to correlate calls
14            null,     // cookie, not used here
15            mUri,     // The URI for the note to update.
16            values,   // The map of column names and new values to apply to them.
17            null      // No SELECT criteria are used
```

Activity 生命周期与状态

onDestory() 方法

Activity 从*Stopped*状态，可以从新进入前台与用户进行交互，或者完成运行从而销毁；这时，Android 系统分别调用 **onRestart() 方法以及 onDestroy() 方法**。

Activity 生命周期与状态

onDestory() 方法

Activity 从*Stopped*状态，可以从新进入前台与用户进行交互，或者完成运行从而销毁；这时，Android 系统分别调用 **onRestart()** 方法以及 **onDestroy()** 方法。

onDestroy() 方法是 Activity 在销毁时所能被执行的最后一个方法。应用可以通过调用 **finish()** 方法，从而通知系统结束该 Activity，或者 Android 将暂时性销毁包含该 Activity 的进程 (用于回收资源)。

Activity 生命周期与状态

onDestory() 方法

Activity 从*Stopped*状态，可以从新进入前台与用户进行交互，或者完成运行从而销毁；这时，Android 系统分别调用 **onRestart()** 方法以及 **onDestroy()** 方法。

onDestroy() 方法是 Activity 在销毁时所能被执行的最后一个方法。应用可以通过调用 **finish()** 方法，从而通知系统结束该 Activity，或者 Android 将暂时性销毁包含该 Activity 的进程 (用于回收资源)。

应用可以通过 **isFinishing()** 方法来区分这两类状态；

Activity 生命周期与状态

onDestory() 方法

Activity 从*Stopped*状态，可以从新进入前台与用户进行交互，或者完成运行从而销毁；这时，Android 系统分别调用 **onRestart()** 方法以及 **onDestroy()** 方法。

onDestroy() 方法是 Activity 在销毁时所能被执行的最后一个方法。应用可以通过调用 **finish()** 方法，从而通知系统结束该 Activity，或者 Android 将暂时性销毁包含该 Activity 的进程 (用于回收资源)。

应用可以通过 **isFinishing()** 方法来区分这两类状态；

当设备屏幕发生横竖屏切换时，Android 系统也将调用 **onDestroy()** 方法，并紧接着调用 **onCreate()** 方法重新创建该应用进程 (以及该进程包含的组件) 用以适应新的屏幕状态。

Activity 何时被销毁及释放

Android 系统并不直接销毁一个 Activity 实例，相反，Android 系统直接销毁运行该实例的进程以及运行在该进程当中的其他组件。

Activity 何时被销毁及释放

Android 系统并不直接销毁一个 Activity 实例，相反，Android 系统直接销毁运行该实例的进程以及运行在该进程当中的其他组件。

Android 系统在需要释放更多内存时会进行进程释放操作。进程被释放的可能性取决于当前进程运行的状态，而进程运行状态则由在该进程中运行的 Activity 状态决定。

Activity 何时被销毁及释放

Android 系统并不直接销毁一个 Activity 实例，相反，Android 系统直接销毁运行该实例的进程以及运行在该进程当中的其他组件。

Android 系统在需要释放更多内存时会进行进程释放操作。进程被释放的可能性取决于当前进程运行的状态，而进程运行状态则由在该进程中运行的 Activity 状态决定。

被释放概率	进程状态	Activity 状态
低	前台运行 (拥有或即将拥有焦点)	Created Started Resumed
中	后台运行 (失去焦点)	Paused
高	后台运行 (不可见)	Stoped
	空	Destroyed

Activity 之间的状态切换与协作

当通过一个 Activity(A) 启动另外一个 Activity(B) 时，两个 Activity 都经历生命周期中的状态切换，Activity A 停止工作，并进入 *Paused* 或 *Stopped* 状态，Activity B 则被创建。

Activity 之间的状态切换与协作

当通过一个 Activity(A) 启动另外一个 Activity(B) 时，两个 Activity 都经历生命周期中的状态切换，Activity A 停止工作，并进入 *Paused* 或 *Stopped* 状态，Activity B 则被创建。

如果两个 Activity 之间通过存储器等介质共享数据，则需要了解的一点是，直到 Activity B 被创建完成 (处在 *Resumed* 状态) 后，Activity A 才进入 *Stopped* 状态。

Activity 之间的状态切换与协作

当通过一个 Activity(A) 启动另外一个 Activity(B) 时，两个 Activity 都经历生命周期中的状态切换，Activity A 停止工作，并进入 *Paused* 或 *Stopped* 状态，Activity B 则被创建。

如果两个 Activity 之间通过存储器等介质共享数据，则需要了解的一点是，直到 Activity B 被创建完成 (处在 *Resumed* 状态) 后，Activity A 才进入 *Stopped* 状态。

换言之，Activity B 的启动过程与 Activity A 的停止过程是相互重叠 (overlaps) 的。

Activity 之间的状态切换与协作

当通过一个 Activity(A) 启动另外一个 Activity(B) 时，两个 Activity 都经历生命周期中的状态切换，Activity A 停止工作，并进入 *Paused* 或 *Stopped* 状态，Activity B 则被创建。

如果两个 Activity 之间通过存储器等介质共享数据，则需要了解的一点是，直到 Activity B 被创建完成 (处在 *Resumed* 状态) 后，Activity A 才进入 *Stopped* 状态。

换言之，Activity B 的启动过程与 Activity A 的停止过程是相互重叠 (overlaps) 的。

- Activity A 的 **onPause()** 方法被执行；
- Activity B 的 **onCreate()**、**onStart()**、**onResume()** 方法依次被执行 (此时，Activity B 获得用户焦点)；
- 如果 Activity A 不可见，则其 **onStop()** 方法被执行；

Activity 何时被销毁

在正常的使用场景下，Activity 有几种被销毁的情况，例如：用户点击返回按钮，或者 Activity 通过 **finish()** 方法触发其析构销毁方法。

Activity 何时被销毁

在正常的使用场景下，Activity 有几种被销毁的情况，例如：用户点击返回按钮，或者 Activity 通过 **finish()** 方法触发其析构销毁方法。

Android 系统也有可能会在需要回收释放内存时将状态为 *Stopped* 并且长时间未被执行的 Activity 所在的进程销毁。

Activity 何时被销毁

在正常的使用场景下，Activity 有几种被销毁的情况，例如：用户点击返回按钮，或者 Activity 通过 **finish()** 方法触发其析构销毁方法。

Android 系统也有可能会在需要回收释放内存时将状态为 *Stopped* 并且长时间未被执行的 Activity 所在的进程销毁。

当由于系统的原因，导致 Activity 被销毁时，Android 系统将会记录下 Activity 被销毁时的各种状态及信息，从而在用户再次切换回该 Activity 时，系统将使用保存的状态及信息重新创建该 Activity 实例。

Activity 何时被销毁

在正常的使用场景下，Activity 有几种被销毁的情况，例如：用户点击返回按钮，或者 Activity 通过 **finish()** 方法触发其析构销毁方法。

Android 系统也有可能会在需要回收释放内存时将状态为 *Stopped* 并且长时间未被执行的 Activity 所在的进程销毁。

当由于系统的原因，导致 Activity 被销毁时，Android 系统将会记录下 Activity 被销毁时的各种状态及信息，从而在用户再次切换回该 Activity 时，系统将使用保存的状态及信息重新创建该 Activity 实例。

系统用于恢复 Acitivity 前一个实例状态的信息的数据称为 *instance state*，其为存储于 **Bundle** 对象的键值对集合。

instance state 用途

默认情况下，Android 系统使用 *instance state*(Bundle 对象) 存储在 Activity 布局当中的每一个视图 (View) 对象的信息 (例如，EditText 控件中所输入的文本信息)。

instance state 用途

默认情况下，Android 系统使用 *instance state*(Bundle 对象) 存储在 Activity 布局当中的每一个视图 (View) 对象的信息 (例如，EditText 控件中所输入的文本信息)。

因此，Activity 实例被销毁并重新创建时，其布局状态及信息将从上一个实例状态中进行恢复，而这些不需要开发人员做任何操作。

instance state 用途

默认情况下，Android 系统使用 *instance state*(Bundle 对象) 存储在 Activity 布局当中的每一个视图 (View) 对象的信息 (例如，EditText 控件中所输入的文本信息)。

因此，Activity 实例被销毁并重新创建时，其布局状态及信息将从上一个实例状态中进行恢复，而这些不需要开发人员做任何操作。

但当 Activity 需要保存更多状态信息并期望在实例被重新创建时进行使用，则需要对这类信息在销毁时进行存储，并在重新创建时再进行恢复。

在销毁前保存 Activity 状态

当 Activity 进入*Stopped*状态时，Android 系统将调用 **onSaveInstanceState()** 方法，在该方法中可通过键值对的形式对保存 Activity 所需的状态数据信息；

在销毁前保存 Activity 状态

当 Activity 进入*Stopped*状态时，Android 系统将调用 **onSaveInstanceState() 方法，在该方法中可通过键值对的形式对保存 Activity 所需的状态数据信息；**

该方法默认保存与 Activity 布局中视图相关的临时性信息，例如：EditText 控件中输入的文本，ListView 控件滚动条的当前位置等。

在销毁前保存 Activity 状态

如果要保存 Activity 的额外信息，就必须覆盖该方法，在实现该方法时，需要调用父类的方法，从而确保上述信息能正确被保存。

```
1   static final String STATE_SCORE = "playerScore";
2   static final String STATE_LEVEL = "playerLevel";
3   ...
4
5   @Override
6   public void onSaveInstanceState(Bundle savedInstanceState) {
7   // Save the user's current game state
8   savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
9   savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);
10
11  // Always call the superclass so it can save the view hierarchy state
12  super.onSaveInstanceState(savedInstanceState);
13  }
```

恢复 Activity 状态

当 Activity 实例被重新创建时 **onCreate()**、**onRestoreInstanceState()** 方法都会收到同一个包含实例状态及数据的 Bundle 对象[5]。可通过该对象恢复前一个实例的状态。

[5] 但如果 Activity 是被创建的新实例，在 **onCreate()** 方法中的 Bundle 对象参数的值为 *null* 因此如果需要在 **onCreate()** 方法中使用该对象时，需确认其是否为*null*

恢复 Activity 状态

当 Activity 实例被重新创建时 **onCreate()**、**onRestoreInstanceState()** 方法都会收到同一个包含实例状态及数据的 Bundle 对象[5]。可通过该对象恢复前一个实例的状态。

```
1   protected void onCreate(Bundle savedInstanceState) {
2     super.onCreate(savedInstanceState); // Always call the superclass first
3
4     // Check whether we're recreating a previously destroyed instance
5     if (savedInstanceState != null) {
6         // Restore value of members from saved state
7         mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
8         mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
9     } else {
10        // Probably initialize members with default values for a new instance
11    }
12    ...
13  }
```

[5] 但如果 Activity 是被创建的新实例，在 **onCreate()** 方法中的 Bundle 对象参数的值为 *null* 因此如果需要在 **onCreate()** 方法中使用该对象时，需确认其是否为*null*

恢复 Activity 状态

除了通过 **onCreate()** 方法恢复 Activity 实例状态，还可以通过实现 **onRestoreInstanceState()** 的方式来实现恢复 Activity 状态；该方法会在 **onStart()** 执行后被调用；

[6] 需要调用父类同一方法，从而执行默认的恢复 Activity 实例中视图状态

恢复 Activity 状态

除了通过 **onCreate()** 方法恢复 Activity 实例状态，还可以通过实现 **onRestoreInstanceState()** 的方式来实现恢复 Activity 状态；该方法会在 **onStart()** 执行后被调用；

Android 系统会判断 Bundle 对象中是否保存了前一个实例的状态信息，从而决定是否执行 **onRestoreInstanceState()** 方法[6]，因此在该方法中不需要检查 Bundle 对象是否为*null*。

[6] 需要调用父类同一方法，从而执行默认的恢复 Activity 实例中视图状态

恢复 Activity 状态

除了通过 **onCreate()** 方法恢复 Activity 实例状态，还可以通过实现 **onRestoreInstanceState()** 的方式来实现恢复 Activity 状态；该方法会在 **onStart()** 执行后被调用；

Android 系统会判断 Bundle 对象中是否保存了前一个实例的状态信息，从而决定是否执行 **onRestoreInstanceState()** 方法[6]，因此在该方法中不需要检查 Bundle 对象是否为*null*。

```
1  public void onRestoreInstanceState(Bundle savedInstanceState) {
2    // Always call the superclass so it can restore the view hierarchy
3    super.onRestoreInstanceState(savedInstanceState);
4
5
6    // Restore state members from saved instance
7    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
8    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
9  }
```

[6]需要调用父类同一方法，从而执行默认的恢复 Activity 实例中视图状态