第二章 Activity

北京邮电大学 计算机学院
刘伟
w.liu@foxmail.com

本章重点

- 掌握Activity的创建及生命周期
- 熟练使用Android资源
- 熟悉样式和主题的使用

2.1 Android组件

- Android系统四大组件 （调用的基本模块）
  - Activity
    - Android程序的呈现层，显示可视化的用户界面，并接收与用户交互所产生的界面事件
    - Android应用程序可以包含一个或多个Activity，一般需要指定一个程序启动时显示的Activity
  - Service
    - Service一般用于没有用户界面，但需要长时间在后台运行的应用
    - 可公开Service的程序接口，供其他进程调用

2.1 Android组件

- Android系统四大组件 （调用的基本模块）
  - BroadcaseReceiver
    - 用来接收广播消息的组件，不包含任何用户界面
    - 可以启动Activity或者Notification通知用户接收到重要信息
      - Notification能够通过多种方法提示用户，包括闪动背景灯、震动设备、发出声音或在状态栏上放置一个图标
  - ContentProvider
    - 是Android系统提供的一种标准的共享数据的机制，其他程序通过ContentProvider访问程序的私有数据
    - Android系统内部提供一些内置的ContentProvider，能够为应用程序提供重要的数据信息
      - 联系人信息
      - 通话记录

2.1.1 Activity的概念

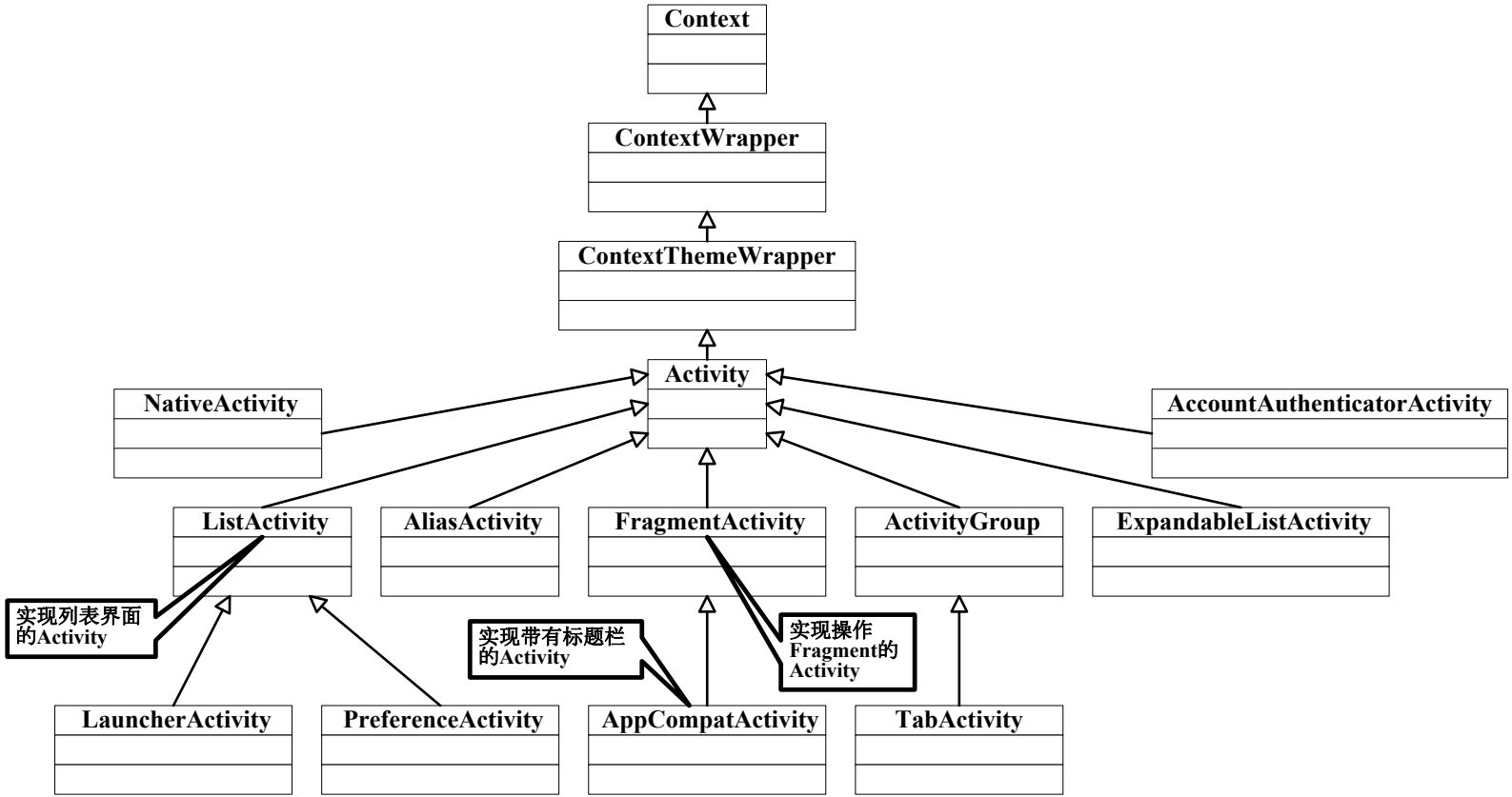
- Activity是Android组件中最基本也是最为常见用的四大组件（Activity，Service服务,Content Provider内容提供者，BroadcastReceiver广播接收器）之一 。

- Activity用于提供可视化用户界面的组件，是用户操作的可视化界面；它为用户提供了一个完成操作指令的窗口，可以与用户进行交互来完成某项任务。

- Activity中所有操作都与用户密切相关，是一个负责与用户交互的组件，当我们创建完毕Activity之后，需要调用setContentView()方法来完成界面的显示。

- 在一个android应用中，一个Activity通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。

2.1.1 Activity的继承关系



2.1.2 Android的创建

两种创建方式

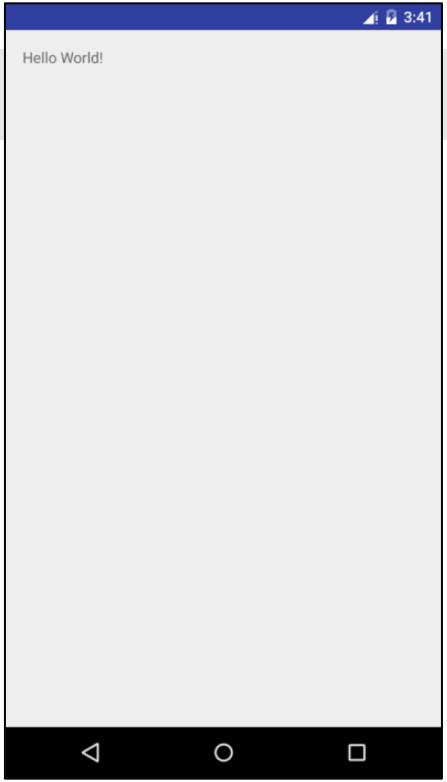
1）包名处点击右键选择【New】→【Activity】→【Empty Activity】选项，填写Activity信息，完成创建。

2）包名处点击右键选择【New】→【Java Class】选项，填写Java类名，完成创建。在该类中继承AppCompatActivity，并在清单文件中进行注册，完成Activity的创建。

2.1.2 创建Activity

通过继承Activity基类的方式实现自定义的BaseActivity类

```
import android.app.Activity;
import android.os.Bundle;
public class BaseActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
  }
}
```



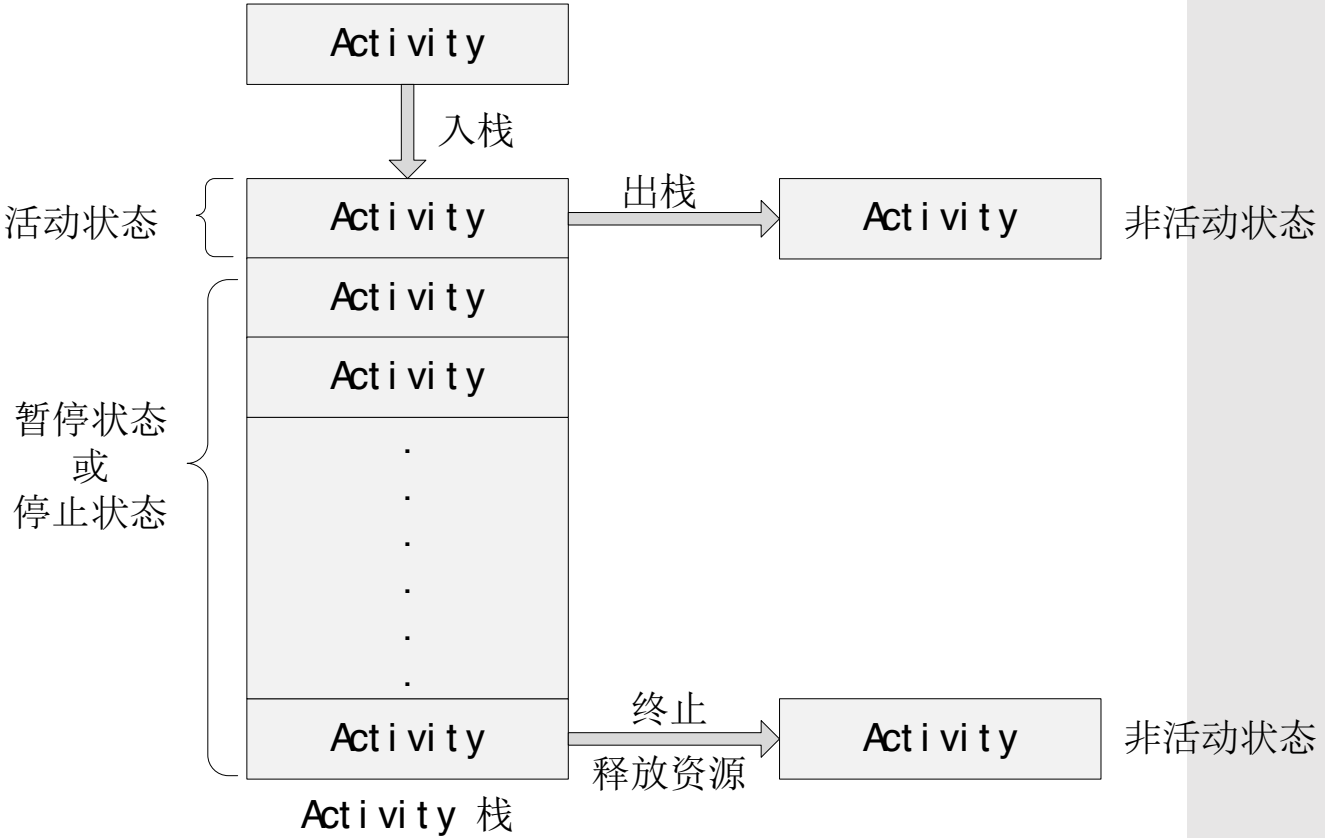
2.1.2 创建Activity

通过继承Activity基类的方式实现自定义的BaseActivity类

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

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

2.2 Activity的生命周期

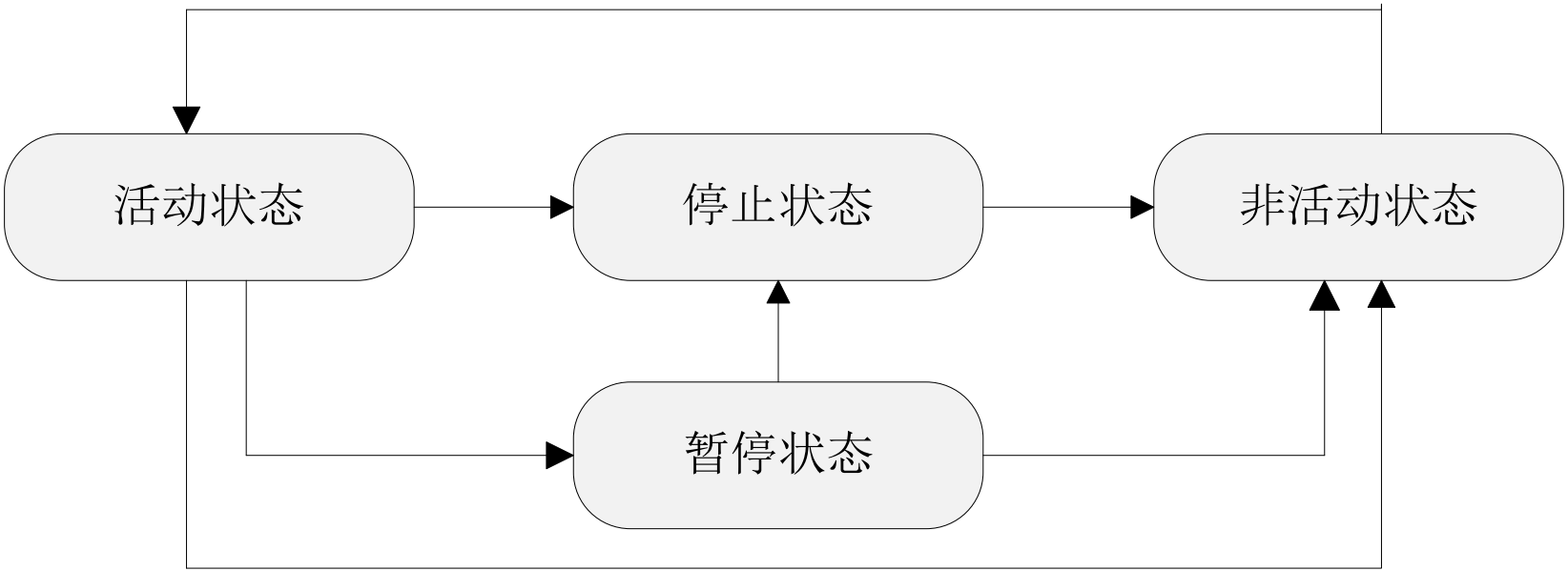


在Android中会维持一个Activity Stack（Activity栈），当一个新的Activity创建时，它就会放到栈顶，这个Activity就处于运行状态。当再有一个新的Activity被创建后，会重新压入栈顶，而之前的Activity则会在这个新的Activity底下，就像枪梭压入子弹一样。而且之前的Activity就会进入后台。

2.2.1 Activity的四个状态

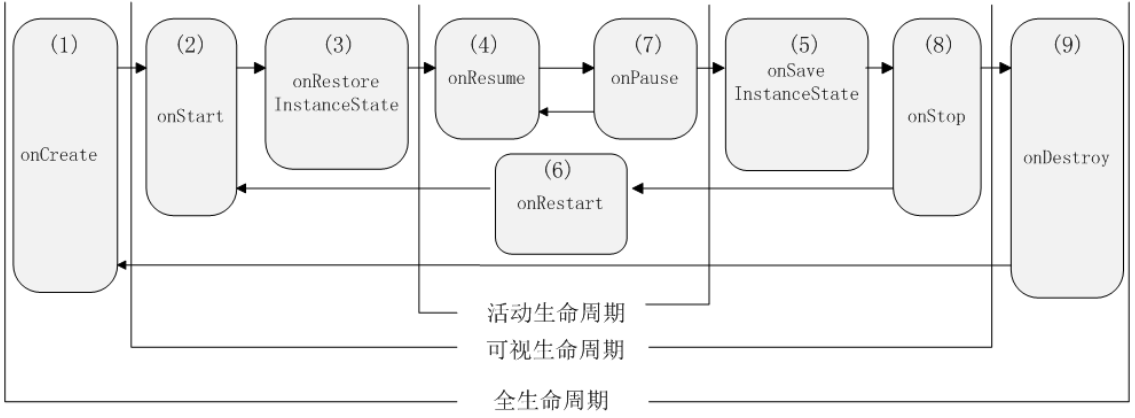
- Active/Running运行状态
  - 一个新 Activity 启动入栈后，它显示在屏幕最前端，处于栈的最顶端（Activity栈顶），此时它处于可见并可和用户交互的激活状态,叫做活动状态或者运行状态（active or running）。
- Paused暂停状态
  - 当 Activity失去焦点， 被一个新的非全屏的Activity 或者一个透明的Activity 被放置在栈顶，此时的状态叫做暂停状态（Paused）。此时它依然与窗口管理器保持连接，Activity依然保持活力（保持所有的状态，成员信息，和窗口管理器保持连接），但是在系统内存极端低下的时候将被强行终止掉。所以它仍然可见，但已经失去了焦点故不可与用户进行交互。
- Stopped停止状态
  - 如果一个Activity被另外的Activity完全覆盖掉，叫做停止状态（Stopped）。它依然保持所有状态和成员信息，但是它不再可见，所以它的窗口被隐藏，当系统内存需要被用在其他地方的时候，Stopped的Activity将被强行终止掉。
- Killed销毁状态
  - 如果一个Activity是Paused或者Stopped状态，系统可以将该Activity从内存中删除，Android系统采用两种方式进行删除，要么要求该Activity结束，要么直接终止它的进程。当该Activity再次显示给用户时，它必须重新开始和重置前面的状态。

2.2.1 Activity的四个状态



2.2.2 Activity的三个生命周期

■ Activity事件回调函数的调用顺序

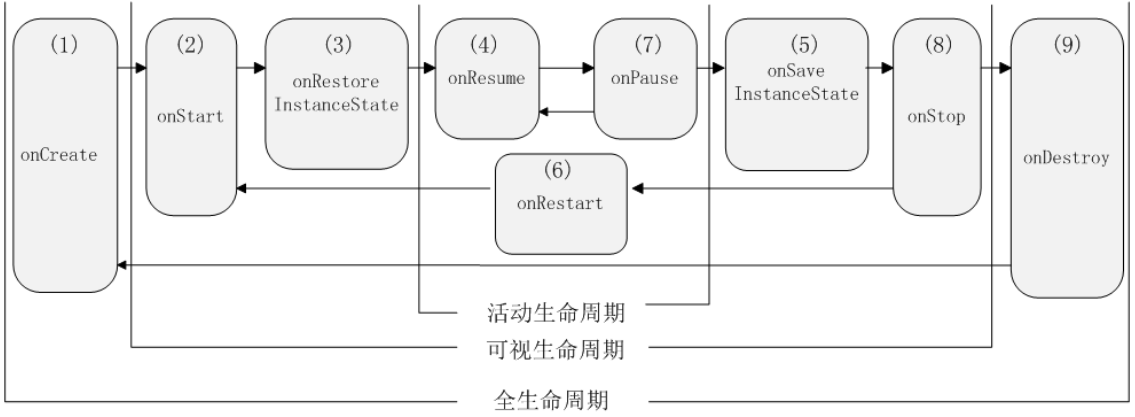


- Activity的生命周期可分为全生命周期、可视生命周期和活动生命周期
- 每种生命周期中包含不同的事件回调函数

2.2.2 Activity的三个生命周期

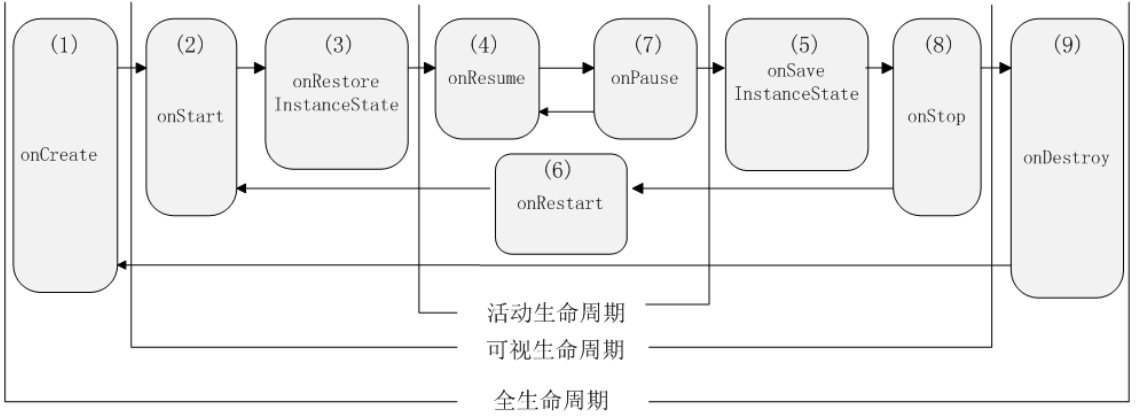
- 全生命周期
  - 从onCreate(Bundle)开始到onDestroy()结束。Activity在onCreate()设置所有的“全局”状态，在onDestory()释放所有的资源。例如：某个Activity有一个在后台运行的线程，用于从网络下载数据，则该Activity可以在onCreate()中创建线程，在onDestory()中停止线程。
- 可视生命周期
  - 从onStart()开始到onStop()结束。在这段时间，可以看到Activity在屏幕上，尽管有可能不在前台，不能和用户交互。在这两个接口之间，需要保持显示给用户的UI数据和资源等，例如：可以在onStart中注册一个IntentReceiver来监听数据变化导致UI的变动，当不再需要显示时候，可以在onStop()中注销它。onStart()，onStop()都可以被多次调用，因为Activity随时可以在可见和隐藏之间转换。
- 活动生命周期
  - 从onResume()开始到onPause()结束。在这段时间里，该Activity处于所有 Activity的最前面，和用户进行交互。Activity可以经常性地在resumed和paused状态之间切换，例如：当设备准备休眠时，当一个 Activity处理结果被分发时，当一个新的Intent被分发时。所以在这些接口方法中的代码应该属于非常轻量级的。

2.2.2 Activity的三个生命周期——全生命周期



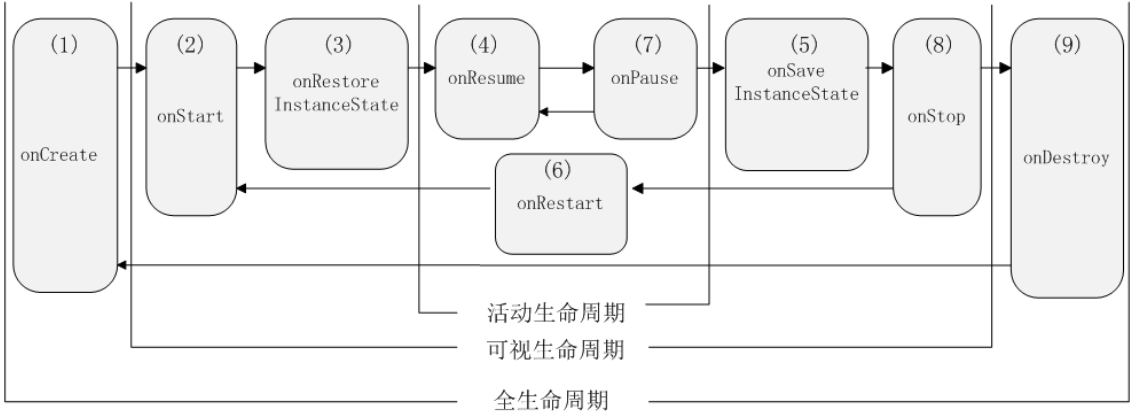
- 全生命周期是从Activity建立到销毁的全部过程，始于onCreate()，结束于onDestroy()
  - 使用者通常在onCreate()中初始化Activity所能使用的全局资源和状态，并在onDestroy()中释放这些资源
  - 在一些极端的情况下，Android系统会不调用onDestroy()函数，而直接终止进程

2.2.2 Activity的三个生命周期——可视生命周期



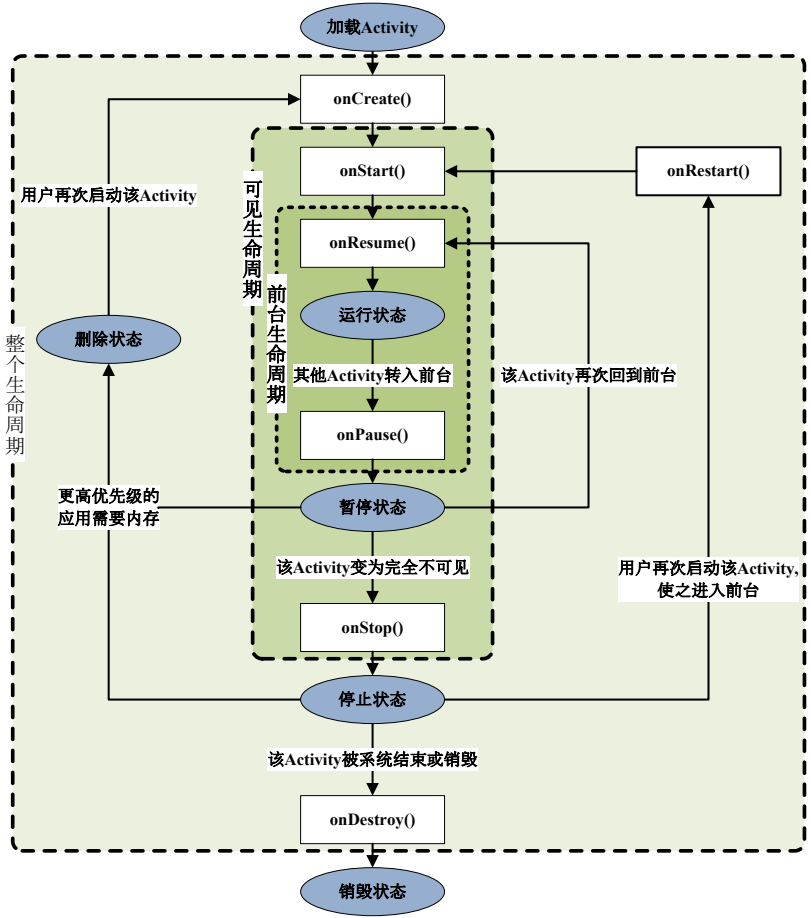
- 可视生命周期是Activity在界面上从可见到不可见的过程，开始于onStart()，结束于onStop()

2.2.2 Activity的三个生命周期——活动生命周期



- 活动生命周期是Activity在屏幕的最上层，并能够与用户交互的阶段，开始于onResume()，结束于onPause()
- 在Activity的状态变换过程中onResume()和onPause()经常被调用，因此这两个函数中应使用更为简单、高效的代码

2.2.3 Activity的生命周期



正常情况下Activity的生命周期是: onCreate->onStart->onResume->onPause->onStop->onDestroy
正常的Activity，第一次启动，会依次回调以下方法：
onCreate->onStart->onResume
打开一个新的Activity或者点击Home键回到桌面后，会依次回调以下方法：
onPause->onStop。

上面提到过,如果新的Activity是透明的(采用的透明主题)，当前的Activity不会回调onStop.
当我们再次回到原Activity，会依次回调以下方法：
onRestart->onStart->onResume.
当我们点击返回键后，会依次回调以下方法： onPause->onStop->onDestroy.
当Activity被系统回收后，再次被打开，会跟第一次启动的时回调生命周期方法一样（不包含 onSaveInstanceState 和 onRestoreInstanceState）。
onCreate 跟 onDestroy ，onStart跟onStop ，onResume 和 onPause 是配对的，它们一个获取焦点和用户交互，一个正好相反。
onStart和onStop 是Activity是否可见的标志，而onResume和onPause是从Activity是否位于前台的标志，它们针对的角度不同。
在onPause里不能做耗时操作，因为如果要启动一个新的Activity，新的Activity必须要在前一个Activity的onPause方法执行完毕之后才会启动的新的Activity。

2.2.4 Activity的生命周期

Activity类的定义

```
public class Activity extends ContextThemeWrapper {
    protected void onCreate(Bundle icicle){...}
    protected void onStart(){...}
    protected void onRestart(){...}
    protected void onResume(){...}
    protected void onPause(){...}
    protected void onStop(){...}
    protected void onDestroy(){...}
}
```

2.2.4 Activity的回调方法

- onCreate
  - 当Activity第一次被创建时调用。是生命周期开始的第一个方法。在这里我们可以做一些初始化的操作，比如:调用setContentView()方法去加载界面，绑定布局里的一些控件，初始化一些Activity需要用到的数据。之后会调用onStart方法。
- onStart
  - 当Activity正在变为可见时调用。这个时候Activity已经可见了，但是还没有出现在前台还不能跟用户交互。可以简单理解为Actvity已经可见但是还没有出现在前台。之后会调用onResume。
- onResume
  - 当Activity可以跟用户交互时调用，这个时候，这个Activity位于栈的顶部。跟onStart相比,它们都是表示Activity已经可见，但是onStart调用时Activity还在后台，而调用onResume时，Activity已经进入了前台，可以跟用户交互了。之后会调用 onPause。

2.2.4 Activity的回调方法

- onPause
  - 当Activity暂停时调用这个方法；在这里我们可以用来保存数据，关闭动画和其它比较耗费CPU的操作；但是在这里做的操作绝对不能耗时，因为如果当前Activity要启动一个新的Activity，这个新的Activity会在当前Activity执行完毕onPause之后才能进入可见状态。这个方法之后一般会调用的方法有onStop或者onResume。
- onStop
  - 当Activity进入后台，并且不会被用户看到时调用。当别的Activity出现在前台时，或者Activity会被销毁时，调用此方法；在这个方法调用之后，系统可能会在内存不够的情况下回收Activity；在这个方法之后一般会调用onRestart或者onDestroy。

2.2.4 Activity的回调方法

- onDestroy
  - 这个方法是Activity生命周期中调用的最后一个方法。它会在Activity被销毁之前调用；Activity销毁原因一般是我们调用Activity的finish方法手动销毁，另一个就是系统在内存紧张的情况下去销毁Activity，以用来节省空间。我们可以通过方法 isFinishing 来判断Activity是否正在被销毁。
- onRestart
  - 这个方法是在Activity处于停止状态后，又回到可视状态时调用。之后会调用onResume。

2.2.4 Activity的问题思考

1、如果所有的初始化都在onCreate()中实现，会有什么问题？

- 首先，Activity的onCreate()被调用时，Activity还不可见，如果要做一些动画，既然视图还不存在，在onCreate中来启动动画，明显有问题；
- 其次，AActivity 切换到 BActivity，再切换到 AActivity，由于实例已经存在，所以onCreate不会再被调用，那问题就在于AActivity从后台切换至前台时，有可能需要一些初始化，就没法被调用到了。

2、如果所有的初始化都在onStart()中实现，会有什么问题？

- 首先，虽然 在onStart()中用 setContentView()、findViewById() 功能也是正常的，但是onCreate()注释中，明确建议 setContentView()、findViewById() 要在 onCreate() 中被调用。
- 其次， onResume()的注释中都明确地说了这不是 Activity 对用户是可见的最好的指示器，如果在 onStart() 中做全部初始化，很有可能初始化还没完成影响到用户的交互体验。

2.2.4 Activity的问题思考

- 3、如果所有资源回收都在onStop()中实现，会有什么问题？

  - 首先，在 onResume() 的注释中，建议是在onResume()中打开独占设备（比如相机），与onResume()对应的是onPause()，关闭相机的操作也应该在此方法中被调用；否则，考虑一下如下场景：如果AActivity打开了相机，我们点击某按钮要跳转到BActivity中，BActivity也想打开相机；假设AActivity的onPause() 在 BActivity启动后再被调用，那BActivity根本就无法再正常启动相机。
  - 在onPause() 的注释中明确表示，应该在这个方法中执行停止动画等比较耗CPU的操作，如果不先执行这些操作，就先启动新应用，然后再来执行此操作，确实是不合逻辑；其次，onStop() 的注释中也明确地写了，在内存不足而导致系统自动回收进程情况下，onStop() 可能都不会被执行。

2.2.4 Activity的问题思考

- 4、Activity间跳转时，为什么AActivity的onPause()被调用后，BActivity的初始化流程（onCreate() -> onStart() -> onResume()），然后AActivity的onStop()被调用？

  - 当用户点击打开新的Activity，肯定是想尽快进入新的视图进行操作。而且上面的问题已经解释了，在onResume()一般会打开独占设备，开启动画等，当需要从AActivity切换到BActivity时，先执行AActivity中的onPause()进行关闭独占设备，关闭动画等，以防止BActivity也需要使用这些资源，因为AActivity的资源回收，也有利于BActivity运行的流畅。
  - 当AActivity中比较消耗资源的部分在onPause()中关闭后，再执行BActivity的初始化，显示视图与用户交互。然后，系统在后台默默执行AActivity的onStop()操作，去回收AActivity占用的其余资源。即使onStop()中会有些比较耗时的操作，也没有关系，这是在后台执行也不会影响到用户的体验。

2.2.5 Activity异常情况下的生命周期

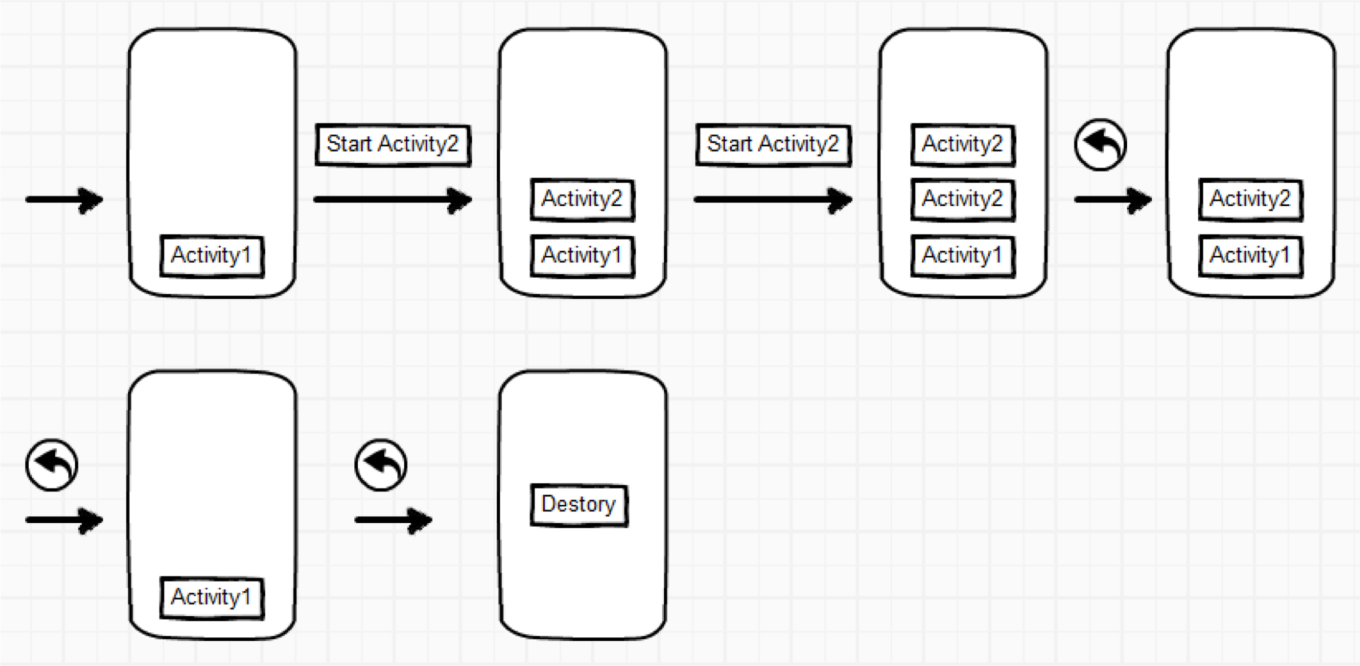
在系统内存不够时会根据优先级杀死Activity。怎么判断Activity的优先级呢？

- 最高的优先级：
  在前台显示并且跟用户交互的Activity，优先级最高；
- 暂停状态的Activity优先级次之：
  如果Activity没有在前台，但是可见，不可与用户交互，比如弹出一个对话框等；
- 处于后台Activity优先级最低：
  执行了onStop方法的Activity优先级最低。它不可见，并且无法跟用户交互。

当系统内存不足，就会按照优先级去销毁Activity，
在销毁Activity时会额外的在onPause和onStop之间调用onSaveInstanceState；
当要重新创建这个Activity 时，会在onStart方法之后调用onRestoreInstanceState方法。

2.2.6 Activity的四种启动模式——standard模式

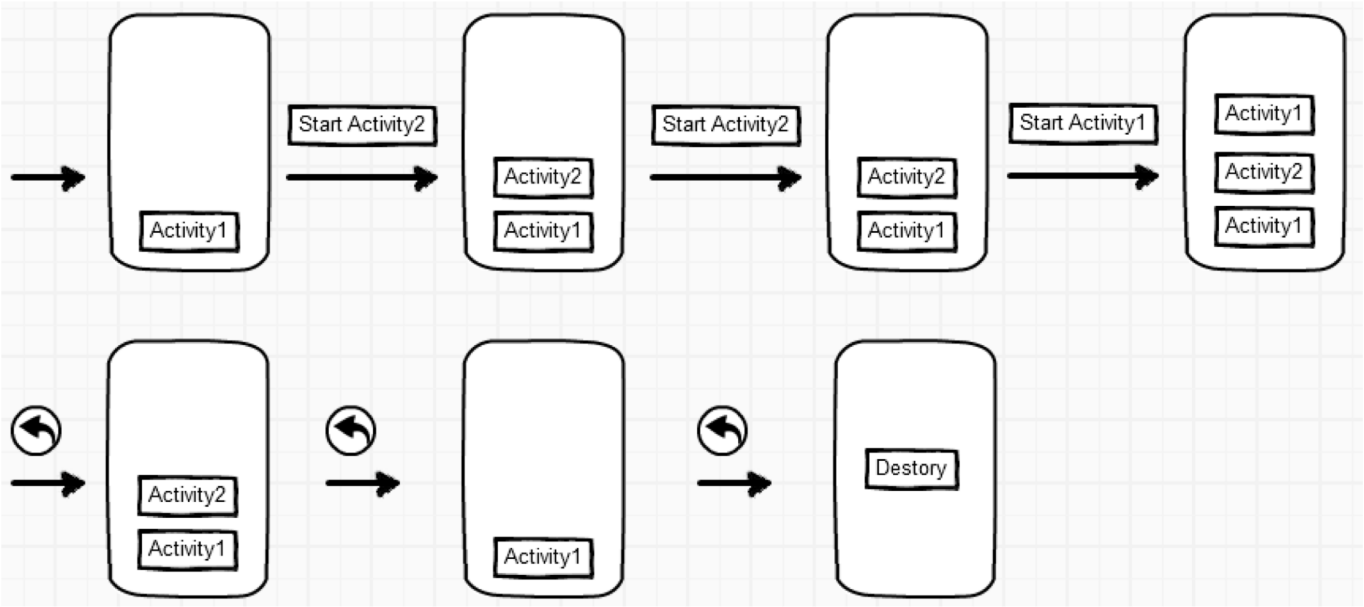
standard模式是Activity的默认启动方式，每启动一个Activity就会在栈顶创建一个新的实例。



演示

2.2.6 Activity的四种启动模式——singleTop模式

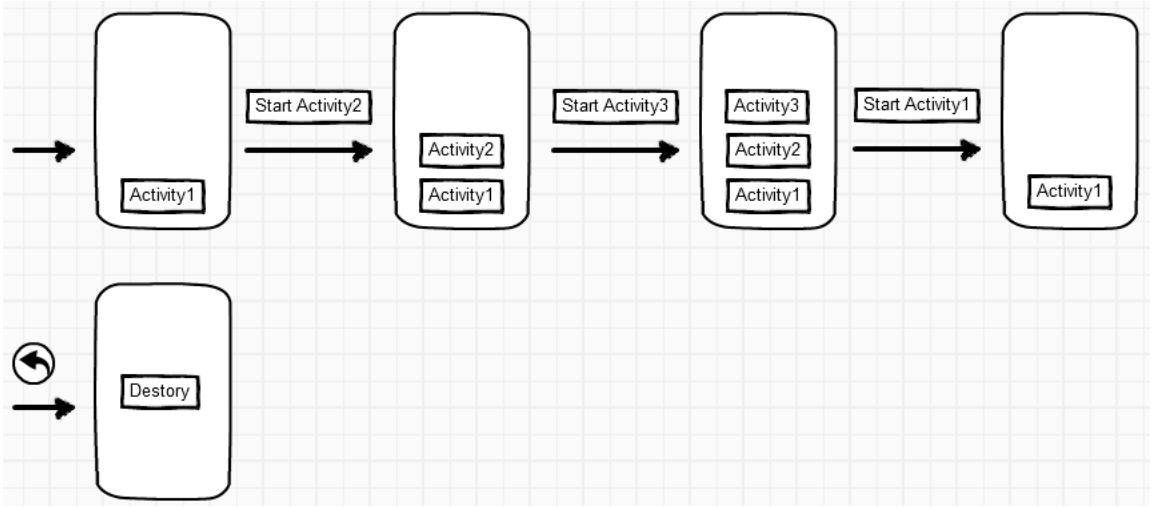
singleTop模式会判断要启动的Activity实例是否位于栈顶，如果位于栈顶则直接复用，否则创建新的实例。



演示

2.2.6 Activity的四种启动模式——singleTask模式

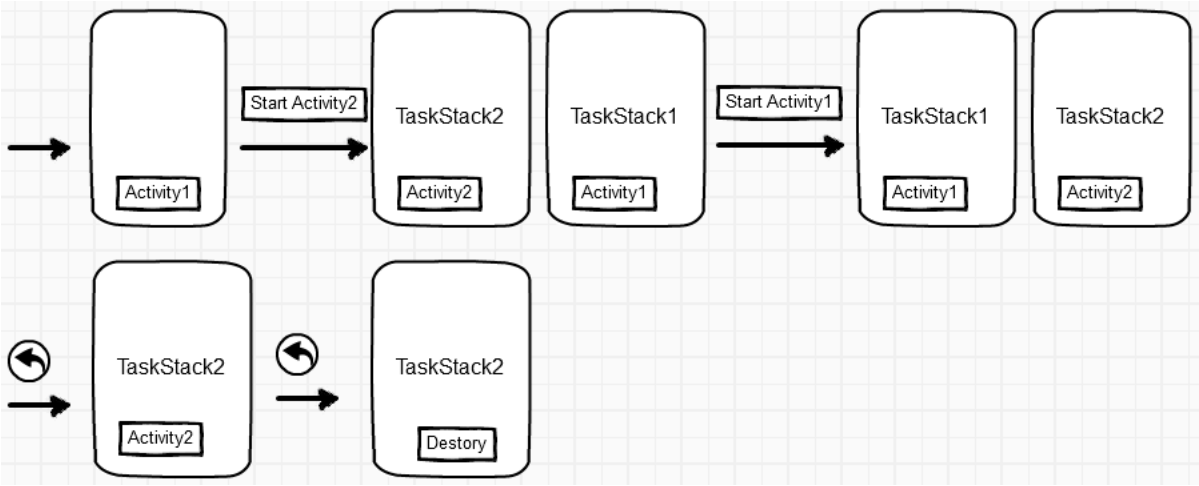
singleTask模式下每次启动该Activity时，系统首先会检查栈中是否存在当前Activity实例，如果存在则直接使用，并把当前Activity之上的所有实例全部出栈。



演示

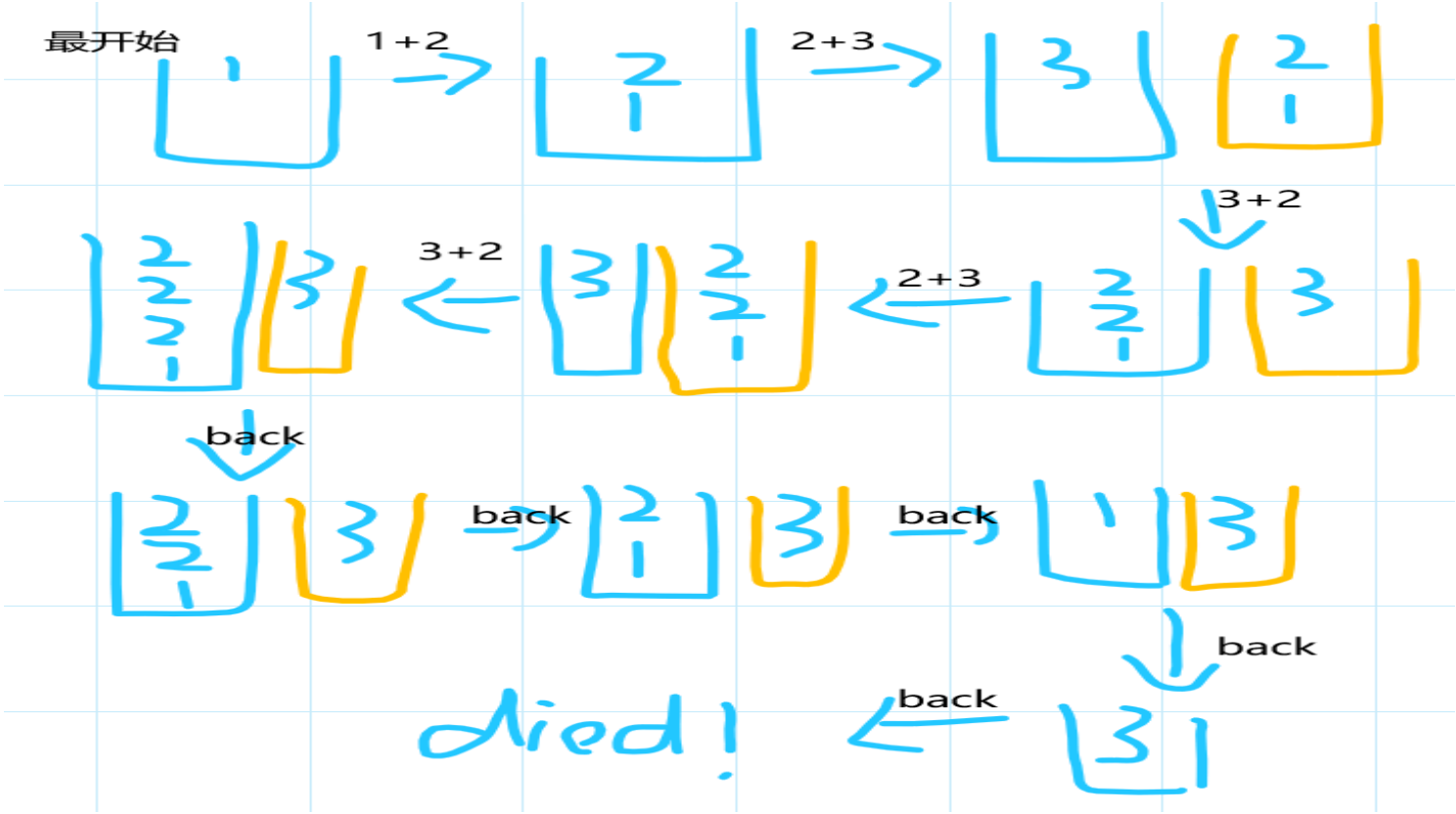
2.2.6 Activity的四种启动模式——singleInstance模式

singleInstance模式会启动一个新的任务栈来管理Activity实例，无论从哪个任务栈中启动该Activity，该实例在整个系统中只有一个。



演示

2.2.6 Activity的四种启动模式——singleInstance模式



演示

2.2.7 Log日志信息

- Log日志类能够记录程序运行过程中的相关信息

```
import android.util.Log;
```

方 法	功能描述
Log.e()	记录错误信息
Log.w()	记录警告信息
Log.i()	记录一般提示性信息
Log.d()	记录调试信息
Log.v()	记录详细的信息

2.3 资源分类

Android的资源可分为两大类：

- 原生资源：无法通过由R类进行索引的原生资源：文本文件、图像文件、网页文件（包括html中引用的js/ccs/jpg等资源）、音频视频文件
- 索引资源：通过R类进行自动索引的资源

2.3 资源分类

- **Android应用资源的类型及存放目录：**

目 录	资源描述
/res/animator/	存放定义属性动画的XML文件
/res/anim/	存放定义了补间动画或逐帧动画的XML文件
/res/color/	存放定义不同状态下颜色列表的XML文件
/res/drawable/	存放能转换为绘制资源的位图文件或者定义了绘制资源的XML文件
/res/layout/	存放各种界面布局文件，每个Activity对应一个XML文件
/res/menu/	存放为应用程序定义的各种菜单资源
/res/raw/	存放直接复制到设备中的任意文件
/res/values/	存放定义多种类型资源的XML文件
/res/xml/	存放任意的原生XML文件
/assets/	存放原生资源，包括音频文件、视频文件等

2.3 资源分类——资源访问方式

资源访问的方式有两种：

- Java代码访问资源
- 在XML文件中访问资源

2.3 资源分类——Java代码访问res资源

- Java代码访问res资源

```
[packageName.]R.resourceType.resourceName
```

- Resources类中提供的访问资源的方法：

方 法	功能描述
int getColor(int id)	对应res/values/colors.xml
Drawable getDrawable(int id)	对应res/drawable/
XmlResourceParser getLayout(int id)	对应res/layout/
String getString(int id)	对应res/values/strings.xml
CharSequence getText(int id)	对应res/values/strings.xml
InputStream openRawResource(int id)	对应res/raw/
void parseBundleExtra (String tagName,AttributeSet attrs, Bundle outBundle)	对应res/xml/
String[] getStringArray(int id)	对应res/values/arrays.xml
float getDimension(int id)	对应res/values/dimens.xml

2.3 资源分类——Java代码访问res资源

- Java代码访问res资源

```
public class Color_ActivityDemo extends
AppCompatActivity {
    TextView tx4;
    @Override
    public void onCreate(Bundle
savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.color_layout);
        tx4= (TextView)
findViewById(R.id.tv4);

tx4.setTextColor(getResources().getColor(
R.color.color_java));
    }
}
```

- xml代码访问res资源

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

android:textAppearance="?android:attr/tex
tAppearanceLarge"
    android:text="XML文件访问colors资源（红
色）"
    android:id="@+id/tv3"

android:textColor="@color/color_xml"/>
```

2.3 资源分类——Java代码访问assets原生资源

- Java代码中通过AssetManager类访问assets原生资源
- 通过Resources类的getAssets()方法获取AssetManager对象

```
getResources().getAssets().open("资源名");
```

- **在XML文件中使用资源**

```
@[packageName:]resourceType/resourceName
```

```
InputStream is = getAssets().open(fileName);
bitmap = BitmapFactory.decodeStream(is);
ivImg.setImageBitmap(bitmap);
```

2.3 XML资源文件

- strings.xml：用于定义文本内容的资源文件
- colors.xml：用于定义颜色设置的资源文件
- dimens.xml：用于定义尺寸的资源文件
- styles.xml：用于定义主题和样式的资源文件

2.3 文本资源文件

- strings.xml文本资源文件

```
<!—文本资源文件res\values\strings.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Resources</string>
    <string name="message">Hello World!</string>
    <string name="error">Wrong resource!</string>
</resources>
```

Java代码中访问字符串

【语法】

```
R.string.字符串名
```

【示例】

```
CharSequence appName = getString(R.string.title);
CharSequence display = getString(R.string.message);
```

2.3 文本资源文件

- **XML文件中访问字符串资源**
  - **【语法】**

```
@string/字符串名
```

  - **【示例】**

```
android:app_name="@string/title"
android:display="@string/message"
```

2.3 colors.xml颜色设置资源文件

- 颜色值的声明有以下四种方式
  - #RGB
  - #ARGB
  - #RRGGBB
  - #AARRGGBB

- 【语法】

```
<color name=color_name>#color_value</color>
```

- 【示例】使用<color>标记定义颜色

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="text_color">#F00</color>
    <color name="translucent_blue">#800000ff</color>
</resources>
```

2.3 colors.xml颜色设置资源文件

- Java代码中访问颜色
  - 【语法】

```
R.color.颜色名
```

  - 【示例】

```
int color1= getResources().getColor(R.color.blue);
int color2= getResources().getColor(R.color.translucent_blue);
```

在getColor(int id)在API 23以上版本中已过时，使用ContextCompat.getColor(Context context,int id)方法进行替代，该方法能够同时兼容高低版本。

2.3 colors.xml颜色设置资源文件

- **XML文件中访问字符串资源**
  - **【语法】**

```
@color/颜色名
```

  - **【示例】**

```
android:titleColor="@color/blue"
android:textColor="@color/translucent_blue"
```

2.3 dimens.xml尺寸定义资源文件

- **Android可以采用以下单位来指定尺寸**

测量单位	描述	资源标记	示例
像素	实际的屏幕像素	px	10px
英寸	物理测量单位	in	6in
毫米	物理测量单位	mm	4mm
点	普通字体测量单位	pt	12pt
密度独立像素	相对于160dpi屏幕的像素	dp	3dp
比例独立像素	对于字体显示的测量	sp	10sp

- **dimens.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resource>
    <dimen name="one_pixel">1px</dimen>
    <dimen name="two_inches">2in</dimen>
    <dimen name="double_density">2dp</dimen>
    <dimen name="fourteen_sp">14sp</dimen>
</resource>
```

2.3 dimens.xml尺寸定义资源文件

- Java代码中尺寸资源
  - 【语法】

```
R.dimen.尺寸名
```

  - 【示例】

```
float dimen= getResources.getDimension(R.dimen.one_pixel);
float dimen= getResources.getDimension(R.dimen.fourteen_sp);
```

2.3 dimens.xml尺寸定义资源文件

- XML文件中访问访问尺寸资源
  - 【语法】

```
@dimen/尺寸名
```

  - 【示例】

```
android:textSize="@dimen/fourteen_sp"
android:textSize="@dimen/double_density "
```

2.3 styles.xml主题风格资源文件

- style.xml

```
<style name=style_name>
    <item name=item_name>Hex value|string value|reference</item>
</style>
```

- **Java代码中访问样式资源**
  - **【语法】**

```
R. style.样式名
```

  - **【示例】**

```
setTheme(R.style.ThemeNew);
setTheme(R.style.myStyle);
```

2.3 styles.xml主题风格资源文件

- XML文件中访问样式资源
  - 【语法】

```
@style/样式名
```

  - 【示例】

```
android:app_name="@style/ThemeNew"
android:display="@style/myStyle"
```

2.3 drawable图像资源目录

- Android应用程序中所使用的小图标、图像或背景图像存放在资源目录res/drawable下
- Java代码中访问图像资源
  - 【语法】

```
R.drawable.图像文件名
```

  - 【示例】

```
Resource.getDrawable(R.drawable.icon);
setBackgroundDrawable(getResources().getDrawable(R.drawable.background));
```

2.3 drawable图像资源目录

- XML文件中访问图像资源
  - 【语法】

```
@drawable/图像文件名
```

  - 【示例】

```
android:icon="@drawable/app_icon"
android:background="@drawable/background"
```

谢谢