Activity是什么?

我们都知道android中有四大组件（Activity 活动，Service 服务，Content Provider 内容提供者，BroadcastReceiver 广播接收器），Activity是我们用的最多也是最基本的组件，因为应用的所有操作都与用户相关，Activity 提供窗口来和用户进行交互。

官方文档这么说：

> An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View).
>
> 大概的意思：
>
> activity是独立平等的，用来处理用户操作。几乎所有的activity都是用来和用户交互的，所以activity类会创建了一个窗口，开发者可以通过setContentView(View)的接口把UI放到给窗口上。

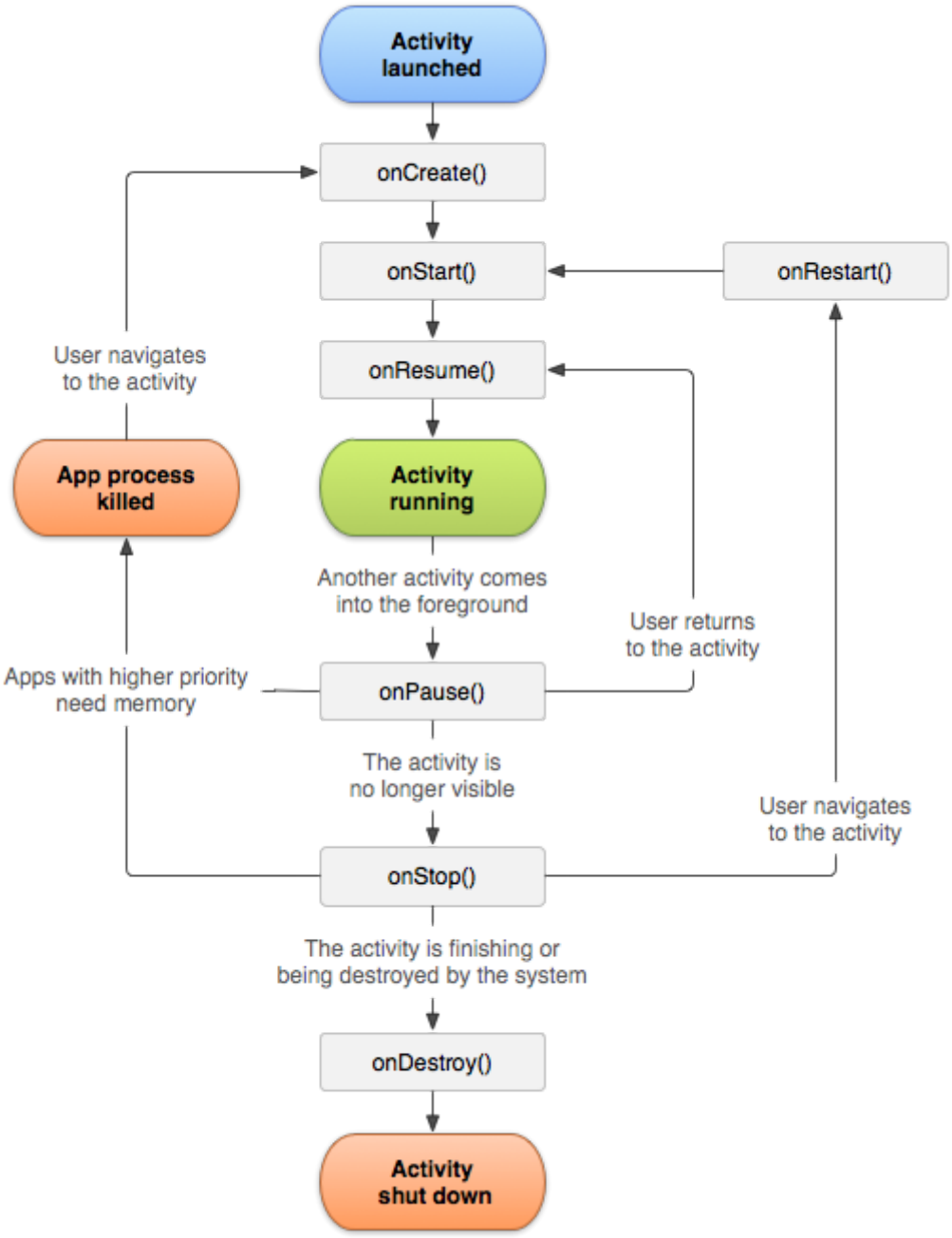
Android中的activity全都归属于task管理 。task 是多个 activity 的集合，这些 activity 按照启动顺序排队存入一个栈（即“back stack”）。android默认会为每个App维持一个task来存放该app的所有activity，task的默认name为该app的packagename。

当然我们也可以在AndroidMainfest.xml中申明activity的taskAffinity属性来自定义task，但不建议使用，如果其他app也申明相同的task，它就有可能启动到你的activity，带来各种安全问题（比如拿到你的Intent）。

Activity的内部调用过程

上面已经说了，系统通过堆栈来管理activity，当一个新的activity开始时，它被放置在堆栈的顶部和成为运行活动，以前的activity始终保持低于它在堆栈，而不会再次到达前台，直到新的活动退出。

还是上这张官网的activity_lifecycle图：



- 首先打开一个新的activity实例的时候，系统会依次调用

> onCreate（） -> onStart() -> onResume() 然后开始running

running的时候被覆盖了（从它打开了新的activity或是被锁屏，但是它**依然在前台**运行， lost focus but is still visible），系统调用onPause();

> 该方法执行activity暂停，通常用于提交未保存的更改到持久化数据，停止动画和其他的东西。但这个activity还是完全活着（它保持所有的状态和成员信息，并保持连接到**窗口管理器**）

接下来它有三条出路
①用户返回到该activity就调用onResume()方法重新running

②用户回到桌面或是打开其他activity，就会调用onStop()进入停止状态（保留所有的状态和成员信息，**对用户不可见**）

③系统内存不足，拥有更高限权的应用需要内存，那么该activity的进程就可能会被系统回收。（回收onRause()和onStop()状态的activity进程）要想重新打开就必须重新创建一遍。

如果用户返回到onStop()状态的activity（又显示在前台了），系统会调用

> onRestart() -> onStart() -> onResume() 然后重新running

在activity结束（调用finish ()）或是被系统杀死之前会调用onDestroy()方法释放所有占用的资源。

> activity生命周期中三个嵌套的循环

- activity的完整生存期会在 onCreate() 调用和 onDestroy() 调用之间发生。
- activity的可见生存期会在 onStart() 调用和 onStop() 调用之间发生。系统会在activity的整个生存期内多次调用 onStart() 和onStop()， 因为activity可能会在显示和隐藏之间不断地来回切换。
- activity的前后台切换会在 onResume() 调用和 onPause() 之间发生。因为这个状态可能会经常发生转换，为了避免切换迟缓引起的用户等待，这两个方法中的代码应该相当地轻量化。

activity被回收的状态和信息保存和恢复过程

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        if (savedInstanceState != null) { //判断是否有以前的保存状态信息
            savedInstanceState.get("Key");
        }

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        // TODO Auto-generated method stub
        //可能被回收内存前保存状态和信息，
        Bundle data = new Bundle();
        data.putString("key", "last words before be kill");
        outState.putAll(data);
        super.onSaveInstanceState(outState);
    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        if (savedInstanceState != null) { //判断是否有以前的保存状态信息
            savedInstanceState.get("Key");
        }

        super.onRestoreInstanceState(savedInstanceState);
    }
}
```

> onSaveInstanceState方法

在activity　可能被回收之前　调用,用来保存自己的状态和信息，以便回收后重建时恢复数据（在onCreate()或onRestoreInstanceState()中恢复）。旋转屏幕重建activity会调用该方法，但其他情况在onRause()和onStop()状态的activity不一定会调用 ， 下面是该方法的文档说明。

> One example of when onPause and onStop is called and not this method is when a user navigates back from activity B to activity A: there is no need to call onSaveInstanceState on B because that particular instance will never be restored, so the system avoids calling it. An example when onPause is called and not onSaveInstanceState is when activity B is launched

> in front of activity A: the system may avoid calling onSaveInstanceState on activity A if it isn't killed during the lifetime of B since the state of the user interface of A will stay intact.

也就是说，系统灵活的来决定调不调用该方法，**但是如果要调用就一定发生在onStop方法之前，但并不保证发生在onPause的前面还是后面。**

> onRestoreInstanceState方法

这个方法在onStart 和 onPostCreate之间调用，在onCreate中也可以状态恢复，但有时候需要所有布局初始化完成后再恢复状态。

onPostCreate：一般不实现这个方法，当程序的代码开始运行时，它调用系统做最后的初始化工作。

启动模式

启动模式什么?

简单的说就是定义activity 实例与task 的关联方式。

为什么要定义启动模式?

为了实现一些默认启动（standard）模式之外的需求：

- 让某个 activity 启动一个新的 task （而不是被放入当前 task ）
- 让 activity 启动时只是调出已有的某个实例（而不是在 back stack 顶创建一个新的实例）
- 或者，你想在用户离开 task 时只保留根 activity，而 back stack 中的其它 activity 都要清空

怎样定义启动模式?

定义启动模式的方法有两种：

使用 manifest 文件

在 manifest 文件中activity声明时，利用 activity 元素的 launchMode 属性来设定 activity 与 task 的关系。

```
<activity
          . . . . . .
          android:launchMode="standard"
           >
         . . . . . . .
      </activity>
```

> 注意： 你用 launchMode 属性为 activity 设置的模式可以被启动 activity 的 intent 标志所覆盖。

有哪些启动模式?

- "standard" （默认模式）
  当通过这种模式来启动Activity时， Android总会为目标 Activity创建一个新的实例,并将该Activity添加到当前Task栈中。这种方式不会启动新的Task,只是将新的 Activity添加到原有的Task中。
- "singleTop"
  该模式和standard模式基本一致,但有一点不同:当将要被启动的Activity已经位于Task栈顶时,系统不会重新创建目标Activity实例,而是直接复用Task栈顶的Activity。

- "singleTask"
  Activity在同一个Task内只有一个实例。
  如果将要启动的Activity不存在,那么系统将会创建该实例,并将其加入Task栈顶;
  如果将要启动的Activity已存在,且存在栈顶,直接复用Task栈顶的Activity。
  如果Activity存在但是没有位于栈顶,那么此时系统会把位于该Activity上面的所有其他Activity全部移出Task,从而使得该目标Activity位于栈顶。
- "singleInstance"
  无论从哪个Task中启动目标Activity,只会创建一个目标Activity实例且会用一个全新的Task栈来装载该Activity实例（全局单例）.
  如果将要启动的Activity不存在,那么系统将会先创建一个全新的Task,再创建目标Activity实例并将该Activity实例放入此全新的Task中。
  如果将要启动的Activity已存在,那么无论它位于哪个应用程序,哪个Task中;系统都会把该Activity所在的Task转到前台,从而使该Activity显示出来。

使用 Intent 标志

在要启动 activity 时，你可以在传给 startActivity() 的 intent 中包含相应标志，以修改 activity 与 task 的默认关系。

```
        Intent i = new Intent(this,NewActivity.class);
        i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        startActivity(i);
```

可以通过标志修改的默认模式有哪些?

- FLAG_ACTIVITY_NEW_TASK
  与"singleTask"模式相同，在新的 task 中启动 activity。如果要启动的 activity 已经运行于某 task 中，则那个 task 将调入前台。
- FLAG_ACTIVITY_SINGLE_TOP
  与 "singleTop"模式相同，如果要启动的 activity位于back stack 顶，系统不会重新创建目标Activity实例,而是直接复用Task栈顶的Activity。
- FLAG_ACTIVITY_CLEAR_TOP
  此种模式在launchMode中没有对应的属性值。如果要启动的 activity 已经在当前 task 中运行，则不再启动一个新的实例，且所有在其上面的 activity 将被销毁。

关于启动模式的一些建议

一般不要改变 activity 和 task 默认的工作方式。 如果你确定有必要修改默认方式，请保持谨慎，并确保 activity 在启动和从其它 activity 返回时的可用性，多做测试和安全方面的工作。

Intent Filter

android的3个核心组件——Activity、services、广播接收器——是通过intent传递消息的。intent消息用于在运行时绑定不同的组件。

在 Android 的 AndroidManifest.xml 配置文件中可以通过 intent-filter 节点为一个 Activity 指定其 Intent Filter，以便告诉系统该 Activity 可以响应什么类型的 Intent。

intent-filter 的三大属性

Action

一个 Intent Filter 可以包含多个 Action，Action 列表用于标示 Activity 所能接受的“动作”，它是一个用户自定义的字符串。

```
<intent-filter >
 <action android:name="android.intent.action.MAIN" />
 <action android:name="com.scu.amazing7Action" />
……
 </intent-filter>
```

在代码中使用以下语句便可以启动该Intent 对象：

```
Intent i=new Intent();
i.setAction("com.scu.amazing7Action");
```

Action 列表中包含了“com.scu.amazing7Action”的 Activity 都将会匹配成功

URL

在 intent-filter 节点中，通过 data节点匹配外部数据，也就是通过 URI 携带外部数据给目标组件。

```
<data android:mimeType="mimeType"
    android:scheme="scheme"
     android:host="host"
     android:port="port"
     android:path="path"/>
```

注意：只有data的所有的属性都匹配成功时 URI 数据匹配才会成功

Category

为组件定义一个 类别列表，当 Intent 中包含这个类别列表的所有项目时才会匹配成功。

```
<intent-filter . . . >
   <action android:name="code android.intent.action.MAIN" />
   <category android:name="code  android.intent.category.LAUNCHER" />
</intent-filter>
```

Activity 种 Intent Filter 的匹配过程

①加载所有的Intent Filter列表

②去掉action匹配失败的Intent Filter

③去掉url匹配失败的Intent Filter

④去掉Category匹配失败的Intent Filter

⑤判断剩下的Intent Filter数目是否为0。如果为0查找失败返回异常；如果大于0，就按优先级排序，返回最高优先级的Intent Filter

开发中Activity的一些问题

- 一般设置Activity为非公开的

```
<activity
. . . . . .
android:exported="false" />
```

注意：非公开的Activity不能设置intent-filter，以免被其他activity唤醒（如果拥有相同的intent-filter）。

- 不要指定activity的taskAffinity属性
- 不要设置activity的LaunchMode（保持默认）
- 注意Activity的intent最好也不要设定为FLAG_ACTIVITY_NEW_TASK
- 在匿名内部类中使用this时加上activity类名（类名.this,不一定是当前activity）
- 设置activity全屏

在其 onCreate()方法中加入：

```
// 设置全屏模式
 getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
 // 去除标题栏
 requestWindowFeature(Window.FEATURE_NO_TITLE);
```