# Github博客原始文档

## 亮点

### 亮点：windowSoftInputMode

|  |
| --- |
| <activity android:name=*".app.CustomTitle"*  android:label=*"@string/activity\_custom\_title"*  android:windowSoftInputMode=*"stateVisible|adjustPan"*> |

### 2．亮点：listView 过滤文字

AbsListView.setTextFilterEnabled(true);

这个方法的作用是用来过滤选项的.

例如在软键盘上打出一个a,则会过滤掉除了a开头的所有选项.

## 功能点

### 功能：自定义 titleBar

|  |
| --- |
| requestWindowFeature(Window.***FEATURE\_CUSTOM\_TITLE***);  setContentView(R.layout.***custom\_title***);  getWindow().setFeatureInt(Window.***FEATURE\_CUSTOM\_TITLE***,  R.layout.***custom\_title\_1***);  **final** TextView leftText = (TextView) findViewById(R.id.***left\_text***); |

1. 功能：监听apk安装卸载，获取安装卸载apk包名和uid

|  |
| --- |
| 注册广播 <receiverandroid:name=*"com.cs.cspermission.InstallApkReceiver"* >  <intent-filter>  <action android:name=*"android.intent.action.PACKAGE\_ADDED"* />  <action android:name=*"android.intent.action.PACKAGE\_REMOVED"* />  <data android:scheme=*"package"* />  </intent-filter>  </receiver> |

|  |
| --- |
| 获取apk uid 和 packageName  **if**(Intent.***ACTION\_PACKAGE\_ADDED***.equalsIgnoreCase(intent.getAction())){  Uri uri = **null**;  uri = intent.getData();  **int** uid = intent.getIntExtra(Intent.***EXTRA\_UID***, -4);  String packageName = uri.getEncodedSchemeSpecificPart();  ……  …… |

### 功能：在values string中定义 数组，代码中使用

|  |
| --- |
| <string-array name=*"dialog\_title\_time"*>  <item>今日</item>  <item>本月</item>  </string-array>  **New** ArrayList<String>(Arrays.*asList*(  getResources().getStringArray(R.array.***dialog\_title\_time***))) |

### 功能：一段文字中包含电话等url高亮显示并可点击触发事件(打电话，调起浏览器)

|  |
| --- |
| **if** (text **instanceof** Spannable) {  **int** end = text.length();  Spannable sp = (Spannable) mTextInfo.getText();  URLSpan[] urls = sp.getSpans(0, end, URLSpan.**class**);  SpannableStringBuilder style = **new**SpannableStringBuilder(text);  style.clearSpans();// should clear old spans  **for** (URLSpan url : urls) {  MyURLSpan myURLSpan = **new** MyURLSpan(url.getURL());  style.setSpan(myURLSpan, sp.getSpanStart(url), sp.getSpanEnd(url), Spannable.***SPAN\_EXCLUSIVE\_INCLUSIVE***);  }  mTextInfo.setText(style);  } |

### 5.功能：文件 IO

|  |
| --- |
| FileOutputStream outStream = this.openFileOutput("itcast.txt", Context.MODE\_PRIVATE);          outStream.write("要写入的数据".getBytes());          outStream.close(); |

第一个参数：指定文件名称，不能包含路径分隔符“/” ，如果文件不存在，Android 自动创建它。保存在/data/data/<package name>/files目录中，如： /data/data/cn.itcast.action/files/ceshi.txt

获取当前apk 系统目录

getFilesDir();

getCacheDir();

getDir(“adc”, MODE\_PRIVATE); /data/data/<package name>/app\_abc

getDatabasePath(“stu.db”); /data/data/<package name>/stu.db

Environment.get

### 6.功能：bitmap 以及 图片 byte[] 保存到本地图片

|  |
| --- |
| **public** **static** **void** writeBitmapToDisk(Bitmap bitmap) {  String imgPath = Environment.*getExternalStorageDirectory*().getPath()+ "/image" + bitmap.hashCode() + ".png";  File imgFile = **new** File(imgPath);  **if** (!imgFile.exists()) {  **try** {  imgFile.createNewFile();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }    FileOutputStream out;  **try** {  out = **new** FileOutputStream(imgFile);  **if** (bitmap.compress(Bitmap.CompressFormat.***PNG***, 70, out))  {  out.flush();  out.close();  }  } **catch** (FileNotFoundException e)  {  e.printStackTrace();  } **catch** (IOException e)  {  e.printStackTrace();  }  } |

|  |
| --- |
| **public** **static** **void** writeImageToDisk(**byte**[] img, String url) {  Logger.*i*(*TAG*, "writeImageToDisk ", "oddshou");  **if** (Environment.*getExternalStorageState*().equals(Environment.***MEDIA\_MOUNTED***)) {  File path = Environment.*getExternalStorageDirectory*();  Logger.*i*(*TAG*, "writeImageToDisk " + path, "oddshou");  **try** {  File file = **new** File(path + url);  FileOutputStream fops = **new** FileOutputStream(file);  fops.write(img);  fops.flush();  fops.close();  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

### 7功能：shape、selector、layer-list 总结

出处《http://blog.csdn.net/brokge/article/details/9713041》

**1、Shape：**

<shape>  Android:shape=["rectangle" | "oval" | "line" | "ring"]

*其中rectagle矩形，oval椭圆，line水平直线，ring环形*

**<shape>中子节点的常用属性：**

**<gradient>**  *渐变*

Android:startColor

起始颜色

Android:endColor

*结束颜色*

Android:angle

*渐变角度，0从左到右，90表示从下到上，数值为45的整数倍，默认为0；*

Android:type

*渐变的样式 liner线性渐变 radial环形渐变 sweep*

**<solid >**  *填充*

Android:color

*填充的颜色*

**<stroke >***描边*

Android:width

*描边的宽度*

Android:color

*描边的颜色*

Android:dashWidth

*表示'-'横线的宽度*

Android:dashGap

*表示'-'横线之间的距离*

**<corners >***圆角*

Android:radius

*圆角的半径 值越大角越圆*

Android:topRightRadius

*右上圆角半径*

Android:bottomLeftRadius

*右下圆角角半径*

Android:topLeftRadius

*左上圆角半径*

Android:bottomRightRadius

*左下圆角半径*

**<padding >***内边距*

android:bottom="1.0dip"

*底部填充*

android:left="1.0dip"

*左边填充*

android:right="1.0dip"

*右边填充*

android:top="0.0dip"

*上面填充*

**2.selector**

android:state\_selected *是选中*

android:state\_focused *是获得焦点*

android:state\_pressed *是点击*

android:state\_enabled *是设置是否响应事件,指所有事件*

*另：*

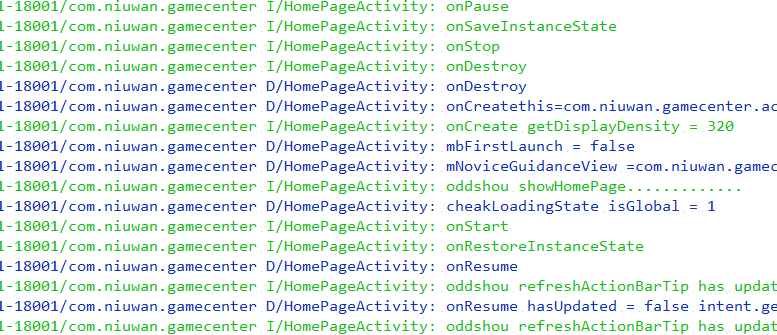
android:state\_window\_focused *默认时的背景图片*

**3.layer-list 将多个图片或上面两种效果安装顺序层叠起来**

### 8.configChanges

配置中设置configChanges， 当相应配置发生，只调用onConfigurationChanged，不在从oncreate重新调用

Oncreate参数bundle，当配置发生，没有忽略这个配置，则会关闭并保存当前数据，重新运行生命周期，此时参数和onrestoreInstanceState中参数应是一致的。



### 9.播放声音相关api SoundPool

### 10.handler内存泄漏（WeakReference）

|  |
| --- |
| public class AutoActivity extends Activity {    MyHandler handler = new MyHandler(this);  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_auto);  }    static class MyHandler extends Handler{  WeakReference<AutoActivity> mactivity;    public MyHandler(AutoActivity activity){  mactivity = new WeakReference<AutoActivity>(activity);  }    @Override  public void handleMessage(Message msg) {  super.handleMessage(msg);  switch (msg.what) {  case 100:  //在这里面处理msg  //通过mactivity.get()获取Activity的引用(即上下文context)  break;  default:  break;  }  }  }  } |

## 注意点

### 1.Launch Mode

singleTask 表示当前栈只会有一个该Activity实例，再次启动会将在其上面栈的Activity 挤掉。

standard 再次启动，不断新建并叠加。

singleTop 如果该Activity在栈顶，则不再新创建，会将该Intent发送给该Activity，执行onNewIntent方法。

singleInstance 该Activity会单独开启一个新的栈

Intent.FLAG\_ACTIVITY\_CLEAR\_TASK 必须结合Intent.FLAG\_ACTIVITY\_NEW\_TASK 一起使用，这个intent会清除已经启动的该Activity以及它后面启动的Activity，并重新启动

在 startActivityForResult 如果设置 为 singleInstance 将 无法 返回

这两个LaunchMode标识只能用在startActivity()的方法中，而不能使用在startActivityForResult方法中。因为从Task的角度看，Android认为不同Task之间的Activity是不能传递数据的。所以也不会有有用的result返回。

#### IntentFlag

LAG\_ACTIVITY\_NEW\_TASK：例如现在栈1的情况是：A B C。C通过intent跳转到D，并且这个intent添加了FLAG\_ACTIVITY\_NEW\_TASK 标记，如果D这个Activity在Manifest.xml中的声明中添加了Task affinity，并且和栈1的affinity不同，系统首先会查找有没有和D的Task affinity相同的task栈存在，如果有存在，将D压入那个栈，如果不存在则会新建一个D的affinity的栈将其压入。如果D的Task affinity默认没有设置，或者和栈1的affinity相同，则会把其压入栈1，变成：A B C D，这样就和不加FLAG\_ACTIVITY\_NEW\_TASK 标记效果是一样的了。 注意如果试图从非activity的非正常途径启动一个activity，比如从一个service中启动一个activity，则intent比如要添加FLAG\_ACTIVITY\_NEW\_TASK 标记。

#### Action.Main 和 category.Launcher的关系

android.intent.action.MAIN： Start as a main entry point, does not expect to receive data.

Input: nothing

Output: nothing

作为一个主入口，没有输入值也没有输出值

android.intent.category.LAUNCHER 表示为一个桌面入口，作为桌面入口必须有android.intent.action.MAIN 的Action

#### android 资源文件位置D:\android-sdk-windows\platforms\android-4\data\res\layout

#### Android 事件传递

#### setEnable 会使点击事件失效

#### jni实现流程 Macintosh HD:Users:houlinhui:personal:图片收藏:JNI.jpg

#### PendingIntent.getActivity 重复

|  |
| --- |
| PendingIntent pendingIntentOneKey = PendingIntent.*getActivity*(App.*getAppContext*(),  0, intentOneKey, PendingIntent.***FLAG\_UPDATE\_CURRENT***); |

以上代码使用在通知（一键下载）发现第二个参数如果一致重复调用会返回用一个PendingIntent.

## 大点

### 1.fragment：

#### 使用方式，xml中定义 FrameLayout，调用FragmentTransaction replace方法

|  |
| --- |
| <FrameLayout  android:id=*"@+id/frameDetail"*  android:layout\_width=*"match\_parent"*  android:layout\_height=*"match\_parent"* >  </FrameLayout>  **private** **void** setFragment() {  FragmentManager fmg = getFragmentManager();  FragmentTransaction ftc = fmg.beginTransaction();  mFragDetail = **new** FragmentDetail(mCurrentTime, mCurrentFrom);  ftc.replace(R.id.***frameDetail***, mFragDetail);  ftc.commit();  } |

#### fragment 生命周期与 activity 生命周期关系



### 2.ScrollView

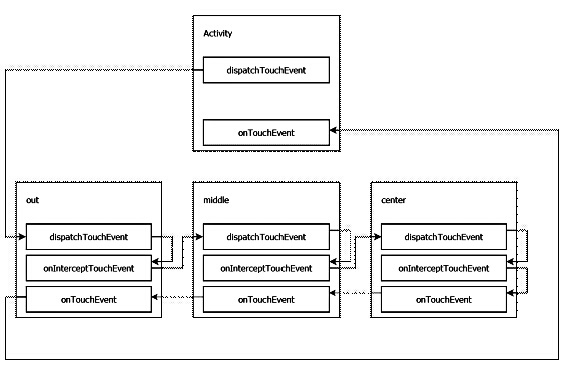
setFillViewport(boolean) scrollView 设置当前滚动视图是否将内容高度拉伸以填充视图可视范围

### 3.View

#### 事件

在一个view中顺序依次1、dispatchTouchEvent，2、 setOnTouchListener的onTouch，3、onTouchEvent。 如果mOnTouchListener.onTouch(this, event)返回true则onTouchEvent(event）不会被执行了。

Viewgroup事件拦截



默认是不拦截的，即返回false；如果你需要拦截，只要return true就行了，这要该事件就不会往子View传递了，并且如果你在DOWN retrun true ，则DOWN,MOVE,UP子View都不会捕获事件。

requestDisallowInterceptTouchEvent(boolean) 用于设置是否允许拦截

1. getParent().requestDisallowInterceptTouchEvent(**true**);

总结：

1、如果ViewGroup找到了能够处理该事件的View，则直接交给子View处理，自己的onTouchEvent不会被触发；

2、可以通过复写onInterceptTouchEvent(ev)方法，拦截子View的事件（即return true），把事件交给自己处理，则会执行自己对应的onTouchEvent方法

3、子View可以通过调用getParent().requestDisallowInterceptTouchEvent(true);  阻止ViewGroup对其MOVE或者UP事件进行拦截；

#### 重要方法说明

**protected** **boolean** overScrollBy(**int** deltaX, **int** deltaY,

**int** scrollX, **int** scrollY,

**int** scrollRangeX, **int** scrollRangeY,

**int** maxOverScrollX, **int** maxOverScrollY,

**boolean** isTouchEvent)

该方法就相当于在scrollTo/By的基础上添加了对overScroll情况的处理, 但父类view中只处理数据,没有实际的scroll操作,父类view处理完数据后将其记录在onOverScrolled方法中,  
子类继承onOverScrolled方法再根据得到的数据scrollTo/By处理即可~

scrollTo() 和scrollBy()方法是将控件内容移动而不是控件本身

offsetLeftAndRight(int offset)用于移动控件本身

#### 重要类说明

Scroller：

Scroller只是用于跟踪控件滑动的轨迹,此类不会移动控件,只相当于一个滚动轨迹记录工具所以需要你在View的一个回调函数computerScroll()中使用Scroller对象获取滑动的数据,  
最后还是用scrollTo/By方法来实际控制某个View的滚动行为

#### ImageView

Backgroud和src 的区别

background会根据ImageView组件给定的长宽进行拉伸，而src就存放的是原图的大小，不会进行拉伸 。src是图片内容（前景），bg是背景，可以同时使用

### 4.屏幕适配

#### Values

values-v11代表在API 11+的设备上，用该目录下的styles.xml代替res/values/styles.xml

values-v14代表在API 14+的设备上，用该目录下的styles.xml代替res/values/styles.xml

其中API 11+代表android 3.0 +

其中API 14+代表android 4.0 +

### 5.Intent、Intent-filter

Intent

packageManager 过滤相关方法：queryIntentActivities()，service，broadcast

intent-filter 必须包含action可以包含category和data

启动有两种方式，显示，隐式

除了使用setComponent()之外, 还可以使用setClass(),setClassName()来显式指定目标组件, 还可以调用getComponent()方法获得Intent中封装的ComponentName对象。

当程序采用这种形式启动组件时, 在Intent中明确的指定了待启动的组件类, 此时的Intent属于显式intent, 显式Intent应用场合比较狭窄, 多用于启动本应用中的component,因为这种方式需要提前获知目标组件类的全限定名。而隐式Intent则通过Intent中的action,category, data属性指定目标组件需要满足的若干条件, 系统筛选出满足所有条件的component,从中选择最合适的component或者由用户选择一个component作为目标组件启动。

如果Intent中指定了ComponentName属性, 则Intent的其他属性将被忽略。

总结：指定component 为显示，没有指定component为隐式（通过action）

自定义的action应该以application的包名作为前缀, 然后附加特定的大写字符串。这才是命名良好的action

Intent对象中可以包含任意多个category属性，Intent类的addCategory()方法为Intent添加Category属性,getCategories()方法用于获取Intent中封装的所有category

intent最多只能定义1个action, 而filter可以定义1个或多个action。

如果filter没有定义action, 则这个filter将阻塞所有intent。如果intent没有定义action, 那么只要filter定义了action就可以通过action测试。

解释：filter可以有多个action，表示这个组件可以处理多个动作，intent最多只能包含一个动作，测试通过即为这个动作可以被处理（所有可以被处理的动作都包含在filter中）

category

测试category属性。 intent可以任意多个category,filter也可以任意个category。通过category测试的条件为: filter定义了intent的所有category，注意。以startActivity(intent)，startActivityForResult()方式启动一个activity时, 系统会为intent增加一个值为"android.intent.category.DEFAULT"的category

任何一个需要隐式启动的Activity都必须要有这项:<category android:name="android.intent.category.DEFAULT"/>

例外情况是：android.intent.category.MAIN和android.intent.category.LAUNCHER的filter中没有必要加入android.intent.category.DEFAULT，当然加入也没有问题

解释：filter满足所有的intent所指定的category

3。测试data属性。 intent最多只能定义1个data,filter则可以定义多个data。

通过data测试的条件为:

uri格式：<scheme>://<host>:<port>[<path>|<pathPrefix>|<pathPattern>]

* If a [scheme](http://developer.android.com/guide/topics/manifest/data-element.html#scheme) is not specified for the intent filter, all the other URI attributes are ignored.
* scheme 没有指定，整个uri失效
* If a [host](http://developer.android.com/guide/topics/manifest/data-element.html#host) is not specified for the filter, the port attribute and all the path attributes are ignored.
* If a scheme is not specified, the host is ignored.
* If a host is not specified, the port is ignored.
* If both the scheme and host are not specified, the path is ignored.
* If a filter specifies only a scheme, all URIs with that scheme match the filter.
* If a filter specifies a scheme and an authority but no path, all URIs with the same scheme and authority pass the filter, regardless of their paths.
* If a filter specifies a scheme, an authority, and a path, only URIs with the same scheme, authority, and path pass the filter.

**Note:** A path specification can contain a wildcard asterisk (\*) to require only a partial match of the path name.

1. An intent that contains neither a URI nor a MIME type passes the test only if the filter does not specify any URIs or MIME types.
2. An intent that contains a URI but no MIME type (neither explicit nor inferable from the URI) passes the test only if its URI matches the filter's URI format and the filter likewise does not specify a MIME type.
3. An intent that contains a MIME type but not a URI passes the test only if the filter lists the same MIME type and does not specify a URI format.
4. An intent that contains both a URI and a MIME type (either explicit or inferable from the URI) passes the MIME type part of the test only if that type matches a type listed in the filter. It passes the URI part of the test either if its URI matches a URI in the filter or if it has a content: or file: URI and the filter does not specify a URI. In other words, a component is presumed to support content: and file: data if its filter lists *only* a MIME type.

d,翻译：

包含了URI和数据类型的Intent对象（或者是数据类型能够从URI中推断出来）只有它的类型跟过滤器中列出的一个类型匹配，才能通过数据类型部分的检测，如果它的URI部分跟过滤器中的一个URI匹配或者Intent对象有一个content:或file:URI并且过滤器没有指定URI，那么才能能够URI部分的检测。换句话说，如果过滤器仅列出了数据类型，那么一个组件被假设为支持content:和file:数据

data属性是一个URI, URI中包含scheme,host, post和path, 典型的URI为:

scheme://host:port/path

scheme, host, post和path都是可选的。比较2个data时, 只比较filter中包含的部分。比如filter的一个data只是指定了scheme部分, 则测试时只是比较data的scheme部分, 只要两者的scheme部分相同, 就视为"相同的data"。

在<intent-filter></intent-filter>里可以有多个<data android:mimeType="xxxx"/>，只需匹配其中一个即可.注意:不可以同时出现下面这条：

在<intent-filter></intent-filter>里可以有多个<data android:scheme="xxxx" android:host="yyyy" android:port="uuu"/>，只需匹配其中一个即可。

当隐式intent发送时，有可能没有组件响应，需要用以下方式判断，否则会crash

// Create the text message with a string  
Intent sendIntent = new Intent();  
sendIntent.setAction(Intent.ACTION\_SEND);  
sendIntent.putExtra(Intent.EXTRA\_TEXT, textMessage);  
sendIntent.setType("text/plain");  
  
// Verify that the intent will resolve to an activity  
if (sendIntent.resolveActivity(getPackageManager()) != null) {  
    startActivity(sendIntent);  
}

[ACTION\_VIEW](http://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW) 一般表示打开

[ACTION\_SEND](http://developer.android.com/reference/android/content/Intent.html#ACTION_SEND)一般表示分享

当需要打开一个选择框时，需要给intent加一个dialo选择哪个应用

Intent chooser = Intent.createChooser(sendIntent, title);

* The [ACTION\_MAIN](http://developer.android.com/reference/android/content/Intent.html#ACTION_MAIN) action indicates this is the main entry point and does not expect any intent data.
* The [CATEGORY\_LAUNCHER](http://developer.android.com/reference/android/content/Intent.html#CATEGORY_LAUNCHER) category indicates that this activity's icon should be placed in the system's app launcher. If the [<activity>](http://developer.android.com/guide/topics/manifest/activity-element.html) element does not specify an icon with icon, then the system uses the icon from the [<application>](http://developer.android.com/guide/topics/manifest/application-element.html) element.
* These two must be paired together in order for the activity to appear in the app launcher

PendingIntent使用途径主要有3种：Notification，app widget，alarmManager

Broadcast

广播接收可以在主线程中，但要注意耗时太长，阻塞主线程

发送广播：[sendBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)), [sendOrderedBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent, java.lang.String)), or [sendStickyBroadcast()](http://developer.android.com/reference/android/content/Context.html#sendStickyBroadcast(android.content.Intent))

发送有序广播时，可以按设置的priority 顺序接收，并决定是否向下继续传递。

无论对于有序广播还是无序广播，广播接收器默认都是运行在主线程中的(main线程，即UI线程)。可以通过在程序中使用registerReceiver(receiver, filter, broadcastPermission, scheduler)方法中的最后一个参数指定要运行的广播接收器的线程。也可以在Manifest.xml文件中设置(Intent-filter标签中设置android:process)

有序广播可以通过setResultExtras(Bundle bundle)添加参数并向下继续传递

发送广播时可以添加自定义权限，加入方式：现在AndroidManifest中注册权限，

<permission android:protectionLevel="normal" android:name="xzq.zhongnan.test"></permission>

发送时使用（sendBroadcast (Intent intent, String receiverPermission)），接收方添加权限即可。

<uses-permission android:name="xzq.zhongnan.test"></uses-permission

Service

Android 5.0 通过隐式启动service ， [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int))会抛出异常，官方解释是用户不确定哪个服务会响应这个intent

Activity

activity栈：

The device Home screen is the starting place for most tasks. When the user touches an icon in the application launcher (or a shortcut on the Home screen), that application's task comes to the foreground. If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

翻译：点击icon，应用程序的任务栈会返回前台，如果没有栈，则会创建新的栈，并使main activity作为栈的root Activity

[onSaveInstanceState()](http://developer.android.com/reference/android/app/Activity.html#onSaveInstanceState(android.os.Bundle))当activity在战中被destroy，返回到对应栈时，会重新创建，这个时候需要通过这个方法保存状态，以便下次创建时使用

### Activity

1. 生命周期

Resumed activity在foreground, 并且获取用户焦点，表示正在运行

Paused 另一个activity处在前台，获取焦点，但是当前activity依然可见，比如前台activity部分透明，或者没有完全覆盖整个屏幕，此时这个activity依然保持运行，并且attach到window manager。但是系统内存极度缺乏的时候可以被杀死

Stopped 完全被遮掩，still alive 但是没有attach 到window manager，系统其他地方需要内存时可以被回收

Activity处在paused或者stopped，系统会回收内存通过调用finish()方法，或者杀掉进程如果activity需要打开，activity需要重新create



Activity 第一个环，主体生命周期：从oncreate到 ondestroy， ondestroy中注意 oncreate 中开启的线程任务需要终止。

第二个环，可见生命周期：从onstart到onstop，广播可以在onstart中注册，onstop注销，

第三个环，前台生命周期：onresume，onpause，设备休眠，弹起dialog执行onpause。因为这两个状态切换频繁，因此只能做一些轻量操作。

生命周期方法说明：

Oncreate create view、bind data，函数参数一般存储activity之前的状态

Onpause 存储一些状态，停止动画，

Ondestroy 执行有两种情况，一种是被执行了finish()方法，一种是系统临时回收，区别可以通过 isFinishing()方法判断  
注意：Onstart 可以直接onStop，当它被hide的时候。

注意：onpause，onstop，ondestroy执行完后都有可能被kill掉，因此如果应用被紧急回收，你应该在onpause中保存重要数据

Android 进程（Processes），线程（Thread）

Processes 四大组件都可以通过process属性在manifest中配置独立进程

## Java基础

### 1. 类加载顺序:

JAVA类首次装入时，会对静态成员变量或方法进行一次初始化,但方法不被调用是不会执行的， 静态成员变量和静态初始化块级别相同，非静态成员变量和非静态初始化块级别相同。先初始化父类的静态代码--->初始化子类的静态代码-->初始化父类的非静态代码--->初始化父类构造函数--->初始化子类非静态代码--->初始化子类构造函数.  
虽没有显示调用父类构造函数，但是子类构造函数默认调用父类无参构造函数。

### 2. static

### 3.内部类

内部类创建：静态内部类可以直接创建，非静态内部类需要一个外部类的引用创建（object.new Inner()，使用.new 语法），而且非静态内部类不可用有静态成员。