[AndroidManifest.xml中一些常用的属性](http://blog.csdn.net/ygc87/article/details/7335647)

一、关于AndroidManifest.xml  
AndroidManifest.xml 是每个android程序中必须的文件。它位于整个项目的根目录，描述了package中暴露的组件（activities, services, 等等），他们各自的实现类，各种能被处理的数据和启动位置。 除了能声明程序中的Activities, ContentProviders, Services, 和BroadcastReceivers,还能指定permissions和instrumentation（安全控制和[**测试**](http://lib.csdn.net/base/softwaretest)）

二、AndroidManifest.xml结构

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><?**xml**xml** ="1.0"encoding="utf-8"**?>**
2. **<manifest** **>**
4. **<application** **>**
5. **<activity** **>**
6. **<intent-filter** **>**
7. **<action** **/>**
9. **<category** **/>**
10. **</intent-filter>**
11. **</activity>**
13. **<activity-alias** **>**
14. **<intent-filter** **>**
15. **</intent-filter>**
17. **<meta-data** **/>**
18. **</activity-alias>**
20. **<service** **>**
21. **<intent-filter** **>**
22. **</intent-filter>**
24. **<meta-data** **/>**
25. **</service>**
27. **<receiver** **>**
28. **<intent-filter** **>**
29. **</intent-filter>**
31. **<meta-data** **/>**
32. **</receiver>**
34. **<provider** **>**
35. **<grant-uri-permission** **/>**
37. **<meta-data** **/>**
38. **</provider>**
40. **<uses-library** **/>**
41. **</application>**
43. **<uses-permission** **/>**
45. **<permission** **/>**
47. **<permission-tree** **/>**
49. **<permission-group** **/>**
51. **<instrumentation** **/>**
53. **<uses-sdk** **/>**
55. **<uses-configuration** **/>**
57. **<uses-feature** **/>**
59. **<supports-screens** **/>**
61. **</manifest></span>**

三、各个节点的详细介绍  
上面就是整个AndroidManifest.xml的结构，下面逐一进行详细介绍

1、第一层(<Manifest>):(属性)

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><manifest** xmlns:android="http://schemas.android.com/apk/res/android"
2. package="com.woody.test"
3. android:sharedUserId="string"
4. android:sharedUserLabel="string resource"
5. android:versionCode="integer"
6. android:versionName="string"
7. android:installLocation=["auto" | "internalOnly" | "preferExternal"]
8. **>**
10. **</manifest></span>**

A、xmlns:android  
定义android命名空间，一般为<http://schemas.android.com/apk/res/android>，这样使得Android中各种标准属性能在文件中使用，提供了大部分元素中的数据。

自定义xmlns：

Android也允许我们自定义xmlns，这样我们就可以使用我们自己定义的一些属性。下面是一个小例子：

首先，我们需要在res/values目录下创建一个attrs.xml文件，然后在其中自定义一些属性，attrs.xml文件内容如下：

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><?xml** version="1.0" encoding="utf-8"**?>**
2. **<resources>**
4. **<declare-styleable** name="RedefineAttrs"**>**
5. **<attr** format="color" name="redefineTextColor" **/>**
6. **<attr** format="dimension" name="redefineTextSize" **/>**
7. **<attr** format="string" name="redefineText" **/>**
8. **</declare-styleable>**
10. **</resources></span>**

关于其中各个标签的详细说明，请见另一篇博文——Android自定义属性时format选项可以取用的值。

然后是我们的自定义View：

**[java]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. <span style="font-size:18px;">**public** **class** RedefineView **extends** TextView {
3. **public** RedefineView(Context context, AttributeSet attrs) {
4. **super**(context, attrs);
5. TypedArray a = context.obtainStyledAttributes(attrs,
6. R.styleable.RedefineAttrs);
7. //通过如下方法来获取每一个属性的值
8. **int** textColor = a.getColor(R.styleable.RedefineAttrs\_redefineTextColor,
9. 0XFFFFFFFF);
10. **float** textSize = a.getDimension(
11. R.styleable.RedefineAttrs\_redefineTextSize, 36);
12. String mString = a.getString(R.styleable.RedefineAttrs\_redefineText);
13. **if** (mString == **null** || mString.length() == 0) {
14. mString = "null";
15. }
17. //把这些属性设置到TextView中
18. setText(mString);
19. setTextSize(textSize);
20. setTextColor(textColor);
21. }
22. }
23. </span>

最后是我们的布局文件：

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><?xml** version="1.0" encoding="utf-8"**?>**
2. **<LinearLayout** xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:redefineAttrs="http://schemas.android.com/apk/res/com.kingsoft"
4. android:layout\_width="fill\_parent"
5. android:layout\_height="fill\_parent"
6. android:orientation="vertical" **>**
8. **<com.kingsoft.RedefineView**
9. android:layout\_width="fill\_parent"
10. android:layout\_height="wrap\_content"
11. redefineAttrs:redefineText="@string/hello"
12. redefineAttrs:redefineTextColor="#ff456633"
13. redefineAttrs:redefineTextSize="20px" **/>**
15. **</LinearLayout></span>**

其中xmlns:redefineAttrs="<http://schemas.android.com/apk/res/com.kingsoft>"至关重要，绿色部分是命名空间的名称，用于引用其中属性的前缀，如代码中的

        redefineAttrs:redefineText="@string/hello"  
        redefineAttrs:redefineTextColor="#ff456633"  
        redefineAttrs:redefineTextSize="20px"

红色部分为程序的包名。

B、package  
指定本应用内[**Java**](http://lib.csdn.net/base/java)主程序包的包名，它也是一个应用进程的默认名称。

C、sharedUserId  
表明数据权限，因为默认情况下，Android给每个APK分配一个唯一的UserID，所以是默认禁止不同APK互相访问数据的。若要共享数据，第一可以采用Share Preference（或Content Provider）方法，第二种可以采用设置相同的sharedUserId，将不同APK的sharedUserId都设为一样，这些APK之间就可以共享数据了。

拥有相同sharedUserId的多个APK可以配置成运行在同一个进程中，所以默认就是可以互相访问数据的； 也可以配置成运行在不同的进程, 同时可以访问其他APK的[**数据库**](http://lib.csdn.net/base/mysql)和文件，就像访问本程序的数据一样。举个例子：APK A 和APK B 都是C公司的产品，那么如果用户从APK A中登陆成功，那么打开APK B的时候就不用再次登陆。 具体实现就是 A和B设置成同一个sharedUserId：  
在2个APK的AndroidManifest.xml 配置sharedUserId:

APK A 的AndroidManifest.xml文件

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><?xml** version="1.0" encoding="utf-8"**?>**
2. **<manifest** xmlns:android="http://schemas.android.com/apk/res/android"
3. package="com.android.demo.a1"
4. android:sharedUserId="com.c" **>**
6. **</manifest></span>**

APK B的AndroidManifest.xml文件

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**><?xml** version="1.0" encoding="utf-8"**?>**
2. **<manifest** xmlns:android="http://schemas.android.com/apk/res/android"
3. package="com.android.demo.b1"
4. android:sharedUserId="com.c" **>**
6. **</manifest></span>**

这个设定好之后，APK B就可以像打开本地数据库那样打开APK A中的数据库了。APK A把登陆信息存放在A的数据目录下面，APK B每次启动的时候读取APK A下面的数据库判断是否已经登陆:  
APK B中的代码:

**[java]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. <span style="font-size:18px;">      **try** {
2. Context friendContext = **this**.createPackageContext(
3. "com.android.demo.a1", Context.CONTEXT\_IGNORE\_SECURITY);
4. } **catch** (NameNotFoundException e) {
5. // TODO Auto-generated catch block
6. e.printStackTrace();
7. }</span>

通过A的package name 就可以得到A的packagecontext，通过这个packagecontext就可以直接打开A的数据库。

D、sharedUserLabel  
一个用户可读的共享ID标签，它只有在设置了sharedUserId属性的前提下才会有意义

E、versionCode  
一个内部版本号。

F、versionName  
这个名称是给用户看的，你可以将你的APP版本号设置为1.1版，后续更新版本设置为1.2、2.0版本等等。

G、installLocation  
安装参数，是Android2.2中的一个新特性，installLocation有三个值可以选择：internalOnly、auto、preferExternal。  
选择preferExternal，系统会优先考虑将APK安装到SD卡上(当然最终用户可以选择为内部ROM存储上，如果SD存储已满，也会安装到内部存储上)；选择auto，系统将会根据存储空间自己去适应；选择internalOnly是指必须安装到内部才能运行(注：需要进行后台类监控的APP最好安装在内部，而一些较大的游戏APP最好安装在SD卡上。现默认为安装在内部，如果把APP安装在SD卡上，首先得设置你的level为8，并且要配置android:installLocation这个参数的属性为preferExternal)。

2、第二层(<Application>):属性  
一个AndroidManifest.xml中必须含有一个Application标签，它包含一些子元素用来声明每一个Application的组件，它也包含一些能作用于所有组件的属性值。其中的一些（例如icon,label,permission,process,taskAffinity, 和allowTaskReparenting）会为组件元素对应的属性设置默认值。其它的（例如debuggable,enabled,description, 和allowClearUserData）一般来说是为Application设置属性，并且组件不能覆盖这些属性值。

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**    **<application**
2. android:allowClearUserData=["true" | "false"]
3. android:allowTaskReparenting==["true" | "false"]
4. android:backupAgent="string"
5. android:debuggable==["true" | "false"]
6. android:description="string resource"
7. android:enabled==["true" | "false"]
8. android:hasCode==["true" | "false"]
9. android:icon="drawable resource"
10. android:killAfterRestore==["true" | "false"]
11. android:label="string resource"
12. android:manageSpaceActivity="string"
13. android:name="string"
14. android:permission="string"
15. android:persistent==["true" | "false"]
16. android:process="string"
17. android:restoreAnyVersion==["true" | "false"]
18. android:taskAffinity="string"
19. android:theme="resource or theme" **>**
20. **</application></span>**

A、android:allowClearUserData('true' or 'false')  
用户是否能选择自行清除数据，默认为true，程序管理器包含一个选择允许用户清除数据。当为true时，用户可自己清理用户数据，反之不可以。

B、android:allowTaskReparenting('true' or 'false')  
是否允许activity更换从属的任务，比如从短信息任务切换到浏览器任务。

C、android:backupAgent  
实现了BackupAgent的子类名称。这个属性值应该是一个完整的类名（例如："com.example.project.MyBackupAgent"）。但是，一种简便的写法，如果这个类名的第一个字符是句点，（例如：".MyBackupAgent"），它将会被追加到manifest标签中指定的包名后面。它没有默认值，必须明确说明。

D、android:debuggable  
这个从字面上就可以看出是什么作用的，当设置为true时，表明该APP在手机上可以被调试，默认为false。

E、android:description/android:label  
android:description：用户可以看见的有关应用程序的文本，比应用程序标签更详细的描述。它的值必须是一个string的资源引用。和标签不同，它不可以是一个string常量值。没有默认值。

android:label：用户可见的关于应用程序的总体文本。

F、android:enabled  
Android系统是否能够实例化该应用程序的组件，如果为true，每个组件的enabled属性决定那个组件是否可以被 enabled。如果为false，它覆盖组件指定的值；所有组件都是disabled。

G、android:hasCode('true' or 'false')  
表示此APP是否包含任何的代码，默认为true，若为false，则系统在运行组件时，不会去尝试加载任何的APP代码，  
一个应用程序自身不会含有任何的代码，除非内置组件类，比如Activity类，此类使用了AliasActivity类，当然这是个罕见的现象。  
(在Android2.3可以用标准C来开发应用程序，可在androidManifest.xml中将此属性设置为false,因为这个APP本身已经不含有任何的JAVA代码了)

H、android:icon  
这个很简单，就是声明整个APP的图标，图片一般都放在drawable文件夹下。

I、android:killAfterRestore

在整个系统恢复出厂设置期间，应用程序设置重建时是否终止应用程序。单一的应用程序恢复设置时通常不会引起程序的终止。整个系统恢复出厂设置通常只在第一次使用手机时才会发生。第三方应用程序通常不需要设置次属性。（翻译自<http://developer.android.com/guide/topics/manifest/application-element.html>，可能有理解不正确的地方，欢迎指正。）

J、android:manageSpaceActivity

一个系统可以启动让用户来管理这个应用程序的内存使用的Activity子类的完整类名。这个Activity必须在<activity>标签中声明。

K、android:name  
Application子类的全名。当应用程序进程开始时，该类在所有应用程序组件之前被实例化。若该类(比方androidMain类)是在声明的package下，则可以直接声明android:name="androidMain"，但如果此类是在package下面的子包中的话，就必须声明为全路径android:name="package名称.子包名成.androidMain"。

L、android:permission  
设置许可名，这个属性若在<application>上定义的话，是一个给应用程序的所有组件设置许可的便捷方式，当然它是可以被各组件设置的许可名所覆盖的。

M、android:presistent  
该应用程序是否应该在任何时候都保持运行状态,默认为false。因为应用程序通常不应该设置本标识，持续模式仅仅应该设置给某些系统应用程序才是有意义的。

N、android:process  
应用程序运行的进程名，它的默认值为<manifest>元素里设置的包名，当然每个组件都可以通过设置该属性来覆盖默认值。如果你想两个应用程序共用一个进程的话，你可以设置他们的android:process相同，但前提条件是他们共享一个用户ID及被赋予了相同证书的时候。

O、android:restoreAnyVersion  
同样也是android2.2的一个新特性，用来表明应用是否准备尝试恢复所有的备份，甚至该备份是比当前设备上更新的版本，默认是false。

P、android:taskAffinity   
拥有相同的affinity的Activity理论上属于相同的Task，应用程序默认的affinity的名字是<manifest>元素中设定的package名。

Q、android:theme  
是一个资源的风格，它定义了一个默认的主题风格给所有的activity,当然也可以在自己的theme里面去设置它，有点类似style。

3、第三层(<Activity>):属性

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**        **<activity**
2. android:allowTaskReparenting=["true" | "false"]
3. android:alwaysRetainTaskState=["true" | "false"]
4. android:clearTaskOnLaunch=["true" | "false"]
5. android:configChanges=["mcc", "mnc", "locale","touchscreen", "keyboard", "keyboardHidden","navigation", "orientation", "screenLayout","fontScale", "uiMode"]
6. android:enabled=["true" | "false"]
7. android:excludeFromRecents=["true" | "false"]
8. android:exported=["true" | "false"]
9. android:finishOnTaskLaunch=["true" | "false"]
10. android:icon="drawable resource"
11. android:label="string resource"
12. android:launchMode=["multiple" | "singleTop" |"singleTask" | "singleInstance"]
13. android:multiprocess=["true" | "false"]
14. android:name="string"
15. android:noHistory=["true" | "false"]
16. android:permission="string"
17. android:process="string"
18. android:screenOrientation=["unspecified" | "user" | "behind" |"landscape" | "portrait" |"sensor" | "nosensor"]
19. android:stateNotNeeded=["true" | "false"]
20. android:taskAffinity="string"
21. android:theme="resource or theme"
22. android:windowSoftInputMode=["stateUnspecified","stateUnchanged", "stateHidden","stateAlwaysHidden", "stateVisible","stateAlwaysVisible", "adjustUnspecified","adjustResize", "adjustPan"] **>**
23. **</activity>**    **</span>**

(注：有些在application中重复的就不多阐述了)

1、android:alwaysRetainTaskState  
系统是否永久保存当前Activity所在的Task的状态。"true"将会保存，"false"系统将允许在明确的情况下把Task的状态重置为初始状态。默认值是"false"。这个属性只对这个Task的root activity有意义，它将忽略其它所有的activity。

通常情况下，当用户从主屏幕重新选择这个Task时系统将清除这个Task(移除root activity之上的所有activity)。通常，如果用户在确定的时间内没有再访问这个Task，例如30分钟，系统将执行清除操作。

可是，当这个属性设置为"true"时，用户将总是回到这个Task的最后状态，无论他们采取任何方式。这是非常有用的，例如，像网页浏览器这样的应用程序有很多的状态（例如有很多tab），用户不想忘记他们在浏览哪个网页。

2、android:clearTaskOnLaunch

无论何时从主屏幕重启这个activity时是否从Task中移除除了root activity的所有activity。这个属性仅仅对那些启动了新的Task的activity有意义（root activity）；它将忽略这个Task中所有其他的activity。

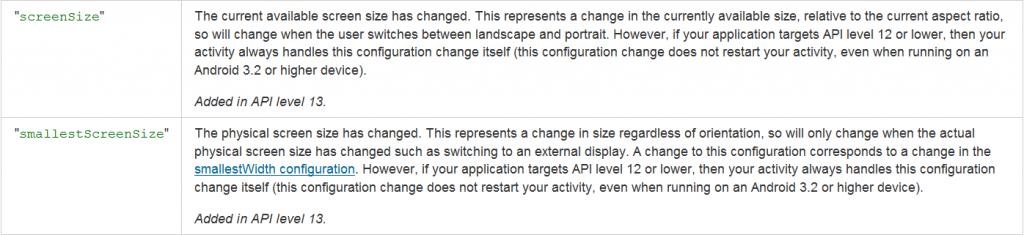
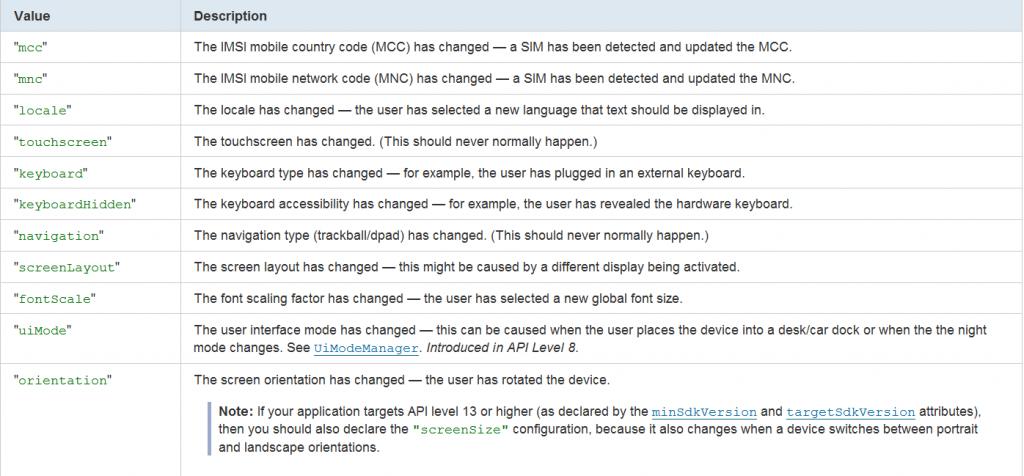
当这个属性值为"true"时，每一次用户重新启动这个Task，都会回到它的root activity，无论它们在这个Task中最后做了什么事，无论是点击back键或者home键离开它。当这个属性值为"false"时，这个Task将会在特定的情况下被清空，但是并不是总会清空。

假如，从主屏幕启动activity P，然后从activity P跳转到activity Q。用户点击home键，然后再启动activity P。通常情况下，用户将会看到activity Q，因为这是在P的Task中的最后的状态。但是，如果P的这个属性值为"true"，当用户点击home键，这个Task转到后台的时候，所有这个activity上面的activity都会被移除。所以，当用户在回到这个Task的时候只会看到P。

3、android:configChanges  
activity会自己处理的configChanges列表。默认情况下，在运行时发生configChanges时，activity会先销毁然后重新启动，但是如果声明了这个属性，就可以阻止activity重新启动。相反，activity会继续运行，然后调用它的 onConfigurationChanged()方法。

注意：应该尽量避免使用这个属性，它应该是你最后的选择。可以阅读android开发文档的 Handling Runtime Changes这篇文章来了解如何正确地处理由于configChanges引起activity重启的更多信息。

下面的所有字符串都是这个属性的可选值。如果是多个值，可以用'|'分割，例如 "locale|navigation|orientation"。



所有这些configChanges都会影响应用程序的可见资源的属性。因此，当 onConfigurationChanged()方法调用时，通常会重新获取所有所需资源(例如视图的layout，drawable等等)来正确地处理这些变化。

4、android:excludeFromRecents  
是否不加入到最近打开的显示给用户的activity列表里，默认是false。

5、android:finishOnTaskLaunch  
当用户重新启动这个任务的时候，是否关闭已打开的activity，默认是false。如果这个属性和allowTaskReparenting都是true，那么只有这个属性起作用。Activity的affinity 属性将被忽略。该Activity已经被摧毁并非可重用的。

6、android:launchMode(Activity加载模式)  
请参见另一篇博文——android中activity的四种加载模式。

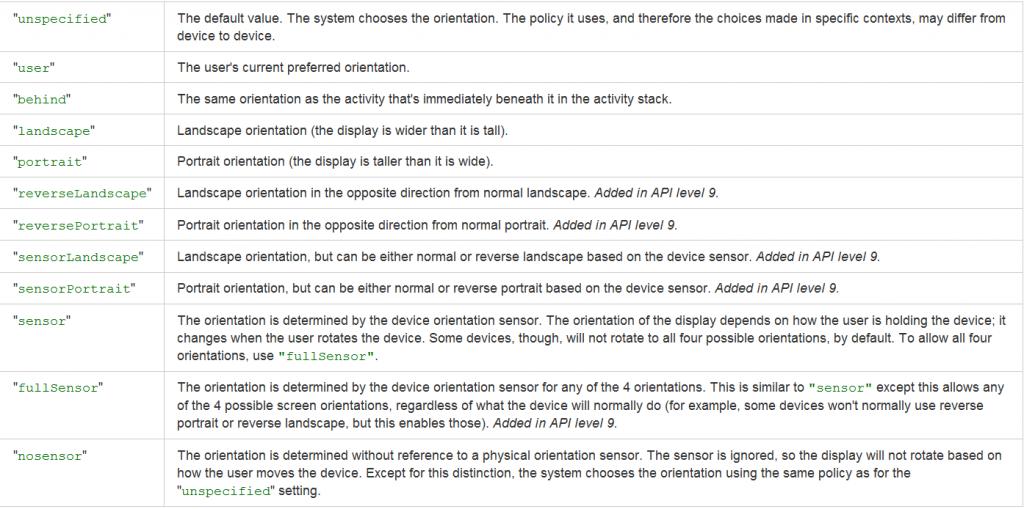
7、android:multiprocess  
是否允许此activity加入到启动它的那个组件的process里。

通常，一个新的activity的实例会加入到应用程序定义的那个process里面，所以这个activity的所有实例都运行在同一个process里面。但是，如果这个属性设置为"true"，这个activity的实例可以运行在多个process里面，允许系统在任何它想使用这个activity的地方创建它的实例，一般情况下，这个属性是不需要的，也是不期望使用的。

8、android:noHistory  
当用户从Activity上离开并且它在屏幕上不再可见时，Activity是否从Activity stack中清除并结束。默认是false。如果设置为''true"，Activity不会留下历史痕迹。它不会在这个Task中保留，用户将不能在这个Task中回到这个activity实例。

9、android:screenOrientation  
activity在设备上显示的方向。

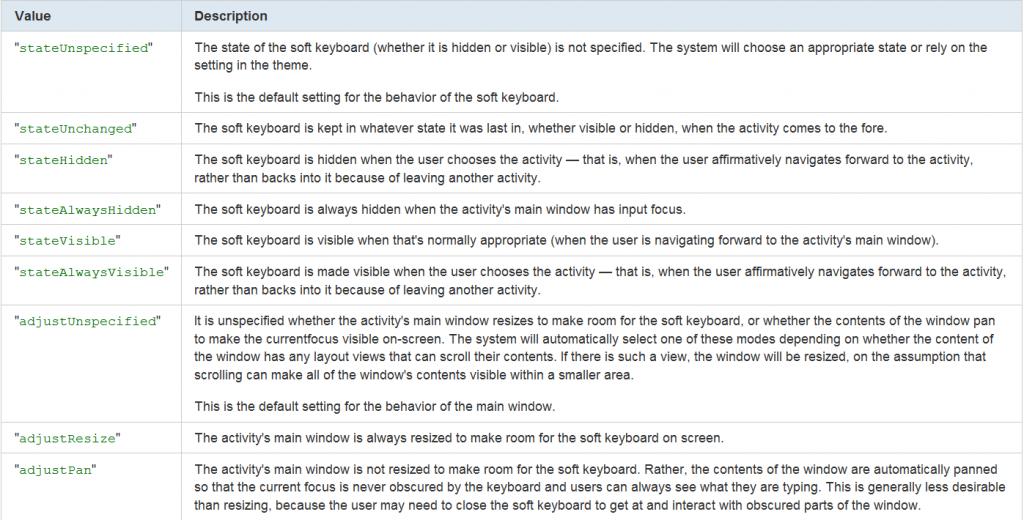
可以是下面字符串中的任意一个值：

   
10、android:stateNotNeeded  
activity是否不需要保存状态就可以被销毁或者重启。

通常，当一个activity保存资源关闭之前，它的onSaveInstanceState() 方法会被调用。这个方法把这个activity的当前状态保存在一个Bundle对象中，当这个activity重新启动的时候传递给onCreate()方法。如果这个属性设置为''true''，onSaveInstanceState() 方法就不会调用，当重新启动这个activity的时候，传递给onCreate()方法的是一个Null值，就像第一次启动这个activity时候一样。

设置为"true"确保activity从原始的状态被启动。例如，显示主界面的activity就使用了这个属性，确保如果由于一些原因导致程序崩溃 时它不会被移除。

11、android:windowSoftInputMode  
activity主窗口与软键盘的交互模式。这个属性能影响两件事情：  
1）软键盘的状态——软键盘是隐藏还是显示——当这个activity变成焦点时。  
2）activity主窗口的调整——是否减少活动主窗口大小以便腾出空间放软键盘，当activity的部分窗口被软键盘覆盖的时候，是否它的内容的当前焦点可见。

可选用的值如下：  


4、第四层(<intent-filter>)  
结构图：

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**            **<intent-filter**
2. android:icon="drawable resource"
3. android:label="string resource"
4. android:priority="integer" **>**
5. **<action** **/>**
7. **<category** **/>**
9. **<data** **/>**
10. **</intent-filter></span>**

intent-filter属性

a，android:icon

一个描述activity，service，broadcast receiver的图标。

b，android:label

一个用户可见的用来描述父组件的标签。  
c，android:priority

这个优先级属性在父组件处理关于过滤器描述的intent类型时将会传递给父组件。这个属性对于activity和broadcast receiver有意义。

它提供了这个activity相对于其他activity想要响应过滤器匹配的这个intent的程度信息。当一个intent能够被多个拥有不同优先级的activity处理时，android系统将会考虑那些具有高优先级的activity作为能够处理这个intent的候选目标。

它控制那些能够处理这个广播信息的broadcast receiver接受这个广播信息的顺序。高优先级的比低优先级的先接受到这个广播信息。（这个原则只对同步的消息有效，对于异步的消息无效。）

只有当你真的需要强制说明broadcast receiver接受这个intent的顺序时在使用这个属性，或者想要强制android系统在其他activity优先执行某个activity。

Intent filter内可以设定的元素包括action,data与category三种，也就是说filter只会与intent里的这三种元素作对比动作。

action元素  
action元素很简单，只有android:name这个属性。常见的android:name值为android.intent.action.MAIN，表明此activity是作为应用程序的入口。有关android:name具体有哪些值，可参照以下内容：

1.从google搜索内容   
Intent intent = new Intent();   
intent.setAction(Intent.ACTION\_WEB\_SEARCH);   
intent.putExtra(SearchManager.QUERY,"searchString")   
startActivity(intent);   
  
2.浏览网页   
Uri uri = Uri.parse("http://www.google.com");   
Intent it  = new Intent(Intent.ACTION\_VIEW,uri);   
startActivity(it);   
  
3.显示地图   
Uri uri = Uri.parse("geo:38.899533,-77.036476");   
Intent it = new Intent(Intent.Action\_VIEW,uri);   
startActivity(it);   
  
4.路径规划   
Uri uri = Uri.parse("http://maps.google.com/maps?f=dsaddr=startLat%20startLng&daddr=endLat%20endLng&hl=en");   
Intent it = new Intent(Intent.ACTION\_VIEW,URI);   
startActivity(it);   
  
5.拨打电话   
Uri uri = Uri.parse("tel:xxxxxx");   
Intent it = new Intent(Intent.ACTION\_DIAL, uri);     
startActivity(it);   
  
6.调用发短信的程序   
Intent it = new Intent(Intent.ACTION\_VIEW);      
it.putExtra("sms\_body", "The SMS text");      
it.setType("vnd.android-dir/mms-sms");      
startActivity(it);   
  
7.发送短信   
Uri uri = Uri.parse("smsto:0800000123");      
Intent it = new Intent(Intent.ACTION\_SENDTO, uri);      
it.putExtra("sms\_body", "The SMS text");      
startActivity(it);   
String body="this is sms demo";   
Intent mmsintent = new Intent(Intent.ACTION\_SENDTO, Uri.fromParts("smsto", number, null));   
mmsintent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_MESSAGE\_BODY, body);   
mmsintent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_COMPOSE\_MODE, true);   
mmsintent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_EXIT\_ON\_SENT, true);   
startActivity(mmsintent);   
  
8.发送彩信   
Uri uri = Uri.parse("content://media/external/images/media/23");      
Intent it = new Intent(Intent.ACTION\_SEND);      
it.putExtra("sms\_body", "some text");      
it.putExtra(Intent.EXTRA\_STREAM, uri);      
it.setType("image/png");      
startActivity(it);   
StringBuilder sb = new StringBuilder();   
sb.append("file://");   
sb.append(fd.getAbsoluteFile());   
Intent intent = new Intent(Intent.ACTION\_SENDTO, Uri.fromParts("mmsto", number, null));   
// Below extra datas are all optional.   
intent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_MESSAGE\_SUBJECT, subject);   
intent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_MESSAGE\_BODY, body);   
intent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_CONTENT\_URI, sb.toString());   
intent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_COMPOSE\_MODE, composeMode);   
intent.putExtra(Messaging.KEY\_ACTION\_SENDTO\_EXIT\_ON\_SENT, exitOnSent);   
startActivity(intent);   
  
9.发送Email   
Uri uri = Uri.parse("mailto:xxx@abc.com");   
Intent it = new Intent(Intent.ACTION\_SENDTO, uri);   
startActivity(it);   
Intent it = new Intent(Intent.ACTION\_SEND);      
it.putExtra(Intent.EXTRA\_EMAIL, "me@abc.com");      
it.putExtra(Intent.EXTRA\_TEXT, "The email body text");      
it.setType("text/plain");      
startActivity(Intent.createChooser(it, "Choose Email Client"));   
Intent it=new Intent(Intent.ACTION\_SEND);        
String[] tos={"me@abc.com"};        
String[] ccs={"you@abc.com"};        
it.putExtra(Intent.EXTRA\_EMAIL, tos);        
it.putExtra(Intent.EXTRA\_CC, ccs);        
it.putExtra(Intent.EXTRA\_TEXT, "The email body text");        
it.putExtra(Intent.EXTRA\_SUBJECT, "The email subject text");        
it.setType("message/rfc822");        
startActivity(Intent.createChooser(it, "Choose Email Client"));      
  
Intent it = new Intent(Intent.ACTION\_SEND);      
it.putExtra(Intent.EXTRA\_SUBJECT, "The email subject text");      
it.putExtra(Intent.EXTRA\_STREAM, "file:///sdcard/mysong.mp3");      
sendIntent.setType("audio/mp3");      
startActivity(Intent.createChooser(it, "Choose Email Client"));   
  
10.播放多媒体     
Intent it = new Intent(Intent.ACTION\_VIEW);   
Uri uri = Uri.parse("file:///sdcard/song.mp3");   
it.setDataAndType(uri, "audio/mp3");   
startActivity(it);   
Uri uri = Uri.withAppendedPath(MediaStore.Audio.Media.INTERNAL\_CONTENT\_URI, "1");      
Intent it = new Intent(Intent.ACTION\_VIEW, uri);      
startActivity(it);   
  
11.uninstall apk   
Uri uri = Uri.fromParts("package", strPackageName, null);      
Intent it = new Intent(Intent.ACTION\_DELETE, uri);      
startActivity(it);   
  
12.install apk   
Uri installUri = Uri.fromParts("package", "xxx", null);   
returnIt = new Intent(Intent.ACTION\_PACKAGE\_ADDED, installUri);   
  
13. 打开照相机   
Intent i = new Intent(Intent.ACTION\_CAMERA\_BUTTON, null);   
this.sendBroadcast(i);   
long dateTaken = System.currentTimeMillis();   
String name = createName(dateTaken) + ".jpg";   
fileName = folder + name;   
ContentValues values = new ContentValues();   
values.put(Images.Media.TITLE, fileName);   
values.put("\_data", fileName);   
values.put(Images.Media.PICASA\_ID, fileName);   
values.put(Images.Media.DISPLAY\_NAME, fileName);   
values.put(Images.Media.DESCRIPTION, fileName);   
values.put(Images.ImageColumns.BUCKET\_DISPLAY\_NAME, fileName);   
Uri photoUri = getContentResolver().insert(   
      MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI, values);   
               
Intent inttPhoto = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);   
inttPhoto.putExtra(MediaStore.EXTRA\_OUTPUT, photoUri);   
startActivityForResult(inttPhoto, 10);   
  
14.从gallery选取图片

Intent i = new Intent();   
i.setType("image/\*");   
i.setAction(Intent.ACTION\_GET\_CONTENT);   
startActivityForResult(i, 11);   
  
15. 打开录音机   
Intent mi = new Intent(Media.RECORD\_SOUND\_ACTION);   
startActivity(mi);   
  
16.显示应用详细列表         
Uri uri = Uri.parse("market://details?id=app\_id");           
Intent it = new Intent(Intent.ACTION\_VIEW, uri);           
startActivity(it);           
//where app\_id is the application ID, find the ID            
//by clicking on your application on Market home            
//page, and notice the ID from the address bar        
  
刚才找app id未果，结果发现用package name也可以   
Uri uri = Uri.parse("market://details?id=<packagename>");   
这个简单多了   
  
17寻找应用         
Uri uri = Uri.parse("market://search?q=pname:pkg\_name");           
Intent it = new Intent(Intent.ACTION\_VIEW, uri);           
startActivity(it);   
//where pkg\_name is the full package path for an application         
  
18打开联系人列表            
Intent i = new Intent();

i.setAction(Intent.ACTION\_GET\_CONTENT);   
i.setType("vnd.android.cursor.item/phone");   
startActivityForResult(i, REQUEST\_TEXT);   
  
Uri uri = Uri.parse("content://contacts/people");   
Intent it = new Intent(Intent.ACTION\_PICK, uri);   
startActivityForResult(it, REQUEST\_TEXT);   
  
19 打开另一程序   
Intent i = new Intent();   
ComponentName cn = new ComponentName("com.yellowbook.android2",   
        "com.yellowbook.android2.AndroidSearch");   
i.setComponent(cn);   
i.setAction("android.intent.action.MAIN");   
startActivityForResult(i, RESULT\_OK);

20.调用系统编辑添加联系人（高版本SDK有效）：  
Intent it = newIntent(Intent.ACTION\_INSERT\_OR\_EDIT);  
it.setType("vnd.android.cursor.item/contact");  
//it.setType(Contacts.CONTENT\_ITEM\_TYPE);  
it.putExtra("name","myName");  
it.putExtra(android.provider.Contacts.Intents.Insert.COMPANY,  "organization");  
it.putExtra(android.provider.Contacts.Intents.Insert.EMAIL,"email");  
it.putExtra(android.provider.Contacts.Intents.Insert.PHONE,"homePhone");  
it.putExtra(android.provider.Contacts.Intents.Insert.SECONDARY\_PHONE,  
         "mobilePhone");  
it.putExtra(  android.provider.Contacts.Intents.Insert.TERTIARY\_PHONE,  
          "workPhone");  
it.putExtra(android.provider.Contacts.Intents.Insert.JOB\_TITLE,"title");  
           startActivity(it);  
   
21.调用系统编辑添加联系人（全有效）：  
Intent intent = newIntent(Intent.ACTION\_INSERT\_OR\_EDIT);  
intent.setType(People.CONTENT\_ITEM\_TYPE);  
intent.putExtra(Contacts.Intents.Insert.NAME, "My Name");  
intent.putExtra(Contacts.Intents.Insert.PHONE, "+1234567890");  
intent.putExtra(Contacts.Intents.Insert.PHONE\_TYPE,Contacts.PhonesColumns.TYPE\_MOBILE);  
intent.putExtra(Contacts.Intents.Insert.EMAIL, "com@com.com");  
intent.putExtra(Contacts.Intents.Insert.EMAIL\_TYPE,Contacts.ContactMethodsColumns.TYPE\_WORK);

startActivity(intent);

|  |  |  |  |
| --- | --- | --- | --- |
| String | ADD\_SHORTCUT\_ACTION | 动作：在系统中添加一个快捷方式。. | "android.intent.action.ADD\_SHORTCUT" |
| String | ALL\_APPS\_ACTION | 动作：列举所有可用的应用。 输入：无。 | "android.intent.action.ALL\_APPS" |
| String | ALTERNATIVE\_CATEGORY | 类别：说明 activity 是用户正在浏览的数据的一个可选操作。 | "android.intent.category.ALTERNATIVE" |
| String | ANSWER\_ACTION | 动作：处理拨入的电话。 | "android.intent.action.ANSWER" |
| String | BATTERY\_CHANGED\_ACTION | 广播：充电状态，或者电池的电量发生变化。 | "android.intent.action.BATTERY\_CHANGED" |
| String | BOOT\_COMPLETED\_ACTION | 广播：在系统启动后，这个动作被广播一次（只有一次）。 | "android.intent.action.BOOT\_COMPLETED" |
| String | BROWSABLE\_CATEGORY | 类别：能够被浏览器安全使用的 activities 必须支持这个类别。 | "android.intent.category.BROWSABLE" |
| String | BUG\_REPORT\_ACTION | 动作：显示 activity 报告错误。 | "android.intent.action.BUG\_REPORT" |
| String | CALL\_ACTION | 动作：拨打电话，被呼叫的联系人在数据中指定。 | "android.intent.action.CALL" |
| String | CALL\_FORWARDING\_STATE\_CHANGED\_ACTION | 广播：语音电话的呼叫转移状态已经改变。 | "android.intent.action.CFF" |
| String | CLEAR\_CREDENTIALS\_ACTION | 动作：清除登陆凭证 (credential)。 | "android.intent.action.CLEAR\_CREDENTIALS" |
| String | CONFIGURATION\_CHANGED\_ACTION | 广播：设备的配置信息已经改变，参见 Resources.Configuration. | "android.intent.action.CONFIGURATION\_CHANGED" |
| Creator | CREATOR | 无 | 无 |
| String | DATA\_ACTIVITY\_STATE\_CHANGED\_ACTION | 广播：电话的数据活动(data activity)状态（即收发数据的状态）已经改变。 | "android.intent.action.DATA\_ACTIVITY" |
| String | DATA\_CONNECTION\_STATE\_CHANGED\_ACTION | 广播：电话的数据连接状态已经改变。 | "android.intent.action.DATA\_STATE" |
| String | DATE\_CHANGED\_ACTION | 广播：日期被改变。 | "android.intent.action.DATE\_CHANGED" |
| String | DEFAULT\_ACTION | 动作：和 VIEW\_ACTION 相同，是在数据上执行的标准动作。 | "android.intent.action.VIEW" |
| String | DEFAULT\_CATEGORY | 类别：如果 activity 是对数据执行确省动作（点击, center press）的一个选项  ，需要设置这个类别。 | "android.intent.category.DEFAULT" |
| String | DELETE\_ACTION | 动作：从容器中删除给定的数据。 | "android.intent.action.DELETE" |
| String | DEVELOPMENT\_PREFERENCE\_CATEGORY | 类别：说明 activity 是一个设置面板 (development preference panel). | "android.intent.category.DEVELOPMENT\_PREFERENCE" |
| String | DIAL\_ACTION | 动作：拨打数据中指定的电话号码。 | "android.intent.action.DIAL" |
| String | EDIT\_ACTION | 动作：为制定的数据显示可编辑界面。 | "android.intent.action.EDIT" |
| String | EMBED\_CATEGORY | 类别：能够在上级（父）activity 中运行。 | "android.intent.category.EMBED" |
| String | EMERGENCY\_DIAL\_ACTION | 动作：拨打紧急电话号码。 | "android.intent.action.EMERGENCY\_DIAL" |
| int | FORWARD\_RESULT\_LAUNCH | 启动标记：如果这个标记被设置，而且被一个已经存在的 activity 用来启动  新的 activity，已有 activity 的回复目标 (reply target) 会被转移给新的 activity。 | 16 0x00000010 |
| String | FOTA\_CANCEL\_ACTION | 广播：取消所有被挂起的 (pending) 更新下载。 | "android.server.checkin.FOTA\_CANCEL" |
| String | FOTA\_INSTALL\_ACTION | 广播：更新已经被确认，马上就要开始安装。 | "android.server.checkin.FOTA\_INSTALL" |
| String | FOTA\_READY\_ACTION | 广播：更新已经被下载，可以开始安装。 | "android.server.checkin.FOTA\_READY" |
| String | FOTA\_RESTART\_ACTION | 广播：恢复已经停止的更新下载。 | "android.server.checkin.FOTA\_RESTART" |
| String | FOTA\_UPDATE\_ACTION | 广播：通过 OTA 下载并安装操作系统更新。 | "android.server.checkin.FOTA\_UPDATE" |
| String | FRAMEWORK\_INSTRUMENTATION\_TEST\_CATEGORY | 类别：To be used as code under test for framework instrumentation tests. | "android.intent.category.FRAMEWORK\_INSTRUMENTATION\_TEST" |
| String | GADGET\_CATEGORY | 类别：这个 activity 可以被嵌入宿主 activity (activity that is hosting gadgets)。 | "android.intent.category.GADGET" |
| String | GET\_CONTENT\_ACTION | 动作：让用户选择数据并返回。 | "android.intent.action.GET\_CONTENT" |
| String | HOME\_CATEGORY | 类别：主屏幕 (activity)，设备启动后显示的第一个 activity。 | "android.intent.category.HOME" |
| String | INSERT\_ACTION | 动作：在容器中插入一个空项 (item)。 | "android.intent.action.INSERT" |
| String | INTENT\_EXTRA | 附加数据：和 PICK\_ACTIVITY\_ACTION 一起使用时，说明用户选择的  用来显示的 activity；和 ADD\_SHORTCUT\_ACTION 一起使用的时候，描述要添加的快捷方式。 | "android.intent.extra.INTENT" |
| String | LABEL\_EXTRA | 附加数据：大写字母开头的字符标签，和 ADD\_SHORTCUT\_ACTION 一起使用。 | "android.intent.extra.LABEL" |
| String | LAUNCHER\_CATEGORY | 类别：Activity 应该被显示在顶级的 launcher 中。 | "android.intent.category.LAUNCHER" |
| String | LOGIN\_ACTION | 动作：获取登录凭证。 | "android.intent.action.LOGIN" |
| String | MAIN\_ACTION | 动作：作为主入口点启动，不需要数据。 | "android.intent.action.MAIN" |
| String | MEDIABUTTON\_ACTION | 广播：用户按下了“Media Button”。 | "android.intent.action.MEDIABUTTON" |
| String | MEDIA\_BAD\_REMOVAL\_ACTION | 广播：扩展介质（扩展卡）已经从 SD 卡插槽拔出，但是挂载点 (mount point) 还没解除 (unmount)。 | "android.intent.action.MEDIA\_BAD\_REMOVAL" |
| String | MEDIA\_EJECT\_ACTION | 广播：用户想要移除扩展介质（拔掉扩展卡）。 | "android.intent.action.MEDIA\_EJECT" |
| String | MEDIA\_MOUNTED\_ACTION | 广播：扩展介质被插入，而且已经被挂载。 | "android.intent.action.MEDIA\_MOUNTED" |
| String | MEDIA\_REMOVED\_ACTION | 广播：扩展介质被移除。 | "android.intent.action.MEDIA\_REMOVED" |
| String | MEDIA\_SCANNER\_FINISHED\_ACTION | 广播：已经扫描完介质的一个目录。 | "android.intent.action.MEDIA\_SCANNER\_FINISHED" |
| String | MEDIA\_SCANNER\_STARTED\_ACTION | 广播：开始扫描介质的一个目录。 | "android.intent.action.MEDIA\_SCANNER\_STARTED" |
| String | MEDIA\_SHARED\_ACTION | 广播：扩展介质的挂载被解除 (unmount)，因为它已经作为 USB 大容量存储被共享。 | "android.intent.action.MEDIA\_SHARED" |
| String | MEDIA\_UNMOUNTED\_ACTION | 广播：扩展介质存在，但是还没有被挂载 (mount)。 | "android.intent.action.MEDIA\_UNMOUNTED" |
| String | MESSAGE\_WAITING\_STATE\_CHANGED\_ACTION | 广播：电话的消息等待（语音邮件）状态已经改变。 | "android.intent.action.MWI" |
| int | MULTIPLE\_TASK\_LAUNCH | 启动标记：和 NEW\_TASK\_LAUNCH 联合使用，禁止将已有的任务改变为前景任务 (foreground)。 | 8 0x00000008 |
| String | NETWORK\_TICKLE\_RECEIVED\_ACTION | 广播：设备收到了新的网络 "tickle" 通知。 | "android.intent.action.NETWORK\_TICKLE\_RECEIVED" |
| int | NEW\_TASK\_LAUNCH | 启动标记：设置以后，activity 将成为历史堆栈中的第一个新任务（栈顶）。 | 4 0x00000004 |
| int | NO\_HISTORY\_LAUNCH | 启动标记：设置以后，新的 activity 不会被保存在历史堆栈中。 | 1 0x00000001 |
| String | PACKAGE\_ADDED\_ACTION | 广播：设备上新安装了一个应用程序包。 | "android.intent.action.PACKAGE\_ADDED" |
| String | PACKAGE\_REMOVED\_ACTION | 广播：设备上删除了一个应用程序包。 | "android.intent.action.PACKAGE\_REMOVED" |
| String | PHONE\_STATE\_CHANGED\_ACTION | 广播：电话状态已经改变。 | "android.intent.action.PHONE\_STATE" |
| String | PICK\_ACTION | 动作：从数据中选择一个项目 (item)，将被选中的项目返回。 | "android.intent.action.PICK" |
| String | PICK\_ACTIVITY\_ACTION | 动作：选择一个 activity，返回被选择的 activity 的类（名）。 | "android.intent.action.PICK\_ACTIVITY" |
| String | PREFERENCE\_CATEGORY | 类别：activity是一个设置面板 (preference panel)。 | "android.intent.category.PREFERENCE" |
| String | PROVIDER\_CHANGED\_ACTION | 广播：更新将要（真正）被安装。 | "android.intent.action.PROVIDER\_CHANGED" |
| String | PROVISIONING\_CHECK\_ACTION | 广播：要求 polling of provisioning service 下载最新的设置。 | "android.intent.action.PROVISIONING\_CHECK" |
| String | RUN\_ACTION | 动作：运行数据（指定的应用），无论它（应用）是什么。 | "android.intent.action.RUN" |
| String | SAMPLE\_CODE\_CATEGORY | 类别：To be used as an sample code example (not part of the normal user experience). | "android.intent.category.SAMPLE\_CODE" |
| String | SCREEN\_OFF\_ACTION | 广播：屏幕被关闭。 | "android.intent.action.SCREEN\_OFF" |
| String | SCREEN\_ON\_ACTION | 广播：屏幕已经被打开。 | "android.intent.action.SCREEN\_ON" |
| String | SELECTED\_ALTERNATIVE\_CATEGORY | 类别：对于被用户选中的数据，activity 是它的一个可选操作。 | "android.intent.category.SELECTED\_ALTERNATIVE" |
| String | SENDTO\_ACTION | 动作：向 data 指定的接收者发送一个消息。 | "android.intent.action.SENDTO" |
| String | SERVICE\_STATE\_CHANGED\_ACTION | 广播：电话服务的状态已经改变。 | "android.intent.action.SERVICE\_STATE" |
| String | SETTINGS\_ACTION | 动作：显示系统设置。输入：无。 | "android.intent.action.SETTINGS" |
| String | SIGNAL\_STRENGTH\_CHANGED\_ACTION | 广播：电话的信号强度已经改变。 | "android.intent.action.SIG\_STR" |
| int | SINGLE\_TOP\_LAUNCH | 启动标记：设置以后，如果 activity 已经启动，而且位于历史堆栈的顶端  ，将不再启动（不重新启动） activity。 | 2 0x00000002 |
| String | STATISTICS\_REPORT\_ACTION | 广播：要求 receivers 报告自己的统计信息。 | "android.intent.action.STATISTICS\_REPORT" |
| String | STATISTICS\_STATE\_CHANGED\_ACTION | 广播：统计信息服务的状态已经改变。 | "android.intent.action.STATISTICS\_STATE\_CHANGED" |
| String | SYNC\_ACTION | 动作：执行数据同步。 | "android.intent.action.SYNC" |
| String | TAB\_CATEGORY | 类别：这个 activity 应该在 TabActivity 中作为一个 tab 使用。 | "android.intent.category.TAB" |
| String | TEMPLATE\_EXTRA | 附加数据：新记录的初始化模板。 | "android.intent.extra.TEMPLATE" |
| String | TEST\_CATEGORY | 类别：作为测试目的使用，不是正常的用户体验的一部分。 | "android.intent.category.TEST" |
| String | TIMEZONE\_CHANGED\_ACTION | 广播：时区已经改变。 | "android.intent.action.TIMEZONE\_CHANGED" |
| String | TIME\_CHANGED\_ACTION | 广播：时间已经改变（重新设置）。 | "android.intent.action.TIME\_SET" |
| String | TIME\_TICK\_ACTION | 广播：当前时间已经变化（正常的时间流逝）。 | "android.intent.action.TIME\_TICK" |
| String | UMS\_CONNECTED\_ACTION | 广播：设备进入 USB 大容量存储模式。 | "android.intent.action.UMS\_CONNECTED" |
| String | UMS\_DISCONNECTED\_ACTION | 广播：设备从 USB 大容量存储模式退出。 | "android.intent.action.UMS\_DISCONNECTED" |
| String | UNIT\_TEST\_CATEGORY | 类别：应该被用作单元测试（通过 test harness 运行）。 | "android.intent.category.UNIT\_TEST" |
| String | VIEW\_ACTION | 动作：向用户显示数据。 | "android.intent.action.VIEW" |
| String | WALLPAPER\_CATEGORY | 类别：这个 activity 能过为设备设置墙纸。 | "android.intent.category.WALLPAPER" |
| String | WALLPAPER\_CHANGED\_ACTION | 广播：系统的墙纸已经改变。 | "android.intent.action.WALLPAPER\_CHANGED" |
| String | WALLPAPER\_SETTINGS\_ACTION | 动作：显示选择墙纸的设置界面。输入：无。 | "android.intent.action.WALLPAPER\_SETTINGS" |
| String | WEB\_SEARCH\_ACTION | 动作：执行 web 搜索。 | "android.intent.action.WEB\_SEARCH" |
| String | XMPP\_CONNECTED\_ACTION | 广播：XMPP 连接已经被建立。 | "android.intent.action.XMPP\_CONNECTED" |
| String | XMPP\_DISCONNECTED\_ACTION | 广播：XMPP 连接已经被断开。 | "[**Android**](http://lib.csdn.net/base/android).intent.action.XMPP\_DI |

category元素  
category也只有android:name属性。常见的android:name值为android.intent.category.LAUNCHER(决定应用程序是否显示在程序列表里)  
有关android:name具体有哪些值，可参照以下内容:

说明 activity 是用户正在浏览的数据的一个可选操作  
android.intent.category.ALTERNATIVE  
  
能够被浏览器安全使用的 activities 必须支持这个类别  
android.intent.category.BROWSABLE  
  
如果 activity 是对数据执行确省动作（点击, center press）的一个选项，需要设置这个类别  
android.intent.category.DEFAULT  
   
说明 activity 是一个设置面板 (development preference panel)  
android.intent.category.DEVELOPMENT\_PREFERENCE  
  
能够在上级（父）activity 中运行。   
android.intent.category.EMBED  
  
被用来作为测试仪器测试框架下的代码  
android.intent.category.FRAMEWORK\_INSTRUMENTATION\_TEST  
  
这个 activity 可以被嵌入宿主 activity (activity that is hosting gadgets)  
android.intent.category.GADGET  
  
主屏幕 (activity)，设备启动后显示的第一个 activity  
android.intent.category.HOME  
   
Activity 应该被显示在顶级的 launcher 中  
android.intent.category.LAUNCHER  
  
activity是一个设置面板 (preference panel)  
android.intent.category.PREFERENCE  
  
被用来作为一个示例代码示例（不是普通用户体验的部分）  
android.intent.category.SAMPLE\_CODE  
   
对于被用户选中的数据，activity 是它的一个可选操作  
android.intent.category.SELECTED\_ALTERNATIVE  
  
作为测试目的使用，不是正常的用户体验的一部分  
android.intent.category.TEST  
  
应该被用作单元测试（通过 test harness 运行）  
android.intent.category.UNIT\_TEST  
  
这个 activity 能过为设备设置墙纸  
android.intent.category.WALLPAPE  
  
这个 activity 应该在 TabActivity 中作为一个 tab 使用  
android.intent.category.TAB

data元素

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**                **<data**
2. android:host="string"
3. android:mimeType="string"
4. android:path="string"
5. android:pathPattern="string"
6. android:pathPrefix="string"
7. android:port="string"
8. android:scheme="string" **/></span>**

data元素可以指定一个数据类型（mimeType属性），或者一个URI，或者一个URI和数据类型。一个URI通过四个分开的属性说明：

scheme://host:port/path or pathPrefix or pathPattern

这些属性都是可选的，但是也相互依赖：如果intent filter没有指定scheme属性，URI的其他属性都会忽略。如果intent filter没有指定host属性，那么port属性和所有的路径属性都会忽略。

host属性：

URI的host部分。

mimeType属性：

一个MIME多媒体类型，例如image/jpeg或者audio/mpeg4-generic。子类型可以是星号通配符（\*），表示匹配所有的子类型。

注意：在android框架中MIME类型匹配是区分大小写，和通用的RFC MIME类型不同。所以，你应该使用小写字母说明MIMIE类型。

path属性：

pathPrefix属性：

pathPattern属性：

URI的path部分。path属性指明一个绝对路径用来匹配intent对象中的绝对路径。pathPrefix属性指明一个路径的起始部分用来匹配intent对象中的路径的起始部分。pathPattern属性指明一个绝对路径用来匹配intent对象中的绝对路径，但是它可以包含以下通配符：

一个星号('\*')匹配紧跟着它前面的0个或多个字符。

一个句号加一个星号(".\*")匹配任意0个或多个字符。

port属性：

URI的port部分。只有当filter指定了scheme和host属性后才有意义。

scheme属性：

URI的scheme部分。说明一个URI至少需要一个scheme属性；至少filter指定一个scheme属性，否则其他的URI属性都没有意义。

5、第四层<meta-data>

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**                **<meta-data**
2. android:name="string"
3. android:resource="resource specification"
4. android:value="string" **/></span>**

meta-data元素可以包含在<activity> <activity-alias> <service> <receiver>四个元素中。

android:name

元数据项的名字，为了保证这个名字是唯一的，采用java风格的命名规范，如com.example.project.activity.fred

android:resource

资源的引用，指定给这个项的值是该资源的id。该id可以通过方法Bundle.getInt()来从meta-data中找到。

android:value

指定给这一项的值。

6、第三层<activity-alias>属性

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<span** style="font-size:18px;"**>**        **<activity-alias** android:enabled=["true" | "false"]
2. android:exported=["true" | "false"]
3. android:icon="drawable resource"
4. android:label="string resource"
5. android:name="string"
6. android:permission="string"
7. android:targetActivity="string" **>**
8. . . .
9. **</activity-alias></span>**

为通过targetActivity属性指定activity创建别名。目标activity必须和别名activity在同一个应用中，而且在manifest中必须在别名activity之前

这个别名activity是作为一个独立的个体代表目标activity的。它能够拥有自己的intent filter列表，相比较目标activity本身的intent filter，别名activity的intent filter列表能够决定哪些intent能够启动目标activity以及系统会如何处理别名activity。例如，在别名activity中的intent filter中声明了 "android.intent.action.MAIN"和"android.intent.category.LAUNCHER" ，就是使它在应用启动列表中显示出来，即使在目标activity的intent filter中没有声明这些属性中的任何一个。

android:enable属性：

系统能否通过别名activity实例化目标activity。默认值是'true'。

<application>标签有它自己应用于所有应用组件的enable属性，也包括别名activity。<application>和<activity-alias>必须同时为'true'系统才能够通过别名activity实例化目标activity。任意一个为'false'，别名activity都会失效。

android:exported属性：

是否其他应用的组件可以通过别名activity启动目标activity。

默认值取决于别名activity是否包含intent filter。如果没有任何的filter意味着别名activity只能通过指定别名activity的精确的类名来调用。这暗示这个别名activity只想在本应用内使用(因为其他人不会知道它的名字)，所以这种情况下默认值是'false'。另一种情况，如果至少指定了一种filter，说明它想要外部能够调用它，所以这种情况下的默认值是'true’。

android:icon属性：

显示给用户时的icon。

android:label属性：

显示给用户一个可读的label信息。

android:name属性：

一个唯一的别名。这个名字应该是一个类似全路径的类名。但是，跟目标activity不同，这个别名是随意的，因为它不会引用任何实际的类。

android:permission属性：

客户端必须具有的启动目标activity的权限的名字或者别名activity获取它去做某些事情。如果 startActivity() 或startActivityForResult()方法调用时没有给予指定的权限，目标activity不允许被启动。

android:targetActivity属性：

通过别名activity启动的目标activity的名字。这个名字必须和在manifest中别名activity之前声明的<activity>元素中的目标activity的名字匹配。

7、第三层<service>

**[html]** [view plaincopy](http://blog.csdn.net/ygc87/article/details/7335647)

1. **<service** android:enabled=["true" | "false"]
2. android:exported[="true" | "false"]
3. android:icon="drawable resource"
4. android:label="string resource"
5. android:name="string"
6. android:permission="string"
7. android:process="string"**>**
8. **</service>**

8、第三层<receiver>  
receiver的属性与service一样，这里就不显示了  
BroadcastReceiver：用于发送广播，broadcast是在应用程序之间传输信息的一种机制，而BroadcastReceiver是对发送出来的 Broadcast进行过滤接受并响应的一类组件，具体…（我也不知道了）  
9、第三层<provider>属性  
<provider android:authorities="list"  
android:enabled=["true" | "false"]  
android:exported=["true" | "false"]  
android:grantUriPermissions=["true" | "false"]  
android:icon="drawable resource"  
android:initOrder="integer"  
android:label="string resource"  
android:multiprocess=["true" | "false"]  
android:name="string"  
android:permission="string"  
android:process="string"  
android:readPermission="string"  
android:syncable=["true" | "false"]  
android:writePermission="string">  
<grant-uri-permission/>  
<meta-data/>  
</provider>  
contentProvider(数据存储)  
【1】android:authorities：  
标识这个ContentProvider，调用者可以根据这个标识来找到它  
【2】android:grantUriPermission：  
对某个URI授予的权限  
【3】android:initOrder  
10、第三层<uses-library>  
用户库，可自定义。所有android的包都可以引用  
11、第一层<supports-screens>  
<supports-screens android:smallScreens=["true" | "false"] android:normalScreens=["true" | "false"] android:largeScreens=["true" | "false"] android:anyDensity=["true" | "false"] />  
这是在android1.6以后的新特性，支持多屏幕机制  
各属性含义：这四个属性，是否支持大屏，是否支持中屏，是否支持小屏，是否支持多种不同密度  
12、第二层<uses-configuration />与<uses-feature>性能都差不多  
<uses-configuration android:reqFiveWayNav=["true" | "false"] android:reqHardKeyboard=["true" | "false"]android:reqKeyboardType=["undefined" | "nokeys" | "qwerty" | "twelvekey"]android:reqNavigation=["undefined" | "nonav" | "dpad" | "trackball" | "wheel"]android:reqTouchScreen=["undefined" | "notouch" | "stylus" | "finger"] />  
<uses-feature android:glEsVersion="integer"android:name="string"android:required=["true" | "false"] />  
这两者都是在描述应用所需要的硬件和软件特性，以便防止应用在没有这些特性的设备上安装。  
13、第二层<uses-sdk />  
<uses-sdk android:minSdkVersion="integer"android:targetSdkVersion="integer"android:maxSdkVersion="integer"/>  
描述应用所需的api level，就是版本，目前是android 2.2 = 8，android2.1 = 7，android1.6 = 4，android1.5=3  
在此属性中可以指定支持的最小版本，目标版本以及最大版本  
14、第二层<instrumentation />  
<instrumentation android:functionalTest=["true" | "false"]android:handleProfiling=["true" | "false"]android:icon="drawable resource"android:label="string resource"android:name="string"android:targetPackage="string"/>  
定义一些用于探测和分析应用性能等等相关的类，可以监控程序。在各个应用程序的组件之前instrumentation类被实例化  
android:functionalTest(解释：instrumentation类是否能运行一个功能测试，默认为false)  
15、<permission>、<uses-permission>、<permission-tree />、<permission-group />区别～  
最常用的当属<uses-permission>，当我们需要获取某个权限的时候就必须在我们的manifest文件中声明，此<uses-permission>与<application>同级，具体权限列表请看此处  
通常情况下我们不需要为自己的应用程序声明某个权限，除非你提供了供其他应用程序调用的代码或者数据。这个时候你才需要使用<permission> 这个标签。很显然这个标签可以让我们声明自己的权限。比如：  
<permission android:name="com.teleca.project.MY\_SECURITY" . . . />  
那么在activity中就可以声明该自定义权限了，如：  
<application . . .>  
<activity android:name="XXX" . . . >  
android:permission="com.teleca.project.MY\_SECURITY"> </activity>  
</application>  
当然自己声明的permission也不能随意的使用，还是需要使用<uses-permission>来声明你需要该权限  
<permission-group> 就是声明一个标签，该标签代表了一组permissions，而<permission-tree>是为一组permissions声明了一个namespace。这两个标签可以看之前的系列文章。