此文章已于 9:41:25 2018/3/30 发布到 Song\_UP

清单文件AndroidManifest详解

类别 Android

# 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Intent Receivers,还能指定permissions和instrumentation（安全控制和测试）

## 二、AndroidManifest.xml结构

<?xmlversion="1.0"encoding="utf-8"?>

<manifest>

<application>

<activity>

<intent-filter>

<action/>

<category/>

</intent-filter>

</activity>

<activity-alias>

<intent-filter></intent-filter>

<meta-data/>

</activity-alias>

<service>

<intent-filter></intent-filter>

<meta-data/>

</service>

<receiver>

<intent-filter></intent-filter>

<meta-data/>

</receiver>

<provider>

<grant-uri-permission/>

<meta-data/>

</provider>

<uses-library/>

</application>

<uses-permission/>

<permission/>

<permission-tree/>

<permission-group/>

<instrumentation/>

<uses-sdk/>

<uses-configuration/>

<uses-feature/>

<supports-screens/>

</manifest>

# 三、各个节点的详细介绍

上面就是整个am(androidManifest).xml的结构，下面以外向内开始阐述～～

## 1、第一层(<Manifest>):(属性)

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.woody.test"

android:sharedUserId="string"

android:sharedUserLabel="string resource"

android:versionCode="integer"

android:versionName="string"

android:installLocation=["auto" | "internalOnly" | "preferExternal"] >

</manifest>

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。

B、package

指定本应用内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访问共享数据的。若要共享数据，第一可以采用Share Preference方法，第二种就可以采用sharedUserId了，将不同APK的sharedUserId都设为一样，则这些APK之间就可以互相共享数据了。详见：http://wallage.blog.163.com/blog/static/17389624201011010539408/

D、sharedUserLabel

一个共享的用户名，它只有在设置了sharedUserId属性的前提下才会有意义

E、versionCode

是给设备程序识别版本(升级)用的必须是一个interger值代表app更新过多少次，比如第一版一般为1，之后若要更新版本就设置为2，3等等。。。

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，这个标签声明了每一个应用程序的组件及其属性(如icon,label,permission等)

<application android:allowClearUserData=["true" | "false"]

android:allowTaskReparenting=["true" | "false"]

android:backupAgent="string"

android:debuggable=["true" | "false"]

android:description="string resource"

android:enabled=["true" | "false"]

android:hasCode=["true" | "false"]

android:icon="drawable resource"

android:killAfterRestore=["true" | "false"]

android:label="string resource"

android:manageSpaceActivity="string"

android:name="string"

android:permission="string"

android:persistent=["true" | "false"]

android:process="string"

android:restoreAnyVersion=["true" | "false"]

android:taskAffinity="string"

android:theme="resource or theme" >

</application>

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，反之亦然

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

这也是Android2.2中的一个新特性，设置该APP的备份，属性值应该是一个完整的类名，如com.project.TestCase，此属性并没有默认值，并且类名必须得指定(就是个备份工具，将数据备份到云端的操作)

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,在false的情况下调试该APP，就会报以下错误：

Device XXX requires that applications explicitely declare themselves as debuggable in their manifest.

Application XXX does not have the attribute 'debuggable' set to TRUE in its manifest and cannot be debugged.

E、android:description/android:label

此两个属性都是为许可提供的，均为字符串资源，当用户去看许可列表(android:label)或者某个许可的详细信息(android:description)时，这些字符串资源就可以显示给用户。label应当尽量简短，之需要告知用户该许可是在保护什么功能就行。而description可以用于具体描述获取该许可的程序可以做哪些事情，实际上让用户可以知道如果他们同意程序获取该权限的话，该程序可以做什么。我们通常用两句话来描述许可，第一句描述该许可，第二句警告用户如果批准该权限会可能有什么不好的事情发生

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

J、android:manageSpaceActivity

K、android:name

为应用程序所实现的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",但此类是在package下面的子包的话，就必须声明为全路径或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要新的版本，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

<activity android:allowTaskReparenting=["true" | "false"]

android:alwaysRetainTaskState=["true" | "false"]

android:clearTaskOnLaunch=["true" | "false"]

android:configChanges=["mcc", "mnc", "locale",

"touchscreen", "keyboard", "keyboardHidden",

"navigation", "orientation", "screenLayout",

"fontScale", "uiMode"]

android:enabled=["true" | "false"]

android:excludeFromRecents=["true" | "false"]

android:exported=["true" | "false"]

android:finishOnTaskLaunch=["true" | "false"]

android:icon="drawable resource"

android:label="string resource"

android:launchMode=["multiple" | "singleTop" |

"singleTask" | "singleInstance"]

android:multiprocess=["true" | "false"]

android:name="string"

android:noHistory=["true" | "false"]

android:permission="string"

android:process="string"

android:screenOrientation=["unspecified" | "user" | "behind" |

"landscape" | "portrait" |

"sensor" | "nosensor"]

android:stateNotNeeded=["true" | "false"]

android:taskAffinity="string"

android:theme="resource or theme"

android:windowSoftInputMode=["stateUnspecified",

"stateUnchanged", "stateHidden",

"stateAlwaysHidden", "stateVisible",

"stateAlwaysVisible", "adjustUnspecified",

"adjustResize", "adjustPan"] >

</activity>

(注：有些在application中重复的就不多阐述了)

1、android:alwaysRetainTaskState

是否保留状态不变， 比如切换回home, 再从新打开，activity处于最后的状态。比如一个浏览器拥有很多状态(当打开了多个TAB的时候)，用户并不希望丢失这些状态时，此时可将此属性设置为true

2、android:clearTaskOnLaunch

比如 P 是 activity, Q 是被P 触发的 activity, 然后返回Home, 重新启动 P，是否显示 Q

3、android:configChanges

当配置list发生修改时， 是否调用 onConfigurationChanged() 方法 比如 "locale|navigation|orientation".

这个我用过,主要用来看手机方向改变的. android手机在旋转后,layout会重新布局, 如何做到呢?

正常情况下. 如果手机旋转了.当前Activity后杀掉,然后根据方向重新加载这个Activity. 就会从onCreate开始重新加载.

如果你设置了 这个选项, 当手机旋转后,当前Activity之后调用onConfigurationChanged() 方法. 而不跑onCreate方法等.

4、android:excludeFromRecents

是否可被显示在最近打开的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,这个属性就是王牌。Activity的亲和力将被忽略。该Activity已经被摧毁并非re-parented

6、android:launchMode(Activity加载模式)

在多Activity开发中，有可能是自己应用之间的Activity跳转，或者夹带其他应用的可复用Activity。可能会希望跳转到原来某个Activity实例，而不是产生大量重复的Activity。这需要为Activity配置特定的加载模式，而不是使用默认的加载模式

Activity有四种加载模式：

standard、singleTop、singleTask、singleInstance(其中前两个是一组、后两个是一组)，默认为standard

standard：就是intent将发送给新的实例，所以每次跳转都会生成新的activity。

singleTop：也是发送新的实例，但不同standard的一点是，在请求的Activity正好位于栈顶时(配置成singleTop的Activity)，不会构造新的实例

singleTask：和后面的singleInstance都只创建一个实例，当intent到来，需要创建设置为singleTask的Activity的时候，系统会检查栈里面是否已经有该Activity的实例。如果有直接将intent发送给它。

singleInstance：

首先说明一下task这个概念，Task可以认为是一个栈，可放入多个Activity。比如启动一个应用，那么Android就创建了一个Task，然后启动这个应用的入口Activity，那在它的界面上调用其他的Activity也只是在这个task里面。那如果在多个task中共享一个Activity的话怎么办呢。举个例来说，如果开启一个导游服务类的应用程序，里面有个Activity是开启GOOGLE地图的，当按下home键退回到主菜单又启动GOOGLE地图的应用时，显示的就是刚才的地图，实际上是同一个Activity，实际上这就引入了singleInstance。singleInstance模式就是将该Activity单独放入一个栈中，这样这个栈中只有这一个Activity，不同应用的intent都由这个Activity接收和展示，这样就做到了共享。当然前提是这些应用都没有被销毁，所以刚才是按下的HOME键，如果按下了返回键，则无效

7、android:multiprocess

是否允许多进程，默认是false

具体可看该篇文章：http://www.bangchui.org/simple/?t3181.html

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。Activity不会留下历史痕迹

9、android:screenOrientation

activity显示的模式

默认为unspecified：由系统自动判断显示方向

landscape横屏模式，宽度比高度大

portrait竖屏模式, 高度比宽度大

user模式，用户当前首选的方向

behind模式：和该Activity下面的那个Activity的方向一致(在Activity堆栈中的)

sensor模式：有物理的感应器来决定。如果用户旋转设备这屏幕会横竖屏切换

nosensor模式：忽略物理感应器，这样就不会随着用户旋转设备而更改了

10、android:stateNotNeeded

activity被销毁或者成功重启时是否保存状态

11、android:windowSoftInputMode

activity主窗口与软键盘的交互模式，可以用来避免输入法面板遮挡问题，Android1.5后的一个新特性。

这个属性能影响两件事情：

【A】当有焦点产生时，软键盘是隐藏还是显示

【B】是否减少活动主窗口大小以便腾出空间放软键盘

各值的含义：

【A】stateUnspecified：软键盘的状态并没有指定，系统将选择一个合适的状态或依赖于主题的设置

【B】stateUnchanged：当这个activity出现时，软键盘将一直保持在上一个activity里的状态，无论是隐藏还是显示

【C】stateHidden：用户选择activity时，软键盘总是被隐藏

【D】stateAlwaysHidden：当该Activity主窗口获取焦点时，软键盘也总是被隐藏的

【E】stateVisible：软键盘通常是可见的

【F】stateAlwaysVisible：用户选择activity时，软键盘总是显示的状态

【G】adjustUnspecified：默认设置，通常由系统自行决定是隐藏还是显示

【H】adjustResize：该Activity总是调整屏幕的大小以便留出软键盘的空间

【I】adjustPan：当前窗口的内容将自动移动以便当前焦点从不被键盘覆盖和用户能总是看到输入内容的部分

## 4、第四层(<intent-filter>)

结构图：

<intent-filter android:icon="drawable resource"

android:label="string resource"

android:priority="integer" >

<action />

<category />

<data />

</intent-filter>

intent-filter属性

android:priority(解释：有序广播主要是按照声明的优先级别，如A的级别高于B，那么，广播先传给A，再传给B。优先级别就是用设置priority属性来确定，范围是从-1000～1000，数越大优先级别越高)

Intent filter内会设定的资料包括action,data与category三种。也就是说filter只会与intent里的这三种资料作对比动作

action属性

action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，可参照这个网址：http://hi.baidu.com/linghtway/blog/item/83713cc1c2d053170ff477a7.html

category属性

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，可参照这个网址：http://chroya.javaeye.com/blog/685871

data属性

<data android:host="string"

android:mimeType="string"

android:path="string"

android:pathPattern="string"

android:pathPrefix="string"

android:port="string"

android:scheme="string"/>

【1】每个<data>元素指定一个URI和数据类型（MIME类型）。它有四个属性scheme、host、port、path对应于URI的每个部分：

scheme://host:port/path

scheme的值一般为"http"，host为包名，port为端口号，path为具体地址。如：http://com.test.project:200/folder/etc

其中host和port合起来构成URI的凭据(authority)，如果host没有指定，则port也会被忽略

要让authority有意义，scheme也必须要指定。要让path有意义，scheme+authority也必须要指定

【2】mimeType（指定数据类型），若mimeType为'Image'，则会从content Provider的指定地址中获取image类型的数据。还有'video'啥的，若设置为video/mp4，则表示在指定地址中获取mp4格式的video文件

【3】而pathPattern和PathPrefix主要是为了格式化path所使用的

## 5、第四层<meta-data>

<meta-data android:name="string"

android:resource="resource specification"

android:value="string"/>

这是该元素的基本结构.可以包含在<activity> <activity-alias> <service> <receiver>四个元素中。

android:name（解释：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.woody.project.fried)

android:resource(解释：资源的一个引用，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(解释：指定给这一项的值。可以作为值来指定的数据类型并且组件用来找回那些值的Bundle方法：[getString],[getInt],[getFloat],[getString],[getBoolean])

## 6、第三层<activity-alias>属性

<activity-alias android:enabled=["true" | "false"]

android:exported=["true" | "false"]

android:icon="drawable resource"

android:label="string resource"

android:name="string"

android:permission="string"

android:targetActivity="string">

<intent-filter/>

<meta-data/>

</activity-alias>

<activity-alias>是为activity创建快捷方式的，如下实例：

<activity android:name=".shortcut">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

</intent-filter>

</activity>

<activity-alias android:name=".CreateShortcuts" android:targetActivity=".shortcut" android:label="@string/shortcut">

<intent-filter>

<action android:name="android.intent.action.CREATE\_SHORTCUT" />

<category android:name="android.intent.category.DEFAULT" />

</intent-filter>

</activity-alias>

其中android.targetActivity是指向对应快捷方式的activity,如上述的shortcut(此Activity名)

android:label是指快捷方式的名称，而快捷方式的图标默认是给定的application图标

## 7、第三层<service>

【1】service与activity同级，与activity不同的是，它不能自己启动的，运行在后台的程序，如果我们退出应用时，Service进程并没有结束，它仍然在后台运行。比如听音乐，网络下载数据等，都是由service运行的

【2】service生命周期：Service只继承了onCreate(),onStart(),onDestroy()三个方法，第一次启动Service时，先后调用了onCreate(),onStart()这两个方法，当停止Service时，则执行onDestroy()方法，如果Service已经启动了，当我们再次启动Service时，不会在执行onCreate()方法，而是直接执行onStart()方法

【3】service与activity间的通信

Service后端的数据最终还是要呈现在前端Activity之上的，因为启动Service时，系统会重新开启一个新的进程，这就涉及到不同进程间通信的问题了(AIDL)，Activity与service间的通信主要用IBinder负责。具体可参照：http://zhangyan1158.blog.51cto.com/2487362/491358

【4】

<service android:enabled=["true" | "false"]

android:exported[="true" | "false"]

android:icon="drawable resource"

android:label="string resource"

android:name="string"

android:permission="string"

android:process="string">

</service>

service标签内的属性之前已有描述，在此不重复了～

## 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，具体参照http://kevin2562.javaeye.com/blog/686787

## 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。