android:sharedUserId作用

Jason_Lee155　于 2021-07-05 21:25:46 发布

前言

Android给每个APK进程分配一个单独的空间，manifest中的userid就是对应一个分配的Linux用户ID，并且为它创建一个沙箱，以防止影响其他应用程序（或者被其他应用程序影响）。

通常，不同的APK会具有不同的userId，因此运行时属于不同的进程中，而不同进程中的资源是不共享的（比如只能访问/data/data/自己包名下面的文件），保障了程序运行的稳定。然后在有些时候，我们自己开发了多个APK并且需要他们之间互相共享资源，那么就需要通过设置shareUserId来实现这一目的。

通过Shared User id，拥有同一个User id的多个APK可以配置成运行在同一个进程中，可以互相访问任意数据。也可以配置成运行成不同的进程，同时可以访问其他APK的数据目录下的数据库和文件，就像访问本程序的数据一样（使用IPC机制，不同进程之间，比如AIDL）。

shareUserId的属性的最大作用是什么呢?

前面说了，Android中每个app都对应一个uid，每个uid都有自己的一个沙箱，这是基于安全考虑的，那么说到沙箱，我们会想到的是data/data/XXXX/目录下面的所有数据，因为我们知道这个目录下面的所有数据是一个应用私有的，一般情况下其他应用是没有权限访问的，当然root之后是另外情况，这里就不多说了。这里只看没有root的情况，下面我们在来看一个场景：

A应用和B应用都是一家公司的，现在想在A应用中能够拿到B应用存储的一些值，那么这时候该怎么办呢?

这时候就需要用到了shareUserId属性了，但是这里我们在介绍shareUserId属性前，我们先来看一个简单的例子：

使用两个工程：

ShareUserIdPlugin中的MainActivity.java代码如下：

这里很简单，我们使用SharedPreferences来存储一个密码，注意模式是：Context.MODE_PRIVATE，关于模式后面会介绍。


```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setPassword();
    }

    private void setPassword(){
        SharedPreferences settings = this.getSharedPreferences("pwd", Context.MODE_PRIVATE);
        Log.i("jw", "sp:"+settings);
        Log.i("jw", "ctx:"+this.getApplicationContext());
        SharedPreferences.Editor localEditor = settings.edit();
        localEditor.putString("passwd", "123456");
        localEditor.commit();
    }

}
```

Context提供了几种模式：

1. Context.MODE_PRIVATE：为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容，如果想把新写入的内容追加到原文件中。可以使用Context.MODE_APPEND。
2. Context.MODE_APPEND：模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。
3. Context.MODE_WORLD_READABLE和Context.MODE_WORLD_WRITEABLE用来控制其他应用是否有权限读写该文件。
   MODE_WORLD_READABLE：表示当前文件可以被其他应用读取；MODE_WORLD_WRITEABLE：表示当前文件可以被其他应用写入

其它应用读取该应用目录下此配置文件中passwd肯定是失败的。

> 如果我们想让A应用访问到B应用的数据，我们可以这么做：把B应用创建模式改成可读模式的，那么A应用就可以操作了，那么这就有一个问题，A应用可以访问了，其他应用也可以访问了，这样所有的应用都可以访问B应用的沙盒数据了，太危险了。

所以要用另外的一种方式，那么这时候就要用到shareUserId属性了，我们只需要将B应用创建方式还是private的，然后A应用和B应用公用一个uid即可，我们下面就来修改一下代码，还是前面的那两个工程，修改他们的AndroidManifest.xml，添shareUserId即可。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cn.wjdiankong.shareuseridplugin"
    android:versionCode="1"
    android:versionName="1.0"
    android:sharedUserId=".jw">
```

这时候，我们发现把代码中的模式改成private的，A应用任然可以访问数据了，其实也好理解，他们两个的uid都相同了，A的文件就是B的，B的就是A的了，他们两个没有沙盒的概念了，数据也是透明的了。

所以这里我们就看到了，使用shareUserId可以达到多个应用之间的数据透明性互相访问。

注意：这里有一个点：这里所有的修改的前提是这个应用的AndroidManifest.xml本身就定义了这个shareUserId，然后我们可以反编译看到这个值，把我们自己的shareUserId改成他的就可以了，但是如果这个应用本身没有这个属性，那么这里就没有办法的，为什么呢，如果要添加，那就是另外一条路了，就是逆向，修改AndroidManifest.xml之后，还需要从新打包在验证，但是这时候没必要了，我们也知道有时候回编译还是很艰难的，如果都能回编译了，那都不需要这些工作了，所以这里需要注意的一个前提。

那么修改之后是不是真的可以呢?

答案是肯定不可以的，如果可以的话，那google也太傻比了，其实Android系统中有一个限制，就是说如果多个应用的uid相同的话，那么他们的apk签名必须一致，不然是安装失败的，如下错误:

```
[2016-05-04 10:22:07 - ShareUserIdHost] Installation error: INSTALL_FAILED_SHARED_USER_INCOMPATIBLE
[2016-05-04 10:22:07 - ShareUserIdHost] Please check logcat output for more details.
[2016-05-04 10:22:07 - ShareUserIdHost] Launch canceled!
```


通过上面的分析，我们就知道了，Android中是不允许相同的uid的不同签名的应用。

通过shareduserid来获取系统权限

- (1)在AndroidManifest.xml中添加android:sharedUserId="android.uid.system"
- (2)在Android.mk文件里面添加LOCAL_CERTIFICATE := platform（使用系统签名）
- (3)在源码下面进行mm编译

这样生成的apk能够获取system权限，可以在任意system权限目录下面进行目录或者文件的创建，以及访问其他apk资源等（注意创建的文件(夹)只有创建者(比如system,root除外)拥有可读可写权限-rw-------）。

扩展

系统中所有使用android.uid.system作为共享UID的APK，都会首先在manifest节点中增加android:sharedUserId="android.uid.system"，然后在Android.mk中增加LOCAL_CERTIFICATE := platform。可以参见Settings等；

系统中所有使用android.uid.shared作为共享UID的APK，都会在manifest节点中增加android:sharedUserId="android.uid.shared"，然后在Android.mk中增加LOCAL_CERTIFICATE := shared。可以参见Launcher等；

系统中所有使用android.media作为共享UID的APK，都会在manifest节点中增加android:sharedUserId="android.media"，然后在Android.mk中增加LOCAL_CERTIFICATE := media。可以参见Gallery等。