**PMS 之UID**

1. UID的种类：

UID：当app没有指定sharedUserId，系统为该app分配一个UID。

shareUID：app之间可以共享的uid，需要app来指定。

1. 数据结构：

final ArrayMap<String, SharedUserSetting> mSharedUsers

private final ArrayList<Object> mUserIds

private final SparseArray<Object> mOtherUserIds

mSharedUsers：

存储所有共享的uid。

mUserIds:

存储所有非系统app的uid，包括shareUID。

mOtherUserIds:

存储所有系统app的uid，包括shareUID。

1. 第一次开机：

1.在构造函数中调用mSettings.addSharedUserLPw函数添加系统定义的shareUID，使用用户输入的uid来创建shareUID。

SharedUserSetting addSharedUserLPw(String name, **int uid**, int pkgFlags, int pkgPrivateFlags) {

SharedUserSetting s = **mSharedUsers**.get(name);

if (s != null) {

if (s.userId == uid) {

return s;

}

return null;

}

s = new SharedUserSetting(name, pkgFlags, pkgPrivateFlags);

s.userId = uid;

if (addUserIdLPw(uid, s, name)) {

**mSharedUsers**.put(name, s);

return s;

}

return null;

}

针对shareuid在容器中存储的对象都为SharedUserSetting。

将创建的shareuid的uid添加到mUserIds或者mOtherUserIds。

private boolean addUserIdLPw(int uid, Object obj, Object name) {

if (uid > Process.LAST\_APPLICATION\_UID) {

return false;

}

if (uid >= Process.FIRST\_APPLICATION\_UID) {

int N = mUserIds.size();

final int index = uid - Process.FIRST\_APPLICATION\_UID;

while (index >= N) {

mUserIds.add(null);

N++;

}

if (mUserIds.get(index) != null) {

return false;

}

**mUserIds**.set(index, obj);

} else {

if (mOtherUserIds.get(uid) != null) {

return false;

}

**mOtherUserIds**.put(uid, obj);

}

return true;

}

2.进入扫描：

当app的AndroidManifest中有sharedUserId的tag的时候调用函数mSettings.getSharedUserLPw创建SharedUserSetting对象返回。

SharedUserSetting getSharedUserLPw(String name,

int pkgFlags, int pkgPrivateFlags, boolean create) {

SharedUserSetting s = mSharedUsers.get(name);

if (s == null) {

if (!create) {

return null;

}

s = new SharedUserSetting(name, pkgFlags, pkgPrivateFlags);

s.userId = newUserIdLPw(s);

if (s.userId >= 0) {

mSharedUsers.put(name, s);

}

}

return s;

}

private int newUserIdLPw(Object obj) {

final int N = mUserIds.size();

for (int i = mFirstAvailableUid; i < N; i++) {

if (mUserIds.get(i) == null) {

mUserIds.set(i, obj);

return Process.FIRST\_APPLICATION\_UID + i;

}

}

if (N > (Process.LAST\_APPLICATION\_UID-Process.FIRST\_APPLICATION\_UID)) {

return -1;

}

mUserIds.add(obj);

return Process.FIRST\_APPLICATION\_UID + N;

}

所有扫描时候生成的shareUID都是保存在mUserIds容器中值为10000以后的uid。

后面为package创建PackageSetting对象的时候直接使用SharedUserSetting的userId值为PackageSettin中的appId赋值。

当mainfeist中没有定义sharedUserId，在为package创建PackageSetting对象的时候调用newUserIdLPw直接为PackageSetting对象创建对应的UID保存在mUserIds容器中。

最后调用mSettings.writeLPr函数将sharedUserId或者uid写入到Packages.xml文件中的package TAG中，将mSharedUsers容器的SharedUserSetting写入到shared-user TAG中，包括共享uid的签名和权限。

1. 第二次及之后的开机

在scan之前调用mSettings.readLPw函数来读取Packages.xml文件中package的uid和shared-user的内容。

boolean readLPw(PackageManagerService service, List<UserInfo> users, int sdkVersion,

boolean onlyCore) {

if (tagName.equals("package")) {

**readPackageLPw(parser);**

} else if (tagName.equals("shared-user")) {

**readSharedUserLPw(parser);**

} else if (tagName.equals("preferred-packages")) {

// no longer used.

} else if (tagName.equals("preferred-activities")) {

。。。。。。

}

private void readPackageLPw(XmlPullParser parser) throws XmlPullParserException, IOException {

String idStr = null;

String sharedIdStr = null;

idStr = parser.getAttributeValue(null, "userId");

sharedIdStr = parser.getAttributeValue(null, "sharedUserId");

**int userId = idStr != null ? Integer.parseInt(idStr) : 0;**

**if (userId > 0) {**

packageSetting = addPackageLPw(name.intern(), realName, new File(codePathStr),

new File(resourcePathStr), legacyNativeLibraryPathStr, primaryCpuAbiString,

secondaryCpuAbiString, cpuAbiOverrideString, userId, versionCode, pkgFlags,

pkgPrivateFlags);

if (packageSetting == null) {

} else {

packageSetting.setTimeStamp(timeStamp);

packageSetting.firstInstallTime = firstInstallTime;

packageSetting.lastUpdateTime = lastUpdateTime;

}

} else if (sharedIdStr != null) {

userId = sharedIdStr != null ? Integer.parseInt(sharedIdStr) : 0;

**if (userId > 0) {**

packageSetting = new PendingPackage(name.intern(), realName, new File(

codePathStr), new File(resourcePathStr), legacyNativeLibraryPathStr,

primaryCpuAbiString, secondaryCpuAbiString, cpuAbiOverrideString,

**userId**, versionCode, pkgFlags, pkgPrivateFlags);

packageSetting.setTimeStamp(timeStamp);

packageSetting.firstInstallTime = firstInstallTime;

packageSetting.lastUpdateTime = lastUpdateTime;

**mPendingPackages**.add((PendingPackage) packageSetting);

} else {

}

} else {

}  
 }

如果从Packages.xml文件的package TAG读出的uid是为userId TAG中的值，通过addPackageLPw函数使用解析出来的UID来创建package的PackageSetting对象,调用函数addUserIdLPw将创建的PackageSetting对象添加到mUserIds或者mOtherUserIds容器中。

如果从Packages.xml文件的package TAG读出的uid是为sharedUserIdTAG中的值，创建一个PendingPackage对象保存在mPendingPackages容器，当解析xml文件后会对该容器中的PendingPackage进行处理。

解析shared-user的TAG调用addSharedUserLPw创建SharedUserSetting对象，保存在mSharedUsers和mUserIds或者mOtherUserIds容器中。

final int N = mPendingPackages.size();

for (int i = 0; i < N; i++) {

final PendingPackage pp = mPendingPackages.get(i);

Object idObj = getUserIdLPr(pp.sharedId);

if (idObj != null && idObj instanceof SharedUserSetting) {

PackageSetting p = getPackageLPw(pp.name, null, pp.realName,

(SharedUserSetting) idObj, pp.codePath, pp.resourcePath,

pp.legacyNativeLibraryPathString, pp.primaryCpuAbiString,

pp.secondaryCpuAbiString, pp.versionCode, pp.pkgFlags, pp.pkgPrivateFlags,

null, true /\* add \*/, false /\* allowInstall \*/);

if (p == null) {

continue;

}

p.copyFrom(pp);

} else if (idObj != null) {

PackageManagerService.reportSettingsProblem(Log.ERROR, msg);

} else {

PackageManagerService.reportSettingsProblem(Log.ERROR, msg);

}

}

mPendingPackages.clear();

当解析xml文件完成以后对mPendingPackages容器中的使用sharedUserId 的packagesetting进行处理，通过getUserIdLPr函数获取uid对应的SharedUserSetting，调用getPackageLPw创建package的PackageSetting对象，使用从xml文件解析出来SharedUserSetting的userId为

PackageSetting的appId赋值。

之后扫描的过程中也不会去创建SharedUserSetting和对应的uid和PackageSetting对应的uid。

1. 安装

4.卸载