源码目录/package/apps/PackageInstaller，其主要的功能就是实现应用的安装和卸载功能。

# 应用安装

### Manifest.XML

|  |
| --- |
| <**manifest   package="com.android.packageinstaller"**>  <**uses-permission android:name="android.permission.INSTALL\_PACKAGES"** />   <**activity android:name=".PackageInstallerActivity"**>  <**intent-filter**>  <**action android:name="android.intent.action.VIEW"** />  <**action android:name="android.intent.action.INSTALL\_PACKAGE"** />  <**category android:name="android.intent.category.DEFAULT"** />  <**data android:scheme="file"** />  <**data android:mimeType="application/vnd.android.package-archive"** />  </**intent-filter**>  <**intent-filter**>  <**action android:name="android.intent.action.INSTALL\_PACKAGE"** />  <**category android:name="android.intent.category.DEFAULT"** />  <**data android:scheme="file"** />  <**data android:scheme="package"** />  </**intent-filter**> </**activity**>  </**manifest**> |

可以总结一些信息：

intent-filter包含android.intent.action.MAIN的Activity Action的Activity会在系统程序列表中列出相应的应用图标。PackageInstaller中的Activity并**没有注册MAIN**的Action，图标**不会列在应用程序列表中**

PackageInstallerActivity采用隐示调用，包含了两个Intent Filter，也就是支持两种方式开启该PackageInstallerActivity；

|  |
| --- |
| Intent intent = new Intent(Intent.ACTION\_INSTALL\_PACKAGE);  intent.setDataAndType(Uri.fromFile(new File("/sdcard/qq.apk")),"application/vnd.android.package-archive");  startActivity(intent); |
| Intent intent = new Intent(Intent.ACTION\_INSTALL\_PACKAGE);  intent.setData(Uri.fromFile(new File("/sdcard/qq.apk")));  startActivity(intent); |
| Adb shell am start -a android.intent.action.VIEW -t application/vnd.android.package-archive -c android.intent.category.DEFAULT -d file:///mnt/sdcard/t.apk |



### 在PackageInstallerActivity.onCreate中

1. 从Intent对象获取Package URL、Scheme信息

2. 校验Scheme，根据Scheme不同的值进行逻辑处理。Scheme为file或者package

|  |
| --- |
| *//scheme只有两个值：file或package*  **if** (scheme != **null** && !**"file"**.equals(scheme) && !**"package"**.equals(scheme)) {  setPmResult(PackageManager.INSTALL\_FAILED\_INVALID\_URI);  finish();  **return**;  }  *// 未知源安装检测*  **if** ((requestFromUnknownSource) && (!isInstallingUnknownAppsAllowed())) {  *//ask user to enable setting first*  showDialogInner(***DLG\_UNKNOWN\_APPS***);  **return**;  } |

### InstallAppProgress

|  |
| --- |
| **if** (**"package"**.equals(**mPackageURI**.getScheme())) {  **try** {  pm.installExistingPackage(**mAppInfo**.**packageName**);  observer.packageInstalled(**mAppInfo**.**packageName**,  PackageManager.INSTALL\_SUCCEEDED);  } **catch** (PackageManager.NameNotFoundException e) {  observer.packageInstalled(**mAppInfo**.**packageName**,  PackageManager.INSTALL\_FAILED\_INVALID\_APK);  }  } **else** {  pm.installPackageWithVerificationAndEncryption(**mPackageURI**, observer, installFlags,  installerPackageName, verificationParams, **null**);  } |

pm.installExistingPackage()方法更新应用，反之如果scheme为file，则调用pm.installPackageWithVerificationAndEncryption()方法进行应用的安装。上面说到的两个方法都是PakcageManager中的方法，**两个方法都是静默安装**，在安装的时候不会出现任何的提示。但是由于上述两个方法在PackageManager中是被注释为@hide的，所以，普通的Android应用中无法调用和访问。静默安装是一个异步过程，所以，无论安装成功或者安装失败，都会向用户弹出结果，所以我们在调用方法中看到有个行参是observer，为PackageInstallObserver实例。PackageInstallObserver主要就是处理安装的结果，其中定义了Handler变量mHandler来进行UI操作，用于实现对用户提示。

总结：从技术上来说，**实现静默安装Android应用就是调用PackageManager.installPackageWithVerificationAndEncryption方法即可**。该方法的参数较多，最重要的是需要一个异步安装结果监听器用于处理安装结果。此监听器必须是**IPackageInstallObserver.Stub**的子类。但**是只有安装结果，并没有具体的进度。**

最后还要说明的是：PackageInstaller安装应用程序实现安装应用的功能是需要申请权限：android.permission.INSTALL\_PACKAGES.该权限属于**系统级别的权限**，在普通的应用中无法使用



# 权限管理

## 配置介绍

Dsf

<activity android:name=".permission.ui.GrantPermissionsActivity"

android:configChanges="orientation|keyboardHidden|screenSize"

android:excludeFromRecents="true"

android:theme="@style/GrantPermissions">

<intent-filter>

<action android:name="android.content.pm.action.REQUEST\_PERMISSIONS" />

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

</intent-filter>

</activity>

<activity android:name=".permission.ui.ManagePermissionsActivity"

android:configChanges="orientation|keyboardHidden|screenSize"

android:excludeFromRecents="true"

android:label="@string/app\_permissions"

android:theme="@style/Settings"

android:permission="android.permission.GRANT\_RUNTIME\_PERMISSIONS">

<intent-filter>

<action android:name="android.intent.action.MANAGE\_PERMISSIONS" />

<action android:name="android.intent.action.MANAGE\_APP\_PERMISSIONS" />

<action android:name="android.intent.action.MANAGE\_PERMISSION\_APPS" />

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

</intent-filter>

</activity>

<activity android:name=".permission.ui.ReviewPermissionsActivity"

android:excludeFromRecents="true"

android:theme="@style/Settings.NoActionBar"

android:permission="android.permission.GRANT\_RUNTIME\_PERMISSIONS">

<intent-filter>

<action android:name="android.intent.action.REVIEW\_PERMISSIONS" />

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

</intent-filter>

</activity>

<activity android:name=".permission.ui.OverlayWarningDialog"

android:excludeFromRecents="true"

android:theme="@android:style/Theme.DeviceDefault.Light.Dialog.NoActionBar" />

<receiver android:name=".permission.model.PermissionStatusReceiver"

android:permission="android.permission.GRANT\_RUNTIME\_PERMISSIONS">

<intent-filter>

<action android:name="android.intent.action.GET\_PERMISSIONS\_COUNT" />

<action android:name="android.intent.action.GET\_PERMISSIONS\_PACKAGES" />

</intent-filter>

</receiver>

## 系统设置页面授权

手动授权的时候，怎么去到对应包名的那个设置的界面呢？

Intent intent = new Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS);

Uri uri = Uri.fromParts("package", context.getPackageName(), null);

intent.setData(uri);

// Start for result

startForResult(activityOrFragment, intent, settingsRequestCode);

其中 Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS，在【settings】中的InstalledAppDetails，作为一个代理，重定位到**PackageInstaller的**ManagePermissionsActivity页面中

### ManagePermissionsActivity

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

if (savedInstanceState != null) {

return;

}

// 这里定义一个 Fragment 对象

Fragment fragment;

// 接收intent的action

String action = getIntent().getAction();

// 根据action进行fragment的实例化

switch (action) {

case Intent.ACTION\_MANAGE\_PERMISSIONS: {

fragment = ManagePermissionsFragment.newInstance();

} break;

// 此处为我们要找的逻辑

case Intent.ACTION\_MANAGE\_APP\_PERMISSIONS: {

String packageName = getIntent().getStringExtra(Intent.EXTRA\_PACKAGE\_NAME);

if (packageName == null) {

Log.i(LOG\_TAG, "Missing mandatory argument EXTRA\_PACKAGE\_NAME");

finish();

return;

}

fragment = AppPermissionsFragment.newInstance(packageName);

} break;

case Intent.ACTION\_MANAGE\_PERMISSION\_APPS: {

String permissionName = getIntent().getStringExtra(Intent.EXTRA\_PERMISSION\_NAME);

if (permissionName == null) {

Log.i(LOG\_TAG, "Missing mandatory argument EXTRA\_PERMISSION\_NAME");

finish();

return;

}

fragment = PermissionAppsFragment.newInstance(permissionName);

} break;

default: {

Log.w(LOG\_TAG, "Unrecognized action " + action);

finish();

return;

}

}

getFragmentManager().beginTransaction().replace(android.R.id.content, fragment).commit();

}

### AppPermissionsFragment.newInstance

**[java]** [view plain](https://blog.csdn.net/jspping/article/details/54288639) [copy](https://blog.csdn.net/jspping/article/details/54288639)

1. // AppPermissions 构造函数
2. public AppPermissions(Context context, PackageInfo packageInfo, String[] permissions,
3. boolean sortGroups, Runnable onErrorCallback) {
4. mContext = context;
5. mPackageInfo = packageInfo;
6. mFilterPermissions = permissions;
7. // 加载 app 的lable 也就是你的app在桌面的名字
8. mAppLabel = loadEllipsizedAppLabel(context, packageInfo);
9. mSortGroups = sortGroups;
10. mOnErrorCallback = onErrorCallback;
11. // 加载权限组
12. loadPermissionGroups();
13. }
15. // 加载权限组
16. private void loadPermissionGroups() {
17. // 当前对象初次调用实例化，清空 ArrayList<AppPermissionGroup> mGroups 对象的数据
18. mGroups.clear();
20. // 当 PackageInfo 为 null 不再往下执行
21. if (mPackageInfo.requestedPermissions == null) {
22. return;
23. }
25. // mFilterPermissions 在本地对象第一次实例化，是默认为 null的，因为参数来自 AppPermissions 且 null，所以会走 else 的逻辑
26. if (mFilterPermissions != null) {
27. for (String filterPermission : mFilterPermissions) {
28. for (String requestedPerm : mPackageInfo.requestedPermissions) {
29. if (!filterPermission.equals(requestedPerm)) {
30. continue;
31. }
33. if (hasGroupForPermission(requestedPerm)) {
34. break;
35. }
37. AppPermissionGroup group = AppPermissionGroup.create(mContext,
38. mPackageInfo, requestedPerm);
39. if (group == null) {
40. break;
41. }
43. mGroups.add(group);
44. break;
45. }
46. }
47. } else {
48. // 遍历 app 的请求权限列表
49. for (String requestedPerm : mPackageInfo.requestedPermissions) {
50. // 如果 ArrayList<AppPermissionGroup> mGroups 对象已存在该权限，不添加，继续下一个
51. if (hasGroupForPermission(requestedPerm)) {
52. continue;
53. }
55. AppPermissionGroup group = AppPermissionGroup.create(mContext,
56. mPackageInfo, requestedPerm);
57. // 若 group 为 null ，继续下一个
58. if (group == null) {
59. continue;
60. }
62. mGroups.add(group);
63. }
64. }
66. // 默认权限排序
67. if (mSortGroups) {
68. Collections.sort(mGroups);
69. }
71. // 清空权限组的 键值名称，如：电话或存储空间
72. mNameToGroupMap.clear();
73. // 变量新的app所有的请求权限组，新增至 mNameToGroupMap
74. for (AppPermissionGroup group : mGroups) {
75. mNameToGroupMap.put(group.getName(), group);
76. }
77. }
79. private boolean hasGroupForPermission(String permission) {
80. for (AppPermissionGroup group : mGroups) {
81. if (group.hasPermission(permission)) {
82. return true;
83. }
84. }
85. return false;
86. }<

### AppPermissions构造函数

Gsdf

**[java]** [view plain](https://blog.csdn.net/jspping/article/details/54288639) [copy](https://blog.csdn.net/jspping/article/details/54288639)

1. **public** **static** AppPermissionsFragment newInstance(String packageName) {
2. **return** setPackageName(**new** AppPermissionsFragment(), packageName);
3. }
5. **private** **static** <T **extends** Fragment> T setPackageName(T fragment, String packageName) {
6. Bundle arguments = **new** Bundle();
7. arguments.putString(Intent.EXTRA\_PACKAGE\_NAME, packageName);
8. // 将包名以 Bundle String 键值对存储，并调用setArguments
9. fragment.setArguments(arguments);
10. **return** fragment;
11. }
13. @Override
14. **public** **void** onCreate(Bundle savedInstanceState) {
15. **super**.onCreate(savedInstanceState);
16. setLoading(**true** /\* loading \*/, **false** /\* animate \*/);
17. setHasOptionsMenu(**true**);
18. **final** ActionBar ab = getActivity().getActionBar();
19. **if** (ab != **null**) {
20. ab.setDisplayHomeAsUpEnabled(**true**);
21. }
23. // getArguments() 从setArguments 函数取出 Bundle String 的包名键值对
24. String packageName = getArguments().getString(Intent.EXTRA\_PACKAGE\_NAME);
25. Activity activity = getActivity();
26. // 根据包名获取 PackageInfo
27. PackageInfo packageInfo = getPackageInfo(activity, packageName);
28. **if** (packageInfo == **null**) {
29. Toast.makeText(activity, R.string.app\_not\_found\_dlg\_title, Toast.LENGTH\_LONG).show();
30. activity.finish();
31. **return**;
32. }
34. // 传入 PackageInfo 进行权限相关操作
35. mAppPermissions = **new** **AppPermissions**(activity, packageInfo, **null**, **true**, **new** Runnable() {
36. @Override
37. **public** **void** run() {
38. getActivity().finish();
39. }
40. });
41. // 初始化GUI
42. loadPreferences();
43. }<

### AppPermissionsFragment 的UI初始化

Sdf

@Override

public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

if (mAppPermissions != null) {

bindUi(this, mAppPermissions.getPackageInfo());

}

}

private static void bindUi(SettingsWithHeader fragment, PackageInfo packageInfo) {

Activity activity = fragment.getActivity();

PackageManager pm = activity.getPackageManager();

ApplicationInfo appInfo = packageInfo.applicationInfo;

Intent infoIntent = null;

if (!activity.getIntent().getBooleanExtra(EXTRA\_HIDE\_INFO\_BUTTON, false)) {

infoIntent = new Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)

.setData(Uri.fromParts("package", packageInfo.packageName, null));

}

// 图标

Drawable icon = appInfo.loadIcon(pm);

// app label

CharSequence label = appInfo.loadLabel(pm);

fragment.setHeader(icon, label, infoIntent);

ActionBar ab = activity.getActionBar();

if (ab != null) {

ab.setTitle(R.string.app\_permissions);

}

ViewGroup rootView = (ViewGroup) fragment.getView();

ImageView iconView = (ImageView) rootView.findViewById(R.id.lb\_icon);

if (iconView != null) {

iconView.setImageDrawable(icon);

}

TextView titleView = (TextView) rootView.findViewById(R.id.lb\_title);

if (titleView != null) {

titleView.setText(R.string.app\_permissions);

}

TextView breadcrumbView = (TextView) rootView.findViewById(R.id.lb\_breadcrumb);

if (breadcrumbView != null) {

breadcrumbView.setText(label);

}

}

### 权限改变，用户手点的实现

Df

@Override

public boolean onPreferenceChange(final Preference preference, Object newValue) {

// 当 Preference change 也就是权限被用户手动改变时回调的对应的 Preference

String groupName = preference.getKey();

// 将 key 去AppPermissionGroup中取出

final AppPermissionGroup group = mAppPermissions.getPermissionGroup(groupName);

// 当group为null默认关

if (group == null) {

return false;

}

// 取出宿主 OverlayTouchActivity

OverlayTouchActivity activity = (OverlayTouchActivity) getActivity();

// 监听当前权限界面是否被覆盖和叠加

if (activity.isObscuredTouch()) {

// 弹出一个警告框，检测到屏幕叠加层，会弹出此框，并提示 要更改此权限设置，您必须首先在“设置”>“应用”中关闭屏幕叠加层"

activity.showOverlayDialog();

return false;

}

// 添加一个新的权限 group 到Togglelist中

addToggledGroup(group);

// 检测是否添加了Manifest.permission\_group.LOCATION 权限组并且该应用为系统应用，包名 equals(ILocationManager.Stub.asInterface(

ServiceManager.getService(Context.LOCATION\_SERVICE)).getNetworkProviderPackage())

if (LocationUtils.isLocationGroupAndProvider(group.getName(), group.getApp().packageName)) {

// 弹出一个警告，提示：是此设备的一个位置信息服务提供程序。您可以在位置信息设置中修改位置信息使用权

LocationUtils.showLocationDialog(getContext(), mAppPermissions.getAppLabel());

return false;

}

// Object newValue == true,用户赋予权限

if (newValue == Boolean.TRUE) {

/\*\*

\*

\* 此处由AppPermissionGroup间接调用抽象类PackageManager的grantRuntimePermissions函数后面会细说

\*

\*/

group.grantRuntimePermissions(false);

} else {

// grantedByDefault为false 提示：此应用专为旧版 Android 打造。拒绝权限可能会导致其无法正常运行

// grantedByDefault为true 提示：如果您拒绝此权限，您设备的基本功能可能会无法正常使用

final boolean grantedByDefault = group.hasGrantedByDefaultPermission();

// grantedByDefault || 当前app targetSdkVersion > Build.VERSION\_CODES.LOLLIPOP\_MR1 && 是否为旧版(即SDK<23)，默认 false

if (grantedByDefault || (!group.hasRuntimePermission() && !mHasConfirmedRevoke)) {

new AlertDialog.Builder(getContext())

.setMessage(grantedByDefault ? R.string.system\_warning

: R.string.old\_sdk\_deny\_warning)

.setNegativeButton(R.string.cancel, null)

.setPositiveButton(R.string.grant\_dialog\_button\_deny,

new OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {

// 更新权限开关

((SwitchPreference) preference).setChecked(false);

// 关闭权限

/\*\*

\*

\* 此处由AppPermissionGroup间接调用抽象类PackageManager的revokeRuntimePermissions函数后面会细说

\*

\*/

group.revokeRuntimePermissions(false);

// 倘若是旧版，mHasConfirmedRevoke = true

if (!grantedByDefault) {

mHasConfirmedRevoke = true;

}

}

})

.show();

return false;

} else {

group.revokeRuntimePermissions(false);

}

}

return true;

### AppPermissionGroup grantRuntimePermissions() revokeRuntimePermissions() 函数具体实现

Asdf

public boolean grantRuntimePermissions(boolean fixedByTheUser) {

final boolean isSharedUser = mPackageInfo.sharedUserId != null;

final int uid = mPackageInfo.applicationInfo.uid;

// We toggle permissions only to apps that support runtime

// permissions, otherwise we toggle the app op corresponding

// to the permission if the permission is granted to the app.

for (Permission permission : mPermissions.values()) {

// 具备运行时权限的 app

if (mAppSupportsRuntimePermissions) {

// Do not touch permissions fixed by the system.

if (permission.isSystemFixed()) {

return false;

}

// Ensure the permission app op enabled before the permission grant.

if (permission.hasAppOp() && !permission.isAppOpAllowed()) {

permission.setAppOpAllowed(true);

mAppOps.setUidMode(permission.getAppOp(), uid, AppOpsManager.MODE\_ALLOWED);

}

// Grant the permission if needed.

// 如果没有授权，授予权限

// mPackageManager 来自 Context.getPackageManager()，而 Context 是一个抽象类，具体方法实现由其子类 ContextImpl 实现

// ContextImpl 下 getPackageManager() 函数调用 ActivityThread.getPackageManager() 函数得到 IPackageManager 实例，倘若这个实例不为 null ，函数返回new ApplicationPackageManager(this, pm)

// ApplicationPackageManager 对象，但是ApplicationPackageManager也继承自 PackageManager 且间接调用 IPackageManager ，而 IPackageManager 最终是以aidl的形式通过 IBinder 传递，由 PackageManagerService 继承

// 抽象方法在该类实现

if (!permission.isGranted()) {

permission.setGranted(true);

mPackageManager.grantRuntimePermission(mPackageInfo.packageName,

permission.getName(), mUserHandle);

}

// Update the permission flags.

if (!fixedByTheUser) {

// Now the apps can ask for the permission as the user

// no longer has it fixed in a denied state.

if (permission.isUserFixed() || permission.isUserSet()) {

permission.setUserFixed(false);

permission.setUserSet(true);

mPackageManager.updatePermissionFlags(permission.getName(),

mPackageInfo.packageName,

PackageManager.FLAG\_PERMISSION\_USER\_FIXED

| PackageManager.FLAG\_PERMISSION\_USER\_SET,

0, mUserHandle);

}

}

} else {

// Legacy apps cannot have a not granted permission but just in case.

// Also if the permissions has no corresponding app op, then it is a

// third-party one and we do not offer toggling of such permissions.

if (!permission.isGranted() || !permission.hasAppOp()) {

continue;

}

if (!permission.isAppOpAllowed()) {

permission.setAppOpAllowed(true);

// It this is a shared user we want to enable the app op for all

// packages in the shared user to match the behavior of this

// shared user having a runtime permission.

if (isSharedUser) {

// Enable the app op.

String[] packageNames = mPackageManager.getPackagesForUid(uid);

for (String packageName : packageNames) {

mAppOps.setUidMode(permission.getAppOp(), uid,

AppOpsManager.MODE\_ALLOWED);

}

} else {

// Enable the app op.

mAppOps.setUidMode(permission.getAppOp(), uid, AppOpsManager.MODE\_ALLOWED);

}

// Mark that the permission should not be be granted on upgrade

// when the app begins supporting runtime permissions.

if (permission.shouldRevokeOnUpgrade()) {

permission.setRevokeOnUpgrade(false);

mPackageManager.updatePermissionFlags(permission.getName(),

mPackageInfo.packageName,

PackageManager.FLAG\_PERMISSION\_REVOKE\_ON\_UPGRADE,

0, mUserHandle);

}

// Legacy apps do not know that they have to retry access to a

// resource due to changes in runtime permissions (app ops in this

// case). Therefore, we restart them on app op change, so they

// can pick up the change.

mActivityManager.killUid(uid, KILL\_REASON\_APP\_OP\_CHANGE);

}

}

}

return true;

}

public boolean revokeRuntimePermissions(boolean fixedByTheUser) {

final boolean isSharedUser = mPackageInfo.sharedUserId != null;

final int uid = mPackageInfo.applicationInfo.uid;

// We toggle permissions only to apps that support runtime

// permissions, otherwise we toggle the app op corresponding

// to the permission if the permission is granted to the app.

for (Permission permission : mPermissions.values()) {

// 具备运行时权限的 app

if (mAppSupportsRuntimePermissions) {

// Do not touch permissions fixed by the system.

if (permission.isSystemFixed()) {

return false;

}

// Revoke the permission if needed.

// 如果有授权，取消授予权限

// mPackageManager 来自 Context.getPackageManager()，而 Context 是一个抽象类，具体方法实现由其子类 ContextImpl 实现

// ContextImpl 下 getPackageManager() 函数调用 ActivityThread.getPackageManager() 函数得到 IPackageManager 实例，倘若这个实例不为 null ，函数返回new ApplicationPackageManager(this, pm)

// ApplicationPackageManager 对象，但是ApplicationPackageManager也继承自 PackageManager 且间接调用 IPackageManager ，而 IPackageManager 最终是以aidl的形式通过 IBinder 传递，由 PackageManagerService 继承

// 抽象方法在该类实现

if (permission.isGranted()) {

permission.setGranted(false);

mPackageManager.revokeRuntimePermission(mPackageInfo.packageName,

permission.getName(), mUserHandle);

}

// Update the permission flags.

if (fixedByTheUser) {

// Take a note that the user fixed the permission.

if (permission.isUserSet() || !permission.isUserFixed()) {

permission.setUserSet(false);

permission.setUserFixed(true);

mPackageManager.updatePermissionFlags(permission.getName(),

mPackageInfo.packageName,

PackageManager.FLAG\_PERMISSION\_USER\_SET

| PackageManager.FLAG\_PERMISSION\_USER\_FIXED,

PackageManager.FLAG\_PERMISSION\_USER\_FIXED,

mUserHandle);

}

} else {

if (!permission.isUserSet()) {

permission.setUserSet(true);

// Take a note that the user already chose once.

mPackageManager.updatePermissionFlags(permission.getName(),

mPackageInfo.packageName,

PackageManager.FLAG\_PERMISSION\_USER\_SET,

PackageManager.FLAG\_PERMISSION\_USER\_SET,

mUserHandle);

}

}

} else {

// Legacy apps cannot have a non-granted permission but just in case.

// Also if the permission has no corresponding app op, then it is a

// third-party one and we do not offer toggling of such permissions.

if (!permission.isGranted() || !permission.hasAppOp()) {

continue;

}

if (permission.isAppOpAllowed()) {

permission.setAppOpAllowed(false);

// It this is a shared user we want to enable the app op for all

// packages the the shared user to match the behavior of this

// shared user having a runtime permission.

if (isSharedUser) {

String[] packageNames = mPackageManager.getPackagesForUid(uid);

for (String packageName : packageNames) {

// Disable the app op.

mAppOps.setUidMode(permission.getAppOp(), uid,

AppOpsManager.MODE\_IGNORED);

}

} else {

// Disable the app op.

mAppOps.setUidMode(permission.getAppOp(), uid, AppOpsManager.MODE\_IGNORED);

}

// Mark that the permission should not be granted on upgrade

// when the app begins supporting runtime permissions.

if (!permission.shouldRevokeOnUpgrade()) {

permission.setRevokeOnUpgrade(true);

mPackageManager.updatePermissionFlags(permission.getName(),

mPackageInfo.packageName,

PackageManager.FLAG\_PERMISSION\_REVOKE\_ON\_UPGRADE,

PackageManager.FLAG\_PERMISSION\_REVOKE\_ON\_UPGRADE,

mUserHandle);

}

// Disabling an app op may put the app in a situation in which it

// has a handle to state it shouldn't have, so we have to kill the

// app. This matches the revoke runtime permission behavior.

mActivityManager.killUid(uid, KILL\_REASON\_APP\_OP\_CHANGE);

}

}

}

return true;

### PackageManagerService.grantRuntimePermissions