1. topic.

# TODO

SELinux

06-01 00:46:47.950 222-222/? E/SELinux: avc: denied { find } for service=v1 pid=7448 uid=10049 scontext=u:r:platform\_app:s0:c512,c768 tcontext=u:object\_r:default\_android\_service:s0 tclass=service\_manager permissive=1

# UserHandle

Fgh

## getAppId

/\*\*

\* @hide Range of uids allocated for a user.

\*/

public static final int PER\_USER\_RANGE = 100000;

@TestApi

public static @AppIdInt int getAppId(int uid) {

return uid % PER\_USER\_RANGE;

}

## isSameApp

public static boolean isSameApp(int uid1, int uid2) {

return getAppId(uid1) == getAppId(uid2);

}

## getUserId

/\*\*

\* Returns the user id for a given uid.

\* @hide

\*/

public static @UserIdInt int getUserId(int uid) {

if (MU\_ENABLED) {

return uid / PER\_USER\_RANGE;

} else {

return UserHandle.USER\_SYSTEM;

}

}

## getCallingUserId

/\*\* @hide \*/

public static @UserIdInt int getCallingUserId() {

return getUserId(Binder.getCallingUid());

}

## isIsolated

孤立的；隔离的；分离的

/\*\* @hide \*/

public static boolean isIsolated(int uid) {

if (uid > 0) {

final int appId = getAppId(uid);

return appId >= Process.FIRST\_ISOLATED\_UID && appId <= Process.LAST\_ISOLATED\_UID;

} else {

return false;

}

}

/\*\* @hide \*/

public static boolean isApp(int uid) {

if (uid > 0) {

final int appId = getAppId(uid);

return appId >= Process.FIRST\_APPLICATION\_UID && appId <= Process.LAST\_APPLICATION\_UID;

} else {

return false;

}

}

# Anroid 6 动态权限申请

对于6.0以下的权限及在安装的时候，根据权限声明产生一个权限列表，用户只有在同意之后才能完成app的安装，造成了我们想要使用某个app，就要默默忍受其一些不必要的权限（比如是个app都要访问通讯录、短信等）。而在6.0以后，我们可以直接安装，当app需要我们授予不恰当的权限的时候，我们可以予以拒绝（比如：单机的象棋对战，请求访问任何权限，我都是不同意的）。当然你也可以在设置界面对每个app的权限进行查看，以及对单个权限进行授权或者解除授权。

新的权限机制更好的保护了用户的隐私，Google将权限分为两类，一类是Normal Permissions，这类权限一般不涉及用户隐私，是不需要用户进行授权的，比如手机震动、访问网络等；另一类是Dangerous Permission，一般是涉及到用户隐私的，需要用户进行授权，比如读取sdcard、访问通讯录等。

## 权限分类

### Normal Permissions如下

ACCESS\_LOCATION\_EXTRA\_COMMANDS

ACCESS\_NETWORK\_STATE

ACCESS\_NOTIFICATION\_POLICY

ACCESS\_WIFI\_STATE

BLUETOOTH

BLUETOOTH\_ADMIN

BROADCAST\_STICKY

CHANGE\_NETWORK\_STATE

CHANGE\_WIFI\_MULTICAST\_STATE

CHANGE\_WIFI\_STATE

DISABLE\_KEYGUARD

EXPAND\_STATUS\_BAR

GET\_PACKAGE\_SIZE

INSTALL\_SHORTCUT

INTERNET

KILL\_BACKGROUND\_PROCESSES

MODIFY\_AUDIO\_SETTINGS

NFC

READ\_SYNC\_SETTINGS

READ\_SYNC\_STATS

RECEIVE\_BOOT\_COMPLETED

REORDER\_TASKS

REQUEST\_INSTALL\_PACKAGES

SET\_ALARM

SET\_TIME\_ZONE

SET\_WALLPAPER

SET\_WALLPAPER\_HINTS

TRANSMIT\_IR

UNINSTALL\_SHORTCUT

USE\_FINGERPRINT

VIBRATE

WAKE\_LOCK

WRITE\_SYNC\_SETTINGS

### Dangerous Permissions

group:android.permission-group.CONTACTS

permission:android.permission.WRITE\_CONTACTS

permission:android.permission.GET\_ACCOUNTS

permission:android.permission.READ\_CONTACTS

group:android.permission-group.PHONE

permission:android.permission.READ\_CALL\_LOG

permission:android.permission.READ\_PHONE\_STATE

permission:android.permission.CALL\_PHONE

permission:android.permission.WRITE\_CALL\_LOG

permission:android.permission.USE\_SIP

permission:android.permission.PROCESS\_OUTGOING\_CALLS

permission:com.android.voicemail.permission.ADD\_VOICEMAIL

group:android.permission-group.CALENDAR

permission:android.permission.READ\_CALENDAR

permission:android.permission.WRITE\_CALENDAR

group:android.permission-group.CAMERA

permission:android.permission.CAMERA

group:android.permission-group.SENSORS

permission:android.permission.BODY\_SENSORS

group:android.permission-group.LOCATION

permission:android.permission.ACCESS\_FINE\_LOCATION

permission:android.permission.ACCESS\_COARSE\_LOCATION

group:android.permission-group.STORAGE

permission:android.permission.READ\_EXTERNAL\_STORAGE

permission:android.permission.WRITE\_EXTERNAL\_STORAGE

group:android.permission-group.MICROPHONE

permission:android.permission.RECORD\_AUDIO

group:android.permission-group.SMS

permission:android.permission.READ\_SMS

permission:android.permission.RECEIVE\_WAP\_PUSH

permission:android.permission.RECEIVE\_MMS

permission:android.permission.RECEIVE\_SMS

permission:android.permission.SEND\_SMS

permission:android.permission.READ\_CELL\_BROADCASTS

可以通过adb shell pm list permissions -d -g进行查看。

看到上面的dangerous permissions，会发现一个问题，好像危险权限都是一组一组的，恩，没错，的确是这样的，

那么有个问题：分组对我们的权限机制有什么影响吗？

的确是有影响的，如果app运行在Android 6.x的机器上，对于授权机制是这样的。如果你申请某个危险的权限，假设你的app早已被用户授权了**同一组**的某个危险权限，那么系统会立即授权，而不需要用户去点击授权。比如你的app对READ\_CONTACTS已经授权了，当你的app申请WRITE\_CONTACTS时，系统会直接授权通过。此外，对于申请时弹出的dialog上面的文本说明也是对整个权限组的说明，而不是单个权限（ps:这个dialog是不能进行定制的）。

不过需要注意的是，不要对权限组过多的依赖，尽可能对每个危险权限都进行正常流程的申请，因为在后期的版本中这个权限组可能会产生变化。

## API

### 检查权限hasPermissions

**[java]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. **private** **boolean** hasPermissions() {
2. PackageManager pm = getPackageManager();
3. String packageName = getPackageName();
4. **int** granted = (mAudioToggle.isChecked() ? pm.checkPermission(RECORD\_AUDIO, packageName) : PackageManager
5. .PERMISSION\_GRANTED)
6. | pm.checkPermission(WRITE\_EXTERNAL\_STORAGE, packageName);
7. **return** granted == PackageManager.PERMISSION\_GRANTED;
8. }

### 请求权限requestPermissions

g

**[java]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. @TargetApi(M)
2. **private** **void** requestPermissions() {
3. String[] permissions = mAudioToggle.isChecked()
4. ? **new** String[]{WRITE\_EXTERNAL\_STORAGE, RECORD\_AUDIO}
5. : **new** String[]{WRITE\_EXTERNAL\_STORAGE};
6. **boolean** showRationale = **false**;
7. **for** (String perm : permissions) {
8. showRationale |= shouldShowRequestPermissionRationale(perm);
9. }
10. **if** (!showRationale) {
11. requestPermissions(permissions, REQUEST\_PERMISSIONS);
12. **return**;
13. }

### 回调结果onRequestPermissionsResult

Sdf

**[java]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. @Override
2. **public** **void** onRequestPermissionsResult(**int** requestCode, String[] permissions, **int**[] grantResults) {
3. **if** (requestCode == REQUEST\_PERMISSIONS) {
4. **int** granted = PackageManager.PERMISSION\_GRANTED;
5. **for** (**int** r : grantResults) {
6. granted |= r;
7. }
8. **if** (granted == PackageManager.PERMISSION\_GRANTED) {
9. startCaptureIntent();
10. } **else** {
11. toast("No Permission!");
12. }
13. }
14. }

## REF

[在运行时请求权限](https://developer.android.com/training/permissions/requesting?hl=zh-cn)

## checkPermission原理

### app：ActivityManagerNative

public int checkPermission(String permission, int pid, int uid)

throws RemoteException {

Parcel data = Parcel.obtain();

Parcel reply = Parcel.obtain();

data.writeInterfaceToken(IActivityManager.descriptor);

data.writeString(permission);

data.writeInt(pid);

data.writeInt(uid);

mRemote.transact(CHECK\_PERMISSION\_TRANSACTION, data, reply, 0);

reply.readException();

int res = reply.readInt();

data.recycle();

reply.recycle();

return res;

}

### ActMS. checkPermission

public int checkPermission(String permission, int pid, int uid) {

if (permission == null) {

return PackageManager.PERMISSION\_DENIED;

}

return checkComponentPermission(permission, pid, UserHandle.getAppId(uid), -1, true);

}

### ActMS .checkComponentPermission

int checkComponentPermission(String permission, int pid, int uid,

int owningUid, boolean exported) {

if (pid == MY\_PID) {

return PackageManager.PERMISSION\_GRANTED;

}

return ActivityManager.checkComponentPermission(permission, uid,

owningUid, exported);

}

### ActivityManager.checkComponentPermission

Dsf

/\*\* @hide \*/

public static int checkComponentPermission(String permission, int uid,

int owningUid, boolean exported) {

// Root, system server get to do everything.

<!--root及System能获取所有权限-->

if (uid == Process.ROOT\_UID || uid == Process.SYSTEM\_UID) {

return PackageManager.PERMISSION\_GRANTED;

}

。。。

<!--普通的权限查询-->

try {

return AppGlobals.getPackageManager()

.checkUidPermission(permission, uid);

} catch (RemoteException e) {

// Should never happen, but if it does... deny!

Slog.e(TAG, "PackageManager is dead?!?", e);

}

return PackageManager.PERMISSION\_DENIED;

}

### PkgMS.checkUidPermission

最终调用PackageManagerService.java去查看是否有权限，到这里，我们只需要知道权限的查询其实是通过PKMS来进行的。心里先有个底，权限的更新，持久化，恢复都是通过PKMS来进行的。

### PMS

https://www.jianshu.com/p/9938d367b6db