**PMS之权限**

1. 权限的种类

静态权限和runtime权限

1. 权限处理流程

1.在scan pkg file 的时候进行PackageParser.parsePackage操作，会分别将apk中的AndroidManifest中的permission tag中的所有定义的权限创建permission和PermissionInfo对象保存在PackageParser.Package对象的permissions容器中，将apk请求时需要的权限保存在requestedPermissions容器中。（获取apk定义了哪些权限和请求时需要哪些权限）

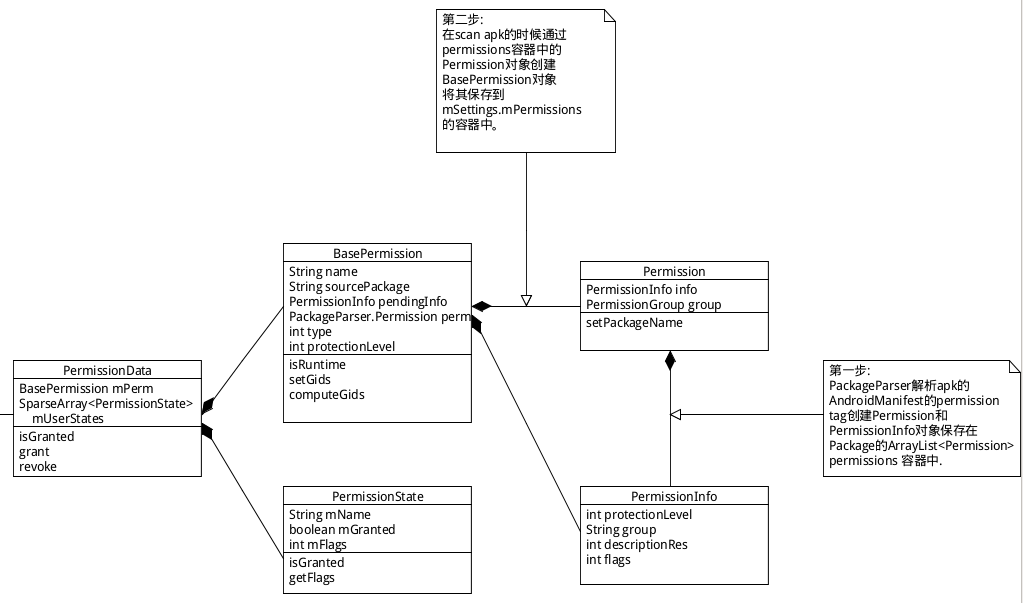
2.在scan apk的时候通过permissions容器中的Permission对象创建BasePermission对象将其保存到mSettings.mPermissions的容器中。（将定义的权限保存到系统中）

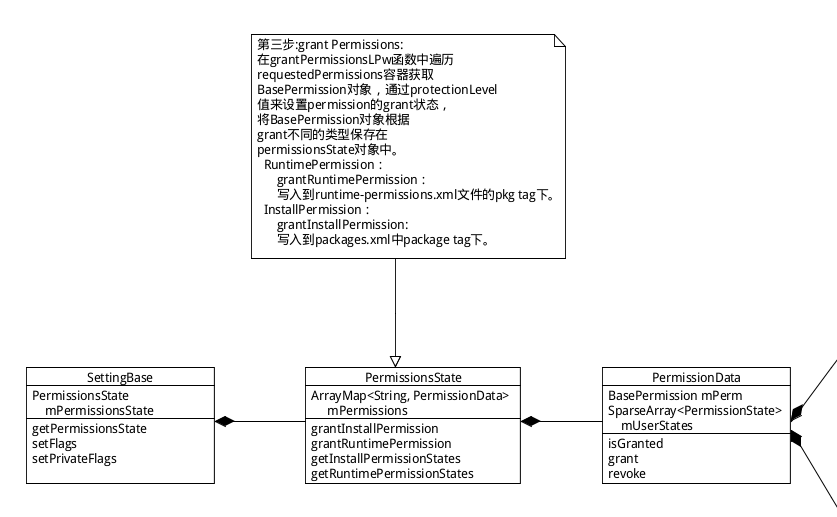
3.在grantPermissionsLPw函数中遍历requestedPermissions容器获取BasePermission对象，通过protectionLevel值来设置permission的grant状态，将BasePermission对象根据grant不同的类型保存在permissionsState对象中。

RuntimePermission：grantRuntimePermission：写入到runtime-permissions.xml文件的pkg tag下。

InstallPermission：grantInstallPermission:写入到packages.xml中package tag下。

（将请求时需要的权限进行分类为安装权限和运行时权限）





请求的Runtime 权限的产生：

private void grantPermissionsLPw(PackageParser.Package pkg, boolean replace,

String packageOfInterest) {

。。。。。。

final int level = bp.protectionLevel & PermissionInfo.PROTECTION\_MASK\_BASE;

switch (level) {

case PermissionInfo.PROTECTION\_DANGEROUS: {

if (pkg.applicationInfo.targetSdkVersion <= Build.VERSION\_CODES.LOLLIPOP\_MR1) {

// For legacy apps dangerous permissions are install time ones.

grant = GRANT\_INSTALL\_LEGACY;

} else if (origPermissions.hasInstallPermission(bp.name)) {

// For legacy apps that became modern, install becomes runtime.

grant = GRANT\_UPGRADE;

} else if (mPromoteSystemApps

&& isSystemApp(ps)

&& mExistingSystemPackages.contains(ps.name)) {

// For legacy system apps, install becomes runtime.

// We cannot check hasInstallPermission() for system apps since those

// permissions were granted implicitly and not persisted pre-M.

grant = GRANT\_UPGRADE;

} else {

**// For modern apps keep runtime permissions unchanged.**

**grant = GRANT\_RUNTIME;**

}

} break;

**并不是说权限的保护等级为ROTECTION\_DANGEROUS该权限就会变成runtime权限。**

4.权限的校验：

Sample：组件broadcast在注册的时候可以指定需要的权限，将相应的permission保存在ams中。

context.registerReceiver(mStrongAuthTimeoutReceiver, strongAuthTimeoutFilter,

PERMISSION\_SELF, null /\* handler \*/);

当进行派送broadcast的时候，ams会为发送broadcast的进程检查是否有注册的receiver的

权限。

int perm = mService.checkComponentPermission(filter.requiredPermission,

r.callingPid, r.callingUid, -1, true);

if (perm != PackageManager.PERMISSION\_GRANTED) {

Slog.w(TAG, "Permission Denial: broadcasting "

+ r.intent.toString()

+ " from " + r.callerPackage + " (pid="

+ r.callingPid + ", uid=" + r.callingUid + ")"

+ " requires " + filter.requiredPermission

+ " due to registered receiver " + filter);

skip = true;

}

最终调用PMS的checkUidPermission函数来检查发送broadcast的app的ui是否有receiver中定义的权限，并且已经granted。

public int checkUidPermission(String permName, int uid) {

Object obj = mSettings.getUserIdLPr(UserHandle.getAppId(uid));

if (obj != null) {

final SettingBase ps = (SettingBase) obj;

final PermissionsState permissionsState = ps.getPermissionsState();

if (permissionsState.hasPermission(permName, userId)) {

return PackageManager.PERMISSION\_GRANTED;

}

if (Manifest.permission.ACCESS\_COARSE\_LOCATION.equals(permName) && permissionsState

.hasPermission(Manifest.permission.ACCESS\_FINE\_LOCATION, userId)) {

return PackageManager.PERMISSION\_GRANTED;

}

}

可以通过权限来控制有些app发送的broadcast不用接收处理。

5.运行时权限的修改：