PMS之update

1. 在安装app的时候哪些情况下是进行replace安装。

1.当从mainfeist中解析出tag “original-package”说明mOriginalPackages不为空，同时mSettings.mRenamedPackages中如果存在安装apk的package name 对应的old name那么该安装的apk为一次更新的操作。

// Packages that have been renamed since they were first installed.

// Keys are the new names of the packages, values are the original

// names. The packages appear everwhere else under their original

// names.

final ArrayMap<String, String> mRenamedPackages = new ArrayMap<String, String>();

2.当packageManager中的mPackages已经包含该pkgName。

1. 在replace之前和安装相同的步骤为：

拷贝apk文件、解压lib库、选择abi、进行odex优化。

1. 比较old 和new package的TargetSdk如果old的TargetSdk大于L，新的package的TargetSdk小于L那么停在更新。
2. 调用verifySignaturesLP函数进行old和new package的签名验证。
3. 检查new packeage中所有定义的permission和mSettings.mPermissions中有相同名称的permission的签名是否相同，如果不相同而且mSettings.mPermissions中该permission的sourcePackage为android，那么退出安装。如果不是则从new package的permissions容器中移除该permission。

6.调用replacePackageLI函数进行更新操作：

1.获取old apk的Package和PackageSetting信息，调用checkUpgradeKeySetLP函数或者函数compareSignatures进行检查signatures。

2.检查有哪些user对该package进行了install的动作。

3.当old apk为system apk调用replaceSystemPackageLI函数进行更新，非system apk调用replaceNonSystemPackageLI函数进行更新。

replaceSystemPackageLI：（不删除system app code disable app）

手机重启会自动恢复。

1.调用killApplication函数杀掉进程。

2.将oldPkg的data Structures从PackageManager中所有的容器中移除，包括mPackages、

mProviders、mServices、mReceivers、mActivities、mInstrumentation等。

3.调用函数mSettings.disableSystemPackageLPw进行PackageSetting创建。

将old的PackageSetting对象放入到mDisabledSysPackages容器中（不会去删除system app的code和resource），使用old的PackageSetting对象创建new PackageSetting对象，调用replacePackageLPw函数使用new PackageSetting对象替换old。如果old PackageSetting有对应的SharedUserSetting那么将new PackageSetting对象add到SharedUserSetting中。使用old PackageSetting对象的appId和新的PackageSetting对象来更新mUserIds等容器。

boolean disableSystemPackageLPw(String name) {

final PackageSetting p = mPackages.get(name);

if(p == null) {

Log.w(PackageManagerService.TAG, "Package:"+name+" is not an installed package");

return false;

}

final PackageSetting dp = mDisabledSysPackages.get(name);

if (dp == null) {

if((p.pkg != null) && (p.pkg.applicationInfo != null)) {

p.pkg.applicationInfo.flags |= ApplicationInfo.FLAG\_UPDATED\_SYSTEM\_APP;

}

**mDisabledSysPackages**.put(name, p);

PackageSetting newp = new PackageSetting(p);

replacePackageLPw(name, newp);

return true;

}

return false;

}

private void replacePackageLPw(String name, PackageSetting newp) {

final PackageSetting p = mPackages.get(name);

if (p != null) {

if (p.sharedUser != null) {

p.sharedUser.removePackage(p);

p.sharedUser.addPackage(newp);

} else {

replaceUserIdLPw(p.appId, newp);

}

}

mPackages.put(name, newp);

}

4.调用deleteCodeCacheDirsLI函数删除/code\_cache目录。

5.扫描new package：scanPackageLI。

6.扫描成功后调用updateSettingsLI更新配置文件。

replaceNonSystemPackageLI：

1.deletePackageLI

// First delete the existing package while retaining the data directory

if (!deletePackageLI(pkgName, null, true, null, null, PackageManager.DELETE\_KEEP\_DATA,

res.removedInfo, true)) {

// If the existing package wasn't successfully deleted

res.setError(INSTALL\_FAILED\_REPLACE\_COULDNT\_DELETE, "replaceNonSystemPackageLI");

deletedPkg = false;

}

PackageManager.DELETE\_KEEP\_DATA：不删除用户的data目录的数据，在该函数中会调用removePackageLI函数清除Package Data Structures 在packageManager中容器中相关的信息。

创建InstallArgs用来删除old package的code dex等文件。

2.deleteCodeCacheDirsLI

3.scanPackageLI

4.updateSettingsLI