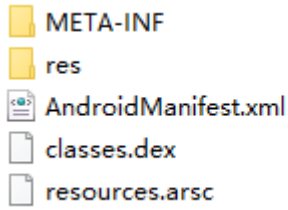
APK Installation(android 5.0)

安装应用的实质

一般 os：安装应用就是解压压缩包，并复制文件到指定的路径的过程。可能还需要在注册表中注册信息，创建快捷方式等。

Android：解析需要安装的 apk，将 apk 文件拷贝到特定的目录下，然后将 androidmenifest.xml 中的信息解析出来放到对应的全局列表中，mProviders，mServices，mReceivers，mActivities。这些工作大多是由一个系统服务 PackageManagerService 提供的。

APK：android package 的缩写，可以直接解压，代码做了编译，但是资源文件和 androidmenifest.xml 保留在目录下



两种安装的流程：

1.开机过程中初始化 PackageManagerService 的时候扫描目录下的 APK 进行安装。

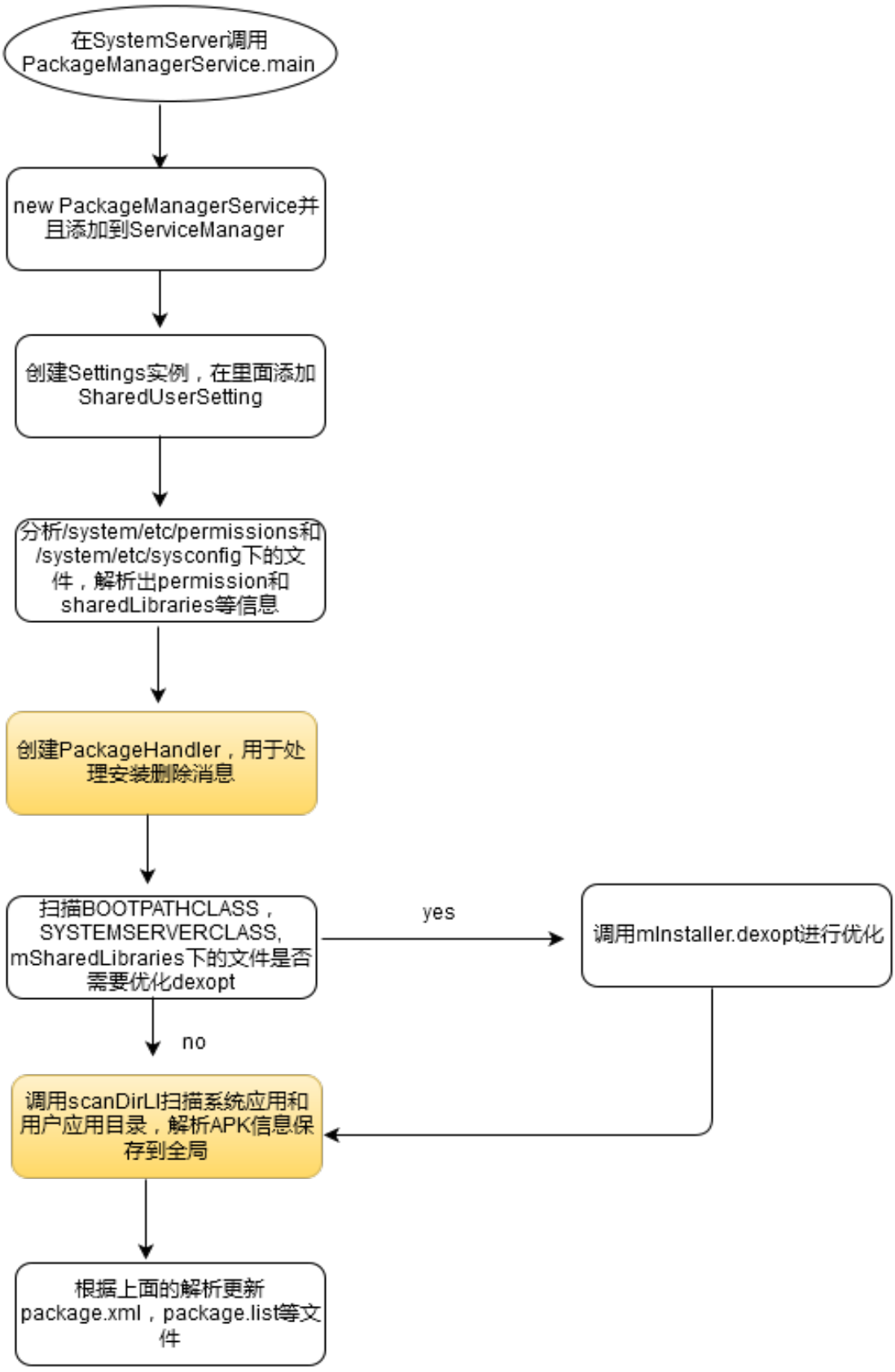
2.开机后安装 APK，开发常用 adb 命令安装应用

```
adb install [-lrtsd] <file>
adb install-multiple [-lrtsdp] <file...>
                        - push this package file to the device and install it
                          (-l: forward lock application)
                          (-r: replace existing application)
                          (-t: allow test packages)
                          (-s: install application on sdcard)
                          (-d: allow version code downgrade)
                          (-p: partial application install)

adb push [-p] <local> <remote>
                        - copy file/dir to device
                          ('-p' to display the transfer progress)

adb uninstall [-k] <package> - remove this app package from the device
                          ('-k' means keep the data and cache directories)
```

PackageManagerService 的初始化流程



代码简析

SystemServer.java

```
private void startBootstrapServices() {
    // Wait for installd to finish starting up so that it has a chance to
    // create critical directories such as /data/user with the appropriate
    // permissions.  We need this to complete before we initialize other services.
    Installer installer = mSystemServiceManager.startService(Installer.class);

// Start the package manager.
Slog.i(TAG, "Package Manager");
mPackageManagerService = PackageManagerService.main(mSystemContext, installer,
        mFactoryTestMode != FactoryTest.FACTORY_TEST_OFF, mOnlyCore);
```

PackageManagerService.java

```
public static final PackageManagerService main(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
    PackageManagerService m = new PackageManagerService(context, installer,
            factoryTest, onlyCore);
    ServiceManager.addService("package", m);
    return m;
}
```

PackageManagerService()

创建 Settings 实例，添加 SharedUserSetting 信息

```
mSettings = new Settings(context);
mSettings.addSharedUserLPw("android.uid.system", Process.SYSTEM_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
mSettings.addSharedUserLPw("android.uid.phone", RADIO_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
mSettings.addSharedUserLPw("android.uid.log", LOG_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
mSettings.addSharedUserLPw("android.uid.nfc", NFC_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
mSettings.addSharedUserLPw("android.uid.bluetooth", BLUETOOTH_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
mSettings.addSharedUserLPw("android.uid.shell", SHELL_UID,
        ApplicationInfo.FLAG_SYSTEM|ApplicationInfo.FLAG_PRIVILEGED);
```

分析/system/etc/permissions 和/system/etc/sysconfig 下的文件，解析出

permission 和 sharedLibraries 等信息，将信息放入

mGlobalGids，mSystemPermissions，mSharedLibraries 等变量中。

```
SystemConfig systemConfig = SystemConfig.getInstance();
mGlobalGids = systemConfig.getGlobalGids();
mSystemPermissions = systemConfig.getSystemPermissions();
mAvailableFeatures = systemConfig.getAvailableFeatures();
```

```
SystemConfig() {
    // Read configuration from system
    readPermissions(Environment.buildPath(
            Environment.getRootDirectory(), "etc", "sysconfig"), false);
    // Read configuration from the old permissions dir
    readPermissions(Environment.buildPath(
            Environment.getRootDirectory(), "etc", "permissions"), false);
    // Only read features from OEM config
    readPermissions(Environment.buildPath(
            Environment.getOemDirectory(), "etc", "sysconfig"), true);
    readPermissions(Environment.buildPath(
            Environment.getOemDirectory(), "etc", "permissions"), true);
}
```

Platform.xml

```
<permission name="android.permission.ACCESS_FM_RADIO" >
    <group gid="media" />
</permission>

<assign-permission name="android.permission.MODIFY_AUDIO_SETTINGS" uid="media" />
<assign-permission name="android.permission.ACCESS_SURFACE_FLINGER" uid="media" />
<assign-permission name="android.permission.WAKE_LOCK" uid="media" />
<assign-permission name="android.permission.UPDATE_DEVICE_STATS" uid="media" />
<assign-permission name="android.permission.UPDATE_APP_OPS_STATS" uid="media" />

<library name="android.test.runner"
        file="/system/framework/android.test.runner.jar" />
<library name="javax.obex"
        file="/system/framework/javax.obex.jar"/>
<library name="javax.btobex"
        file="/system/framework/javax.btobex.jar"/>
```

创建 PackageHandler，用于处理安装删除消息

```
mHandlerThread = new ServiceThread(TAG,
        Process.THREAD_PRIORITY_BACKGROUND, true /*allowIo*/);
mHandlerThread.start();
mHandler = new PackageHandler(mHandlerThread.getLooper());
```

扫描 BOOTPATHCLASS，SYSTEMSERVERCLASS,mSharedLibraries 下的文件是否需要优化 dexopt

```
try {
    byte dexoptRequired = DexFile.isDexOptNeededInternal(lib, null,
                                                         dexCodeInstructionSet,
                                                         false);
    if (dexoptRequired != DexFile.UP_TO_DATE) {
        alreadyDexOpted.add(lib);

        // The list of "shared libraries" we have at this point is
        if (dexoptRequired == DexFile.DEXOPT_NEEDED) {
            mInstaller.dexopt(lib, Process.SYSTEM_UID, true, dexCodeInstructionSet);
        } else {
            mInstaller.patchoat(lib, Process.SYSTEM_UID, true, dexCodeInstructionSet);
        }
    }
```

调用 scanDirLI 扫描系统应用和用户应用目录，解析 APK 信息保存到全局

```
final File privilegedAppDir = new File(Environment.getRootDirectory(), "priv-app");
scanDirLI(privilegedAppDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR
        | PackageParser.PARSE_IS_PRIVILEGED, scanFlags, 0);
```

根据上面的解析更新 package.xml，package.list 等文件

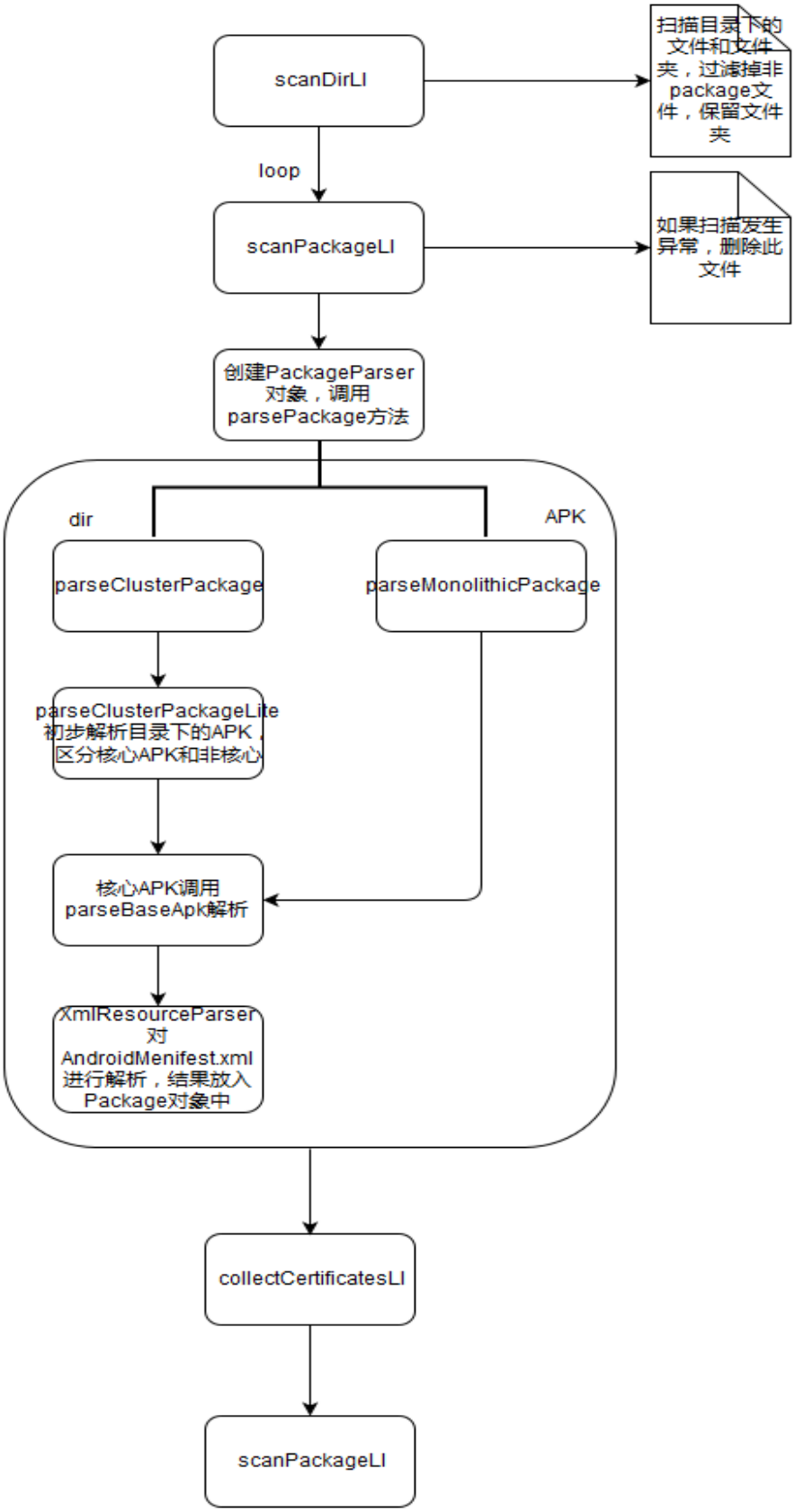
```
updatePermissionsLPw(null, null, UPDATE_PERMISSIONS_ALL
        | (regrantPermissions
                ? (UPDATE_PERMISSIONS_REPLACE_PKG|UPDATE_PERMISSIONS_REPLACE_ALL)
                : 0));
```

```
<package name="android" codePath="/system/framework/framework-res.apk" nativeLibraryPath="/system/lib/framework-res" primaryCpuAbi="armeabi-v7a" flags="1078
    <sigs count="1">
        <cert index="0" />
    </sigs>
    <proper-signing-keyset identifier="1" />
    <signing-keyset identifier="1" />
</package>
```

分析 scanDirLI—step1:主要目的是解析出 Package 对象

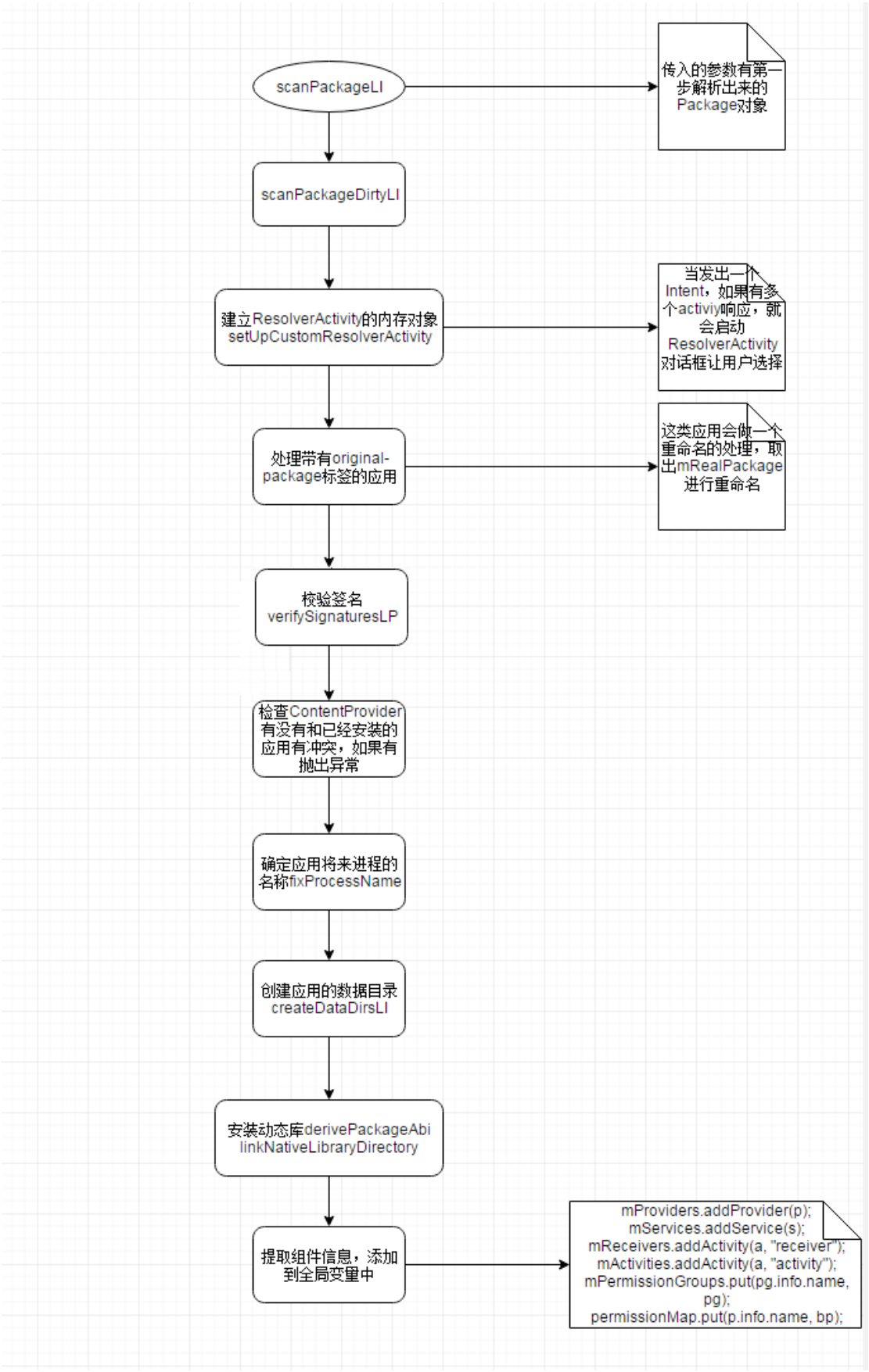
PackageParser.Package

```
public String packageName;
public String codePath;
public final ArrayList<Permission> permissions = new
ArrayList<Permission>(0);
        public final ArrayList<PermissionGroup> permissionGroups = new
ArrayList<PermissionGroup>(0);
public final ArrayList<Activity> activities = new ArrayList<Activity>(0);
public final ArrayList<Activity> receivers = new ArrayList<Activity>(0);
public final ArrayList<Provider> providers = new ArrayList<Provider>(0);
public final ArrayList<Service> services = new ArrayList<Service>(0);
public ArrayList<String> mOriginalPackages = null;
```


scanDirLI
扫描目录下的文件和文件夹，过滤掉非package文件，保留文件夹
loop
scanPackageLI
如果扫描发生异常，删除此文件
创建PackageParser对象，调用parsePackage方法
dir
APK
parseClusterPackage
parseMonolithicPackage
parseClusterPackageLite初步解析目录下的APK，区分核心APK和非核心
核心APK调用parseBaseApk解析
XmlResourceParser对AndroidMenifest.xml进行解析，结果放入Package对象中
collectCertificatesLI
scanPackageLI

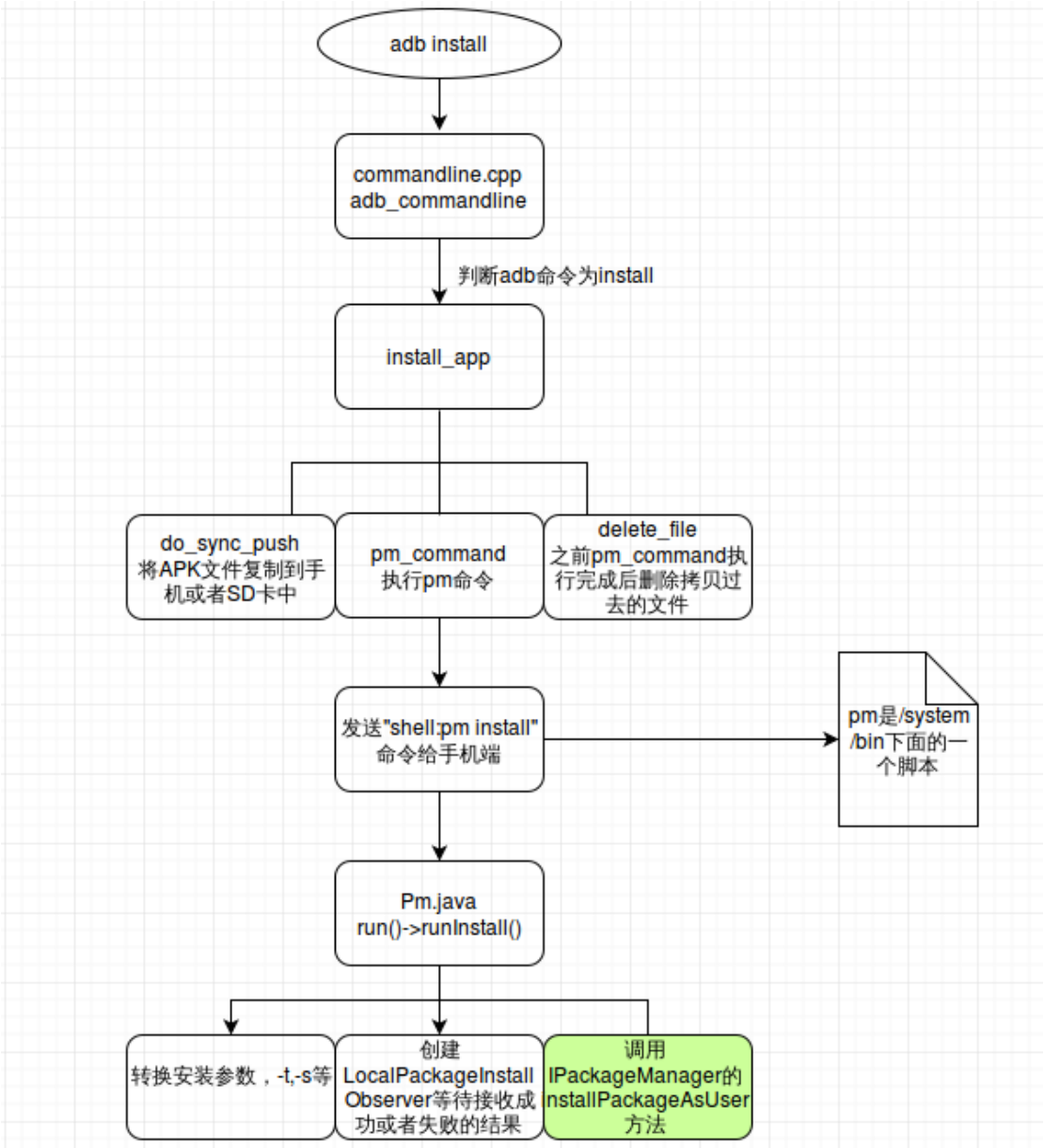
scanPackageLI--step2：主要任务将解析出来的 Package 对象填充在 mActivities， mReceivers， mServices， mProviders 这四个关键的全局变量中

```
final ActivityIntentResolver mActivities = new ActivityIntentResolver();
final ActivityIntentResolver mReceivers = new ActivityIntentResolver();
final ServiceIntentResolver mServices = new ServiceIntentResolver();
final ProviderIntentResolver mProviders = new ProviderIntentResolver();
```


scanPackageLI
传入的参数有第一步解析出来的Package对象
scanPackageDirtyLI
建立ResolverActivity的内存对象setUpCustomResolverActivity
当发出一个Intent，如果有多个activiy响应，就会启动ResolverActivity对话框让用户选择
处理带有original-package标签的应用
这类应用会做一个重命名的处理，取出mRealPackage进行重命名
校验签名verifySignaturesLP
检查ContentProvider有没有和已经安装的应用有冲突，如果有抛出异常
确定应用将来进程的名称fixProcessName
创建应用的数据目录createDataDirsLI
安装动态库derivePackageAbilinkNativeLibraryDirectory
提取组件信息，添加到全局变量中
mProviders.addProvider(p);
mServices.addService(s);
mReceivers.addActivity(a, "receiver");
mActivities.addActivity(a, "activity");
mPermissionGroups.put(pg.info.name, pg);
permissionMap.put(p.info.name, bp);

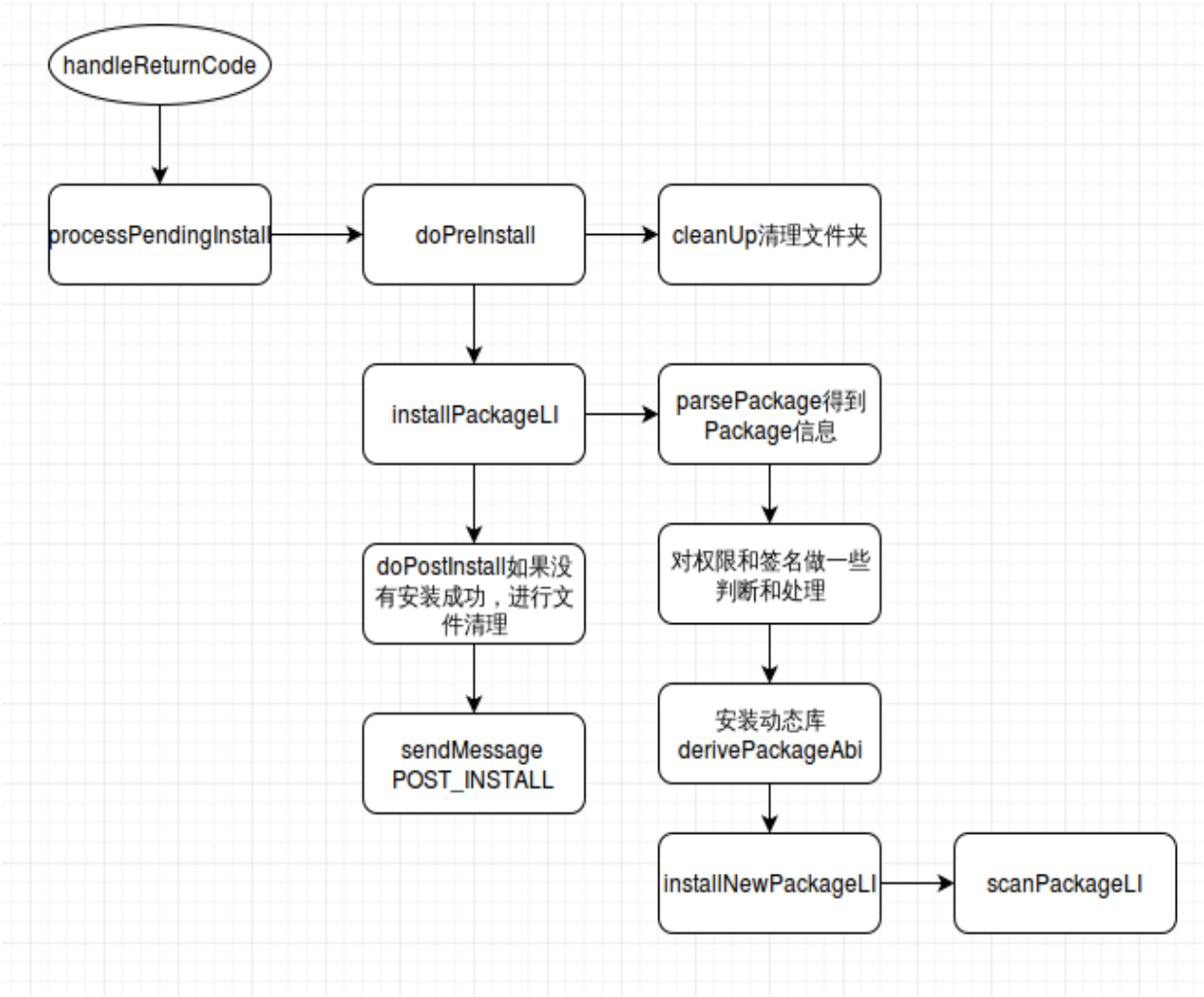
通过 adb install 安装应用

step1:commandline.cpp->PackageManagerService.java



step2:PackageManagerService.java 中的安装过程

step3:handleReturnCode 分析



小结

只是分析了安装过程中的大致流程，很多方法中的细节逻辑还需要仔细研究。

Permission

Signature

Uninstall process

...