APK Installation(android 5.0)

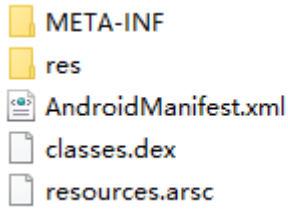
chenxu

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



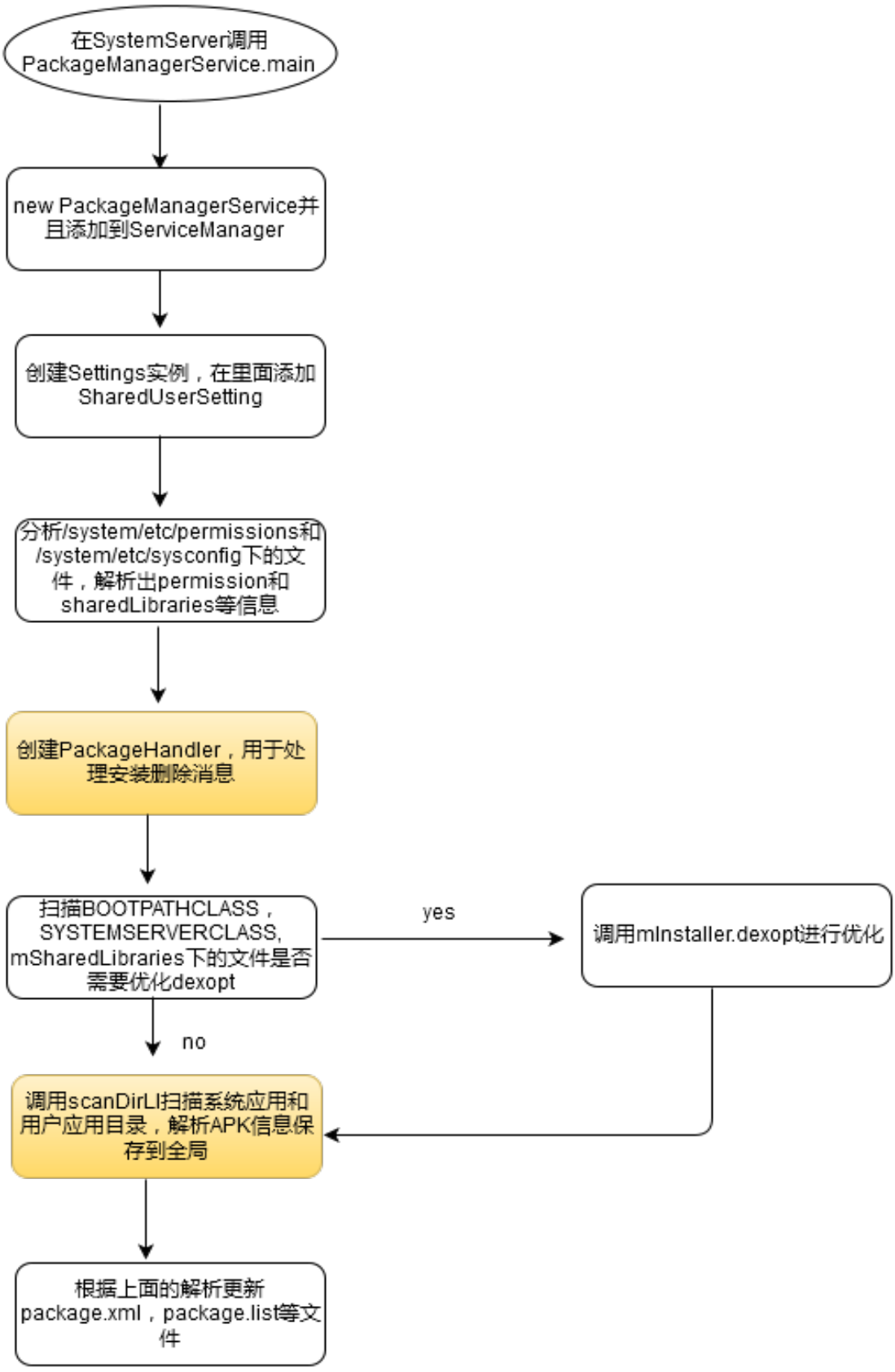
两种安装的流程：

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

```
/**
 * The main entry point from zygote.
 */
public static void main(String[] args) {
    new SystemServer().run();
}
```

```
    // Start services.
    try {
        startBootstrapServices();
        startCoreServices();
        startOtherServices();
    } catch (Throwable ex) {
        Slog.e("System", "************************************************");
        Slog.e("System", "************ Failure starting system services", ex);
        throw ex;
    }
```

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

PackageManagerService.java

```
public static 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()

step1:创建 Settings 实例，添加 SharedUserSetting 信息

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

```
SharedUserSetting addSharedUserLPw(String name, int uid, int pkgFlags, int pkgPrivateFlags) {
    SharedUserSetting s = mSharedUsers.get(name);
    if (s != null) {
        if (s.userId == uid) {
            return s;
        }
        PackageManagerService.reportSettingsProblem(Log.ERROR,
                "Adding duplicate shared user, keeping first: " + name);
        return null;
    }
    s = new SharedUserSetting(name, pkgFlags, pkgPrivateFlags);
    s.userId = uid;
    if (addUserIdLPw(uid, s, name)) {
        mSharedUsers.put(name, s);
        return s;
    }
    return null;
}
```

```
  private boolean addUserIdLPw(int uid, Object obj, Object name) {
      if (uid > Process.LAST_APPLICATION_UID) {
          return false;
      }

      if (uid >= Process.FIRST_APPLICATION_UID) {
          int N = mUserIds.size();
          final int index = uid - Process.FIRST_APPLICATION_UID;
          while (index >= N) {
              mUserIds.add(null);
              N++;
          }
          if (mUserIds.get(index) != null) {
              PackageManagerService.reportSettingsProblem(Log.ERROR,
                      "Adding duplicate user id: " + uid
                      + " name=" + name);
              return false;
          }
          mUserIds.set(index, obj);
      } else {
          if (mOtherUserIds.get(uid) != null) {
              PackageManagerService.reportSettingsProblem(Log.ERROR,
                      "Adding duplicate shared id: " + uid
                              + " name=" + name);
              return false;
          }
          mOtherUserIds.put(uid, obj);
      }
      return true;
  }
```

step2: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();
```

```
public static SystemConfig getInstance() {
    synchronized (SystemConfig.class) {
        if (sInstance == null) {
            sInstance = new SystemConfig();
        }
        return sInstance;
    }
}
```

```
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);
}
```

```
            if ("group".equals(name) && !onlyFeatures) {
                String gidStr = parser.getAttributeValue(null, "gid");
                if (gidStr != null) {
                    int gid = android.os.Process.getGidForName(gidStr);
                    mGlobalGids = appendInt(mGlobalGids, gid);
                } else {
                    Slog.w(TAG, "<group> without gid in " + permFile + " at "
                            + parser.getPositionDescription());
                }

                XmlUtils.skipCurrentTag(parser);
                continue;
            } else if ("permission".equals(name) && !onlyFeatures) {
```

“group”->mGlobalGids

“assign-permission”->mSystemPermissions

"library"->mSharedLibraries

"feature"->mAvailableFeatures

……还有很多其他的字段。

```
<permission name="android.permission.BLUETOOTH_ADMIN" >
    <group gid="net_bt_admin" />
</permission>

<permission name="android.permission.BLUETOOTH" >
    <group gid="net_bt" />
</permission>

<permission name="android.permission.BLUETOOTH_STACK" >
    <group gid="net_bt_stack" />
</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" />

<assign-permission name="android.permission.ACCESS_SURFACE_FLINGER" uid="graphics" />
```

```
<library name="android.test.runner"
        file="/system/framework/android.test.runner.jar" />
<library name="javax.obex"
        file="/system/framework/javax.obex.jar" />
<library name="org.apache.http.legacy"
        file="/system/framework/org.apache.http.legacy.jar" />
```

Step3:创建 PackageHandler，用于处理安装删除消息

```
    synchronized (mPackages) {
        mHandlerThread = new ServiceThread(TAG,
                Process.THREAD_PRIORITY_BACKGROUND, true /*allowIo*/);
        mHandlerThread.start();
        mHandler = new PackageHandler(mHandlerThread.getLooper());
        Watchdog.getInstance().addThread(mHandler, WATCHDOG_TIMEOUT);
```

step4:扫描 BOOTPATHCLASS，SYSTEMSERVERCLASS,mSharedLibraries 下的文件是否需要优化 dexopt

```
    final String bootClassPath = System.getenv("BOOTCLASSPATH");
    final String systemServerClassPath = System.getenv("SYSTEMSERVERCLASSPATH");

    if (bootClassPath != null) {
        String[] bootClassPathElements = splitString(bootClassPath, ':');
        for (String element : bootClassPathElements) {
            alreadyDexOpted.add(element);
        }
    } else {
        Slog.w(TAG, "No BOOTCLASSPATH found!");
    }

    if (systemServerClassPath != null) {
        String[] systemServerClassPathElements = splitString(systemServerClassPath, ':');
        for (String element : systemServerClassPathElements) {
            alreadyDexOpted.add(element);
        }
    } else {
        Slog.w(TAG, "No SYSTEMSERVERCLASSPATH found!");
    }
```

```
/**
 * Ensure all external libraries have had dexopt run on them.
 */
if (mSharedLibraries.size() > 0) {
    // NOTE: For now, we're compiling these system "shared libraries"
    // (and framework jars) into all available architectures. It's possible
    // to compile them only when we come across an app that uses them (there's
    // already logic for that in scanPackageLI) but that adds some complexity.
    for (String dexCodeInstructionSet : dexCodeInstructionSets) {
        for (SharedLibraryEntry libEntry : mSharedLibraries.values()) {
            final String lib = libEntry.path;
            if (lib == null) {
                continue;
            }

            try {
                int dexoptNeeded = DexFile.getDexOptNeeded(lib, null, dexCodeInstructionSet
                if (dexoptNeeded != DexFile.NO_DEXOPT_NEEDED) {
                    alreadyDexOpted.add(lib);
                    mInstaller.dexopt(lib, Process.SYSTEM_UID, true, dexCodeInstructionSet,
                }
            } catch (FileNotFoundException e) {
                Slog.w(TAG, "Library not found: " + lib);
            } catch (IOException e) {
                Slog.w(TAG, "Cannot dexopt " + lib + "; is it an APK or JAR? "
                        + e.getMessage());
            }
        }
    }
}
```

step5: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

```
// Collect vendor overlay packages.
// (Do this before scanning any apps.)
// For security and version matching reason, only consider
// overlay packages if they reside in VENDOR_OVERLAY_DIR.
File vendorOverlayDir = new File(VENDOR_OVERLAY_DIR);
scanDirLI(vendorOverlayDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR, scanFlags | SCAN_TRUSTED_OVERLAY, 0);

// Find base frameworks (resource packages without code).
scanDirLI(frameworkDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR
        | PackageParser.PARSE_IS_PRIVILEGED,
        scanFlags | SCAN_NO_DEX, 0);

// Collected privileged system packages.
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);

// Collect ordinary system packages.
final File systemAppDir = new File(Environment.getRootDirectory(), "app");
scanDirLI(systemAppDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR, scanFlags, 0);
```

```
// Collect all vendor packages.
File vendorAppDir = new File("/vendor/app");
try {
    vendorAppDir = vendorAppDir.getCanonicalFile();
} catch (IOException e) {
    // failed to look up canonical path, continue with original one
}
scanDirLI(vendorAppDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR, scanFlags, 0);

// Collect all OEM packages.
final File oemAppDir = new File(Environment.getOemDirectory(), "app");
scanDirLI(oemAppDir, PackageParser.PARSE_IS_SYSTEM
        | PackageParser.PARSE_IS_SYSTEM_DIR, scanFlags, 0);
```

step6:根据上面的解析更新 package.xml，package.list 等文件

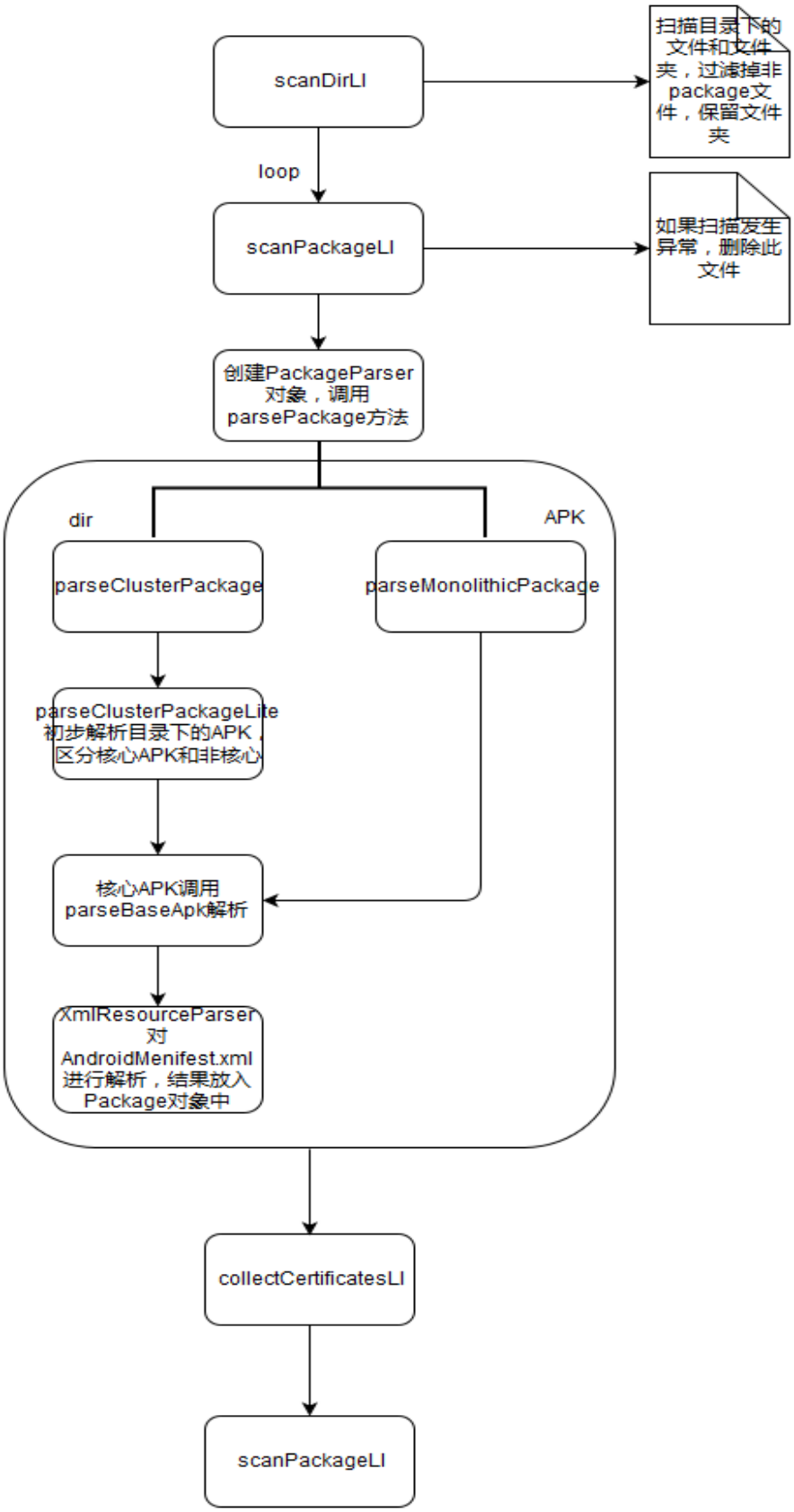
```
int updateFlags = UPDATE_PERMISSIONS_ALL;
if (ver.sdkVersion != mSdkVersion) {
    Slog.i(TAG, "Platform changed from " + ver.sdkVersion + " to "
            + mSdkVersion + "; regranting permissions for internal storage");
    updateFlags |= UPDATE_PERMISSIONS_REPLACE_PKG | UPDATE_PERMISSIONS_REPLACE_ALL;
}
updatePermissionsLPw(null, null, StorageManager.UUID_PRIVATE_INTERNAL, updateFlags);
ver.sdkVersion = mSdkVersion;
```

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

PackageParser.Package

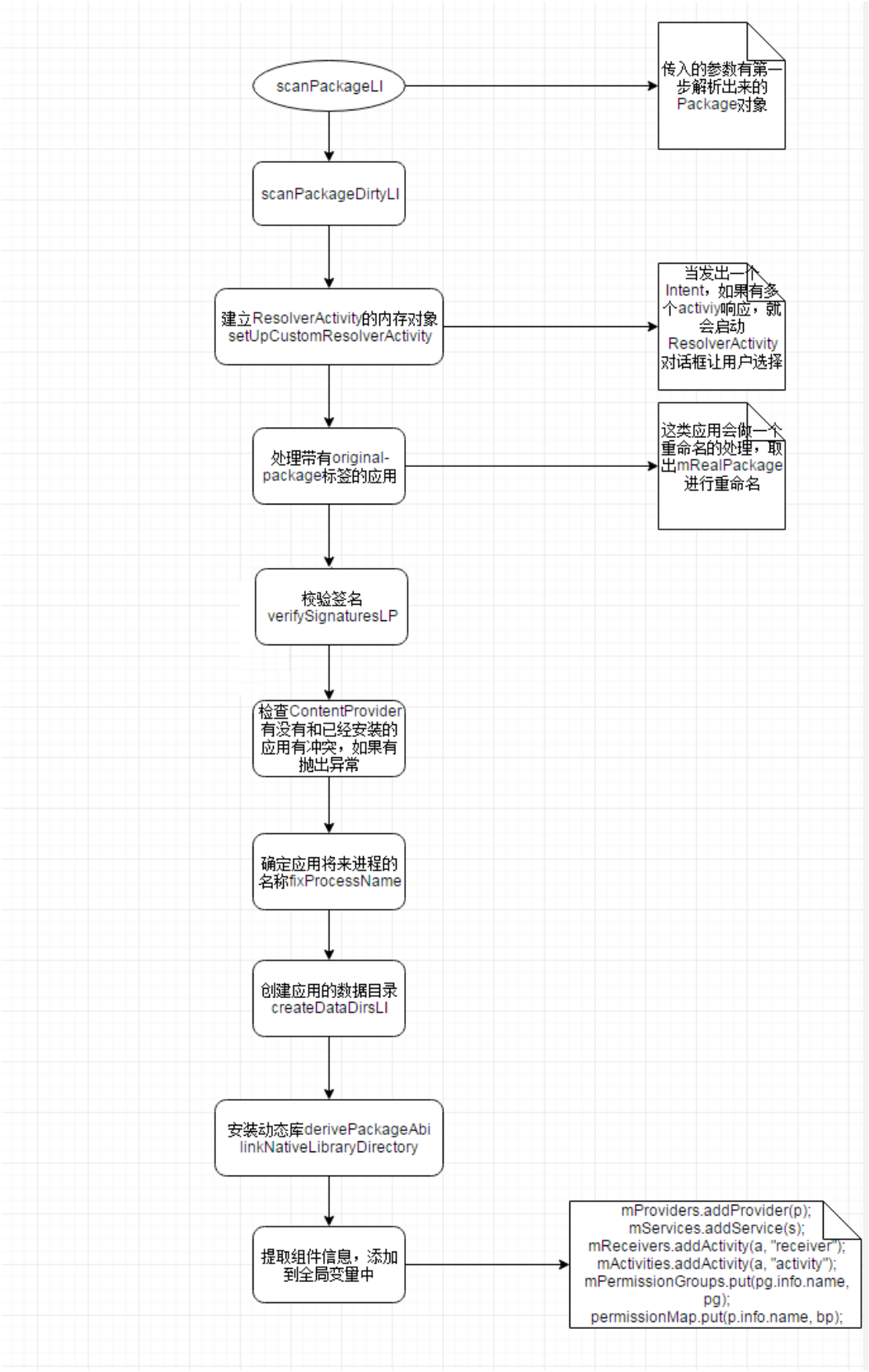
```
public final static class Package {

    public String packageName;

    /** Names of any split APKs, ordered by parsed splitName */
    public String[] splitNames;

    // TODO: work towards making these paths invariant

    public String volumeUuid;
```

```
// For now we only support one application per package.
public final ApplicationInfo applicationInfo = new ApplicationInfo();

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 final ArrayList<Instrumentation> instrumentation = new ArrayList<Instrumentation>(0);

public final ArrayList<String> requestedPermissions = new ArrayList<String>();

public ArrayList<String> protectedBroadcasts;
```


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

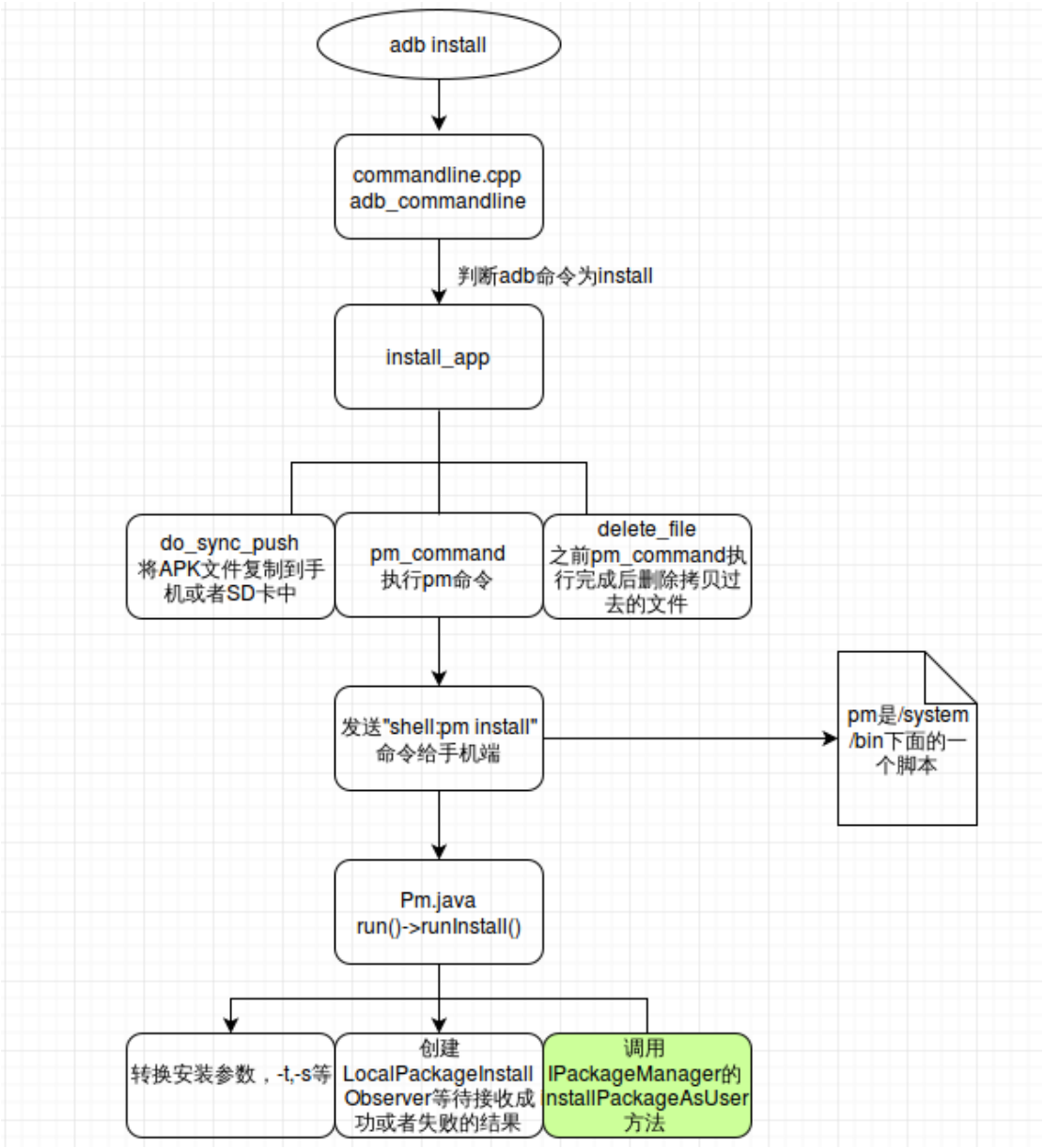
scanPackageLI--step2：主要任务将解析出来的 Package 对象填充在 mActivities， mReceivers， mServices， mProviders 这四个关键的全局变量中

```
// All available activities, for your resolving pleasure.
final ActivityIntentResolver mActivities =
        new ActivityIntentResolver();

// All available receivers, for your resolving pleasure.
final ActivityIntentResolver mReceivers =
        new ActivityIntentResolver();

// All available services, for your resolving pleasure.
final ServiceIntentResolver mServices = new ServiceIntentResolver();

// All available providers, for your resolving pleasure.
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);

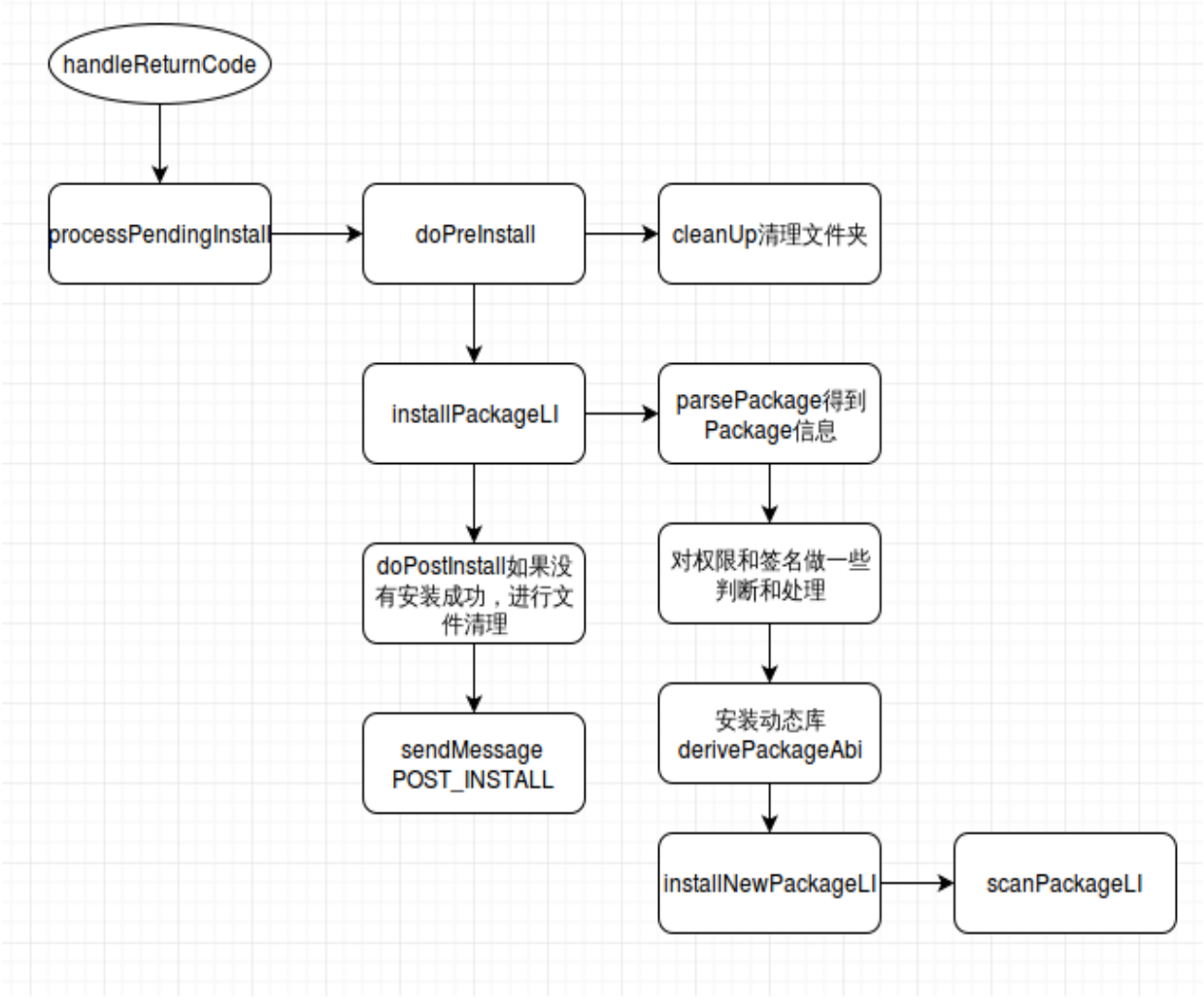
通过 adb install 安装应用

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
...