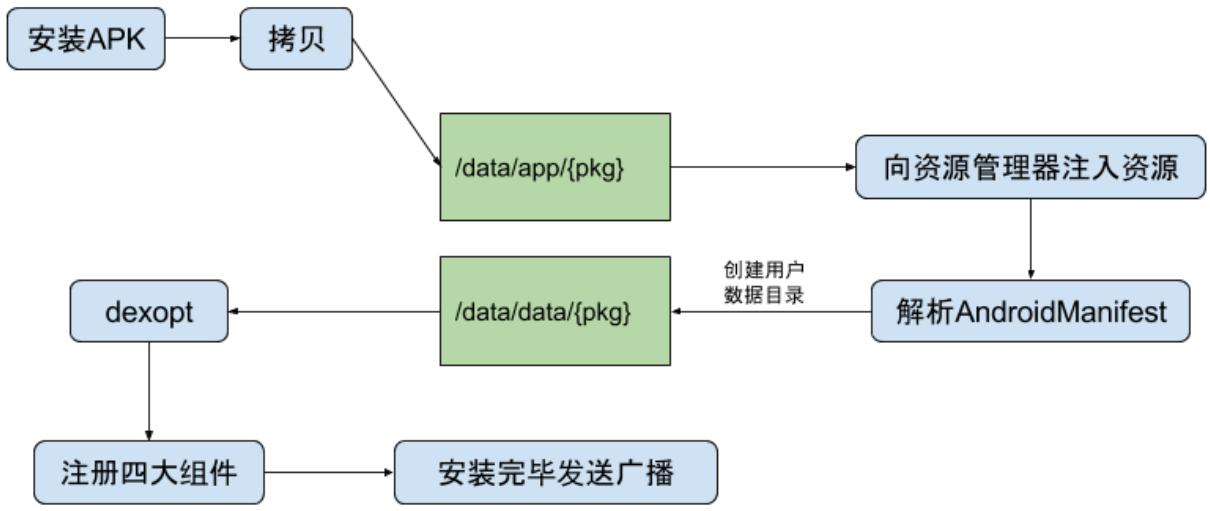
面试宝典-Framework

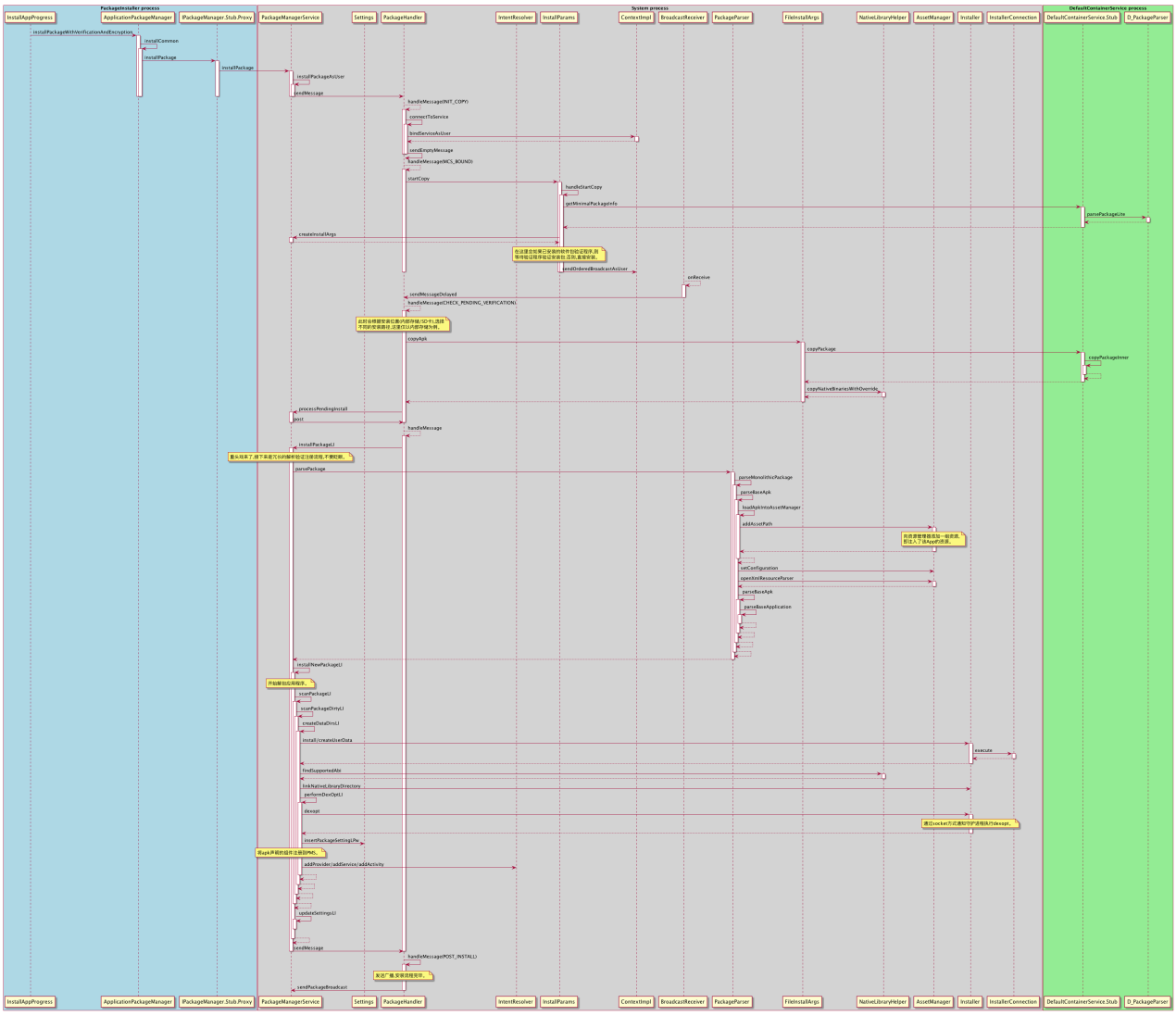
安装

安装过程

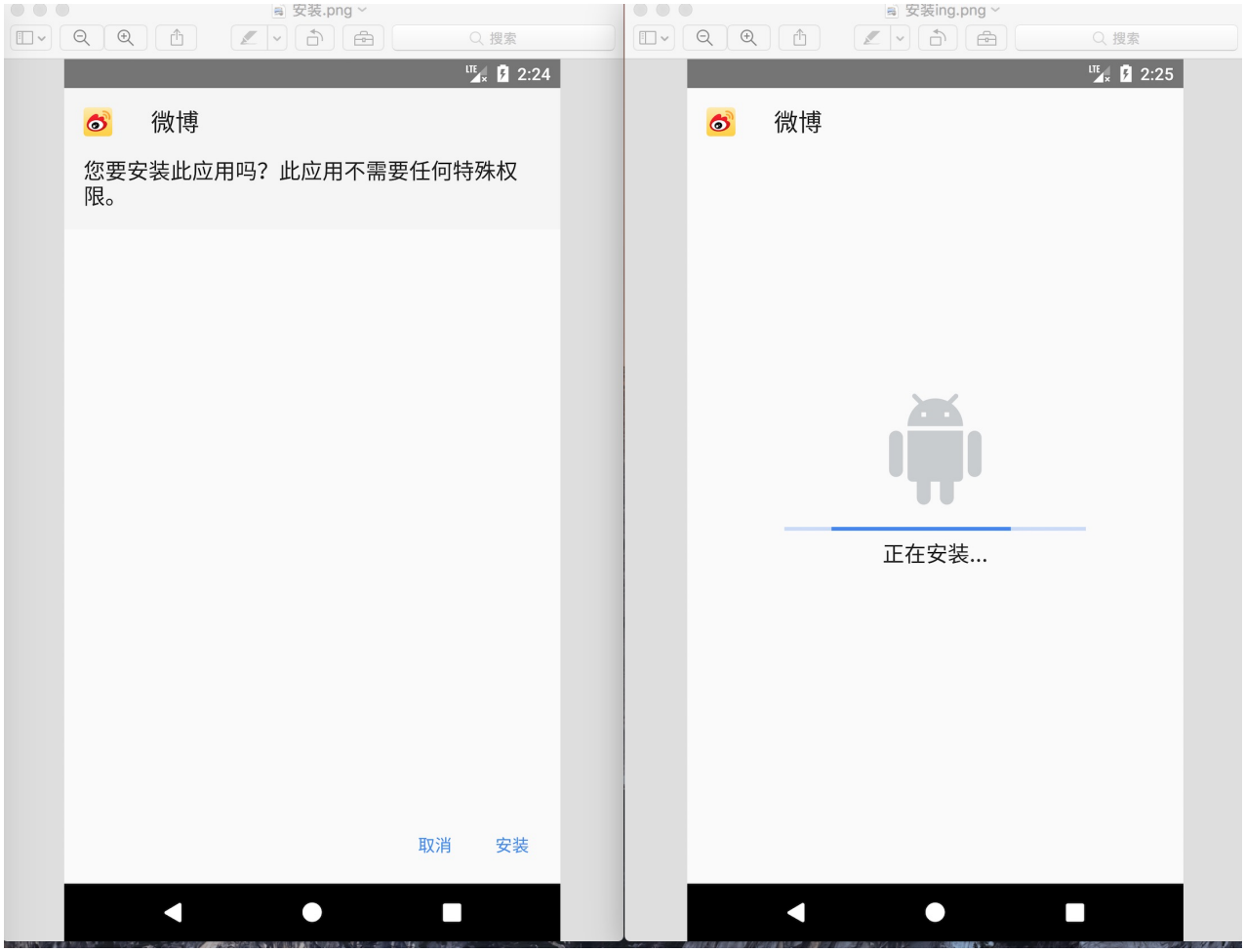
安装大致流程图：



安装过程：复制 apk 安装包到 /data/app 目录下，解压并扫描安装包，向资源管理器注入 apk 资源，解析 AndroidManifest 文件，并在 /data/data 目录下创建对应的应用数据目录，然后针对 dalvik/art 环境优化 dex 文件，保存到 dalvik-cache 目录，将 AndroidManifest 文件解析出的组件、权限注册到 PackageManagerService，完成后发送广播。



1. 开始：InstallAppProcess 是一个 activity，右边的图就是 InstallAppProcess。



在 InstallAppProcess 的 oncreate() 方法中会调用 initView() 方法，initView() 方法会根据不同的 scheme 进行不同模式的安装。

- scheme 为 package 调用的是 pm.installExistingPackage(mAppInfo.packageName) 进行安装
- scheme 为其他（file）的调用的 pm.installPackageWithVerificationAndEncryption(mPackageURI, observer, installFlags, installerPackageName, verificationParams, null); 进行安装。

pm 就是 PackageManager。

2. 拷贝：PackageManager 的实现类 ApplicationPackageManager 会通过 AIDL 调用服务端的 PackageManagerService 的 installPackage 方法进行安装。

   先完成第一步拷贝。

   （1）PackageManagerService 会发送 INIT_COPY 消息，PackageHandler 会接收消息并处理。

   （2）PackageHandler 处理 INIT_COPY 消息，会调用 connectToService() 方法绑定 DefaultContainerService 服务。

   （3）DefaultContainerService 服务是干什么的?

   （4）服务连接成功后，会发送 MCS_BOUND 消息，PackageHandler 会接收消息并处理。

   （5）PackageHandler 处理 MCS_BOUND 消息，调用 HandlerParams 的 startCopy 方法处理安装请求。

（6）HandlerParams 的 startCopy() 方法中会调用 handleStartCopy() 方法进行拷贝操作，如果失败，会进行重试 4 次，如果 4 次还失败，则调用 handleServiceError() 返回失败。如果成功，会调用 handleStartCopy 返回状态码。

（7）startCopy() 方法调用 HandlerParams 子类的 InstallParams 的 handleStartCopy()来完成拷贝工作的，在 handleStartCopy() 方法中如果内存不足，则会调用 Installer.freeCache() 释放存储空间；接下来调用InstallArgs的copyApk方法进行包拷贝。InstallArgs的copyApk 会创建目录文件，调用 imc.copyPackage() 方法进行代码拷贝，还有Native 代码（也就是 so 进行）拷贝。

（8）imc 其实是一个远程的代理，实际的调用方是 DefaultContainerService 的成员变量 mBinder。而 DefaultContainerService 的 mBinder 的 copyPackage() 方法会调用 DefaultContainerService 的 copyFile() 方法，copyFile() 方法调用了 Stream.copy() 方法完成拷贝。

（9）拷贝操作成功后，在 HandlerParams 的 startCopy() 方法中会调用 handleStartCopy 返回状态码，handleStartCopy() 方法里面调用的 processPendingInstall() 方法，processPendingInstall() 方法中会向 mHandler 发送一个 Runnable 对象，如果状态是 PackageManager.INSTALL_SUCCESSED，则分为 3 个阶段：

- 预安装阶段：检查当前安装包的状态以及确保SDCARD的挂载，并返回状态信息。在安装前确保安装环境的可靠。
- 安装阶段：对mInstallLock加锁，表明同时只能由一个安装包进行安装，然后调用 installPackageLI方法完成具体的安装操作。
- 安装收尾阶段：检查状态，如果安装失败，删除相关目录文件。

3. 解析验证注册流程：

（1）解析验证从 PackageManagerService 的installPackageLI() 方法开始。

- 第一步：PackageManagerService 的installPackageLI() 方法中调用了 PackageParser.Package 的 parsePackage() 方法进行解析 APK，解析的结果会记录在 PackageParser.Package 中。
- 第二步：判断是新安装还是升级安装，调用 shouldCheckUpgradeKeySetLP() 方法检查密钥集合是否一致。
- 第三步：检查权限
- 第四步：根据不同的安装标志，来进行操作，分为三种情况
  - 移动操作：
  - 非锁定安装且没有安装在SD卡上：**新安装走这里**，这里面主要做两个操作：第①步是**so拷贝**，第②步是**进行dex优化**，第③步机械性dex2oat操作，将dex文件转化为oat。
  - 如果上面两个条件都不满足，则什么也不做
- 第五步：重命名安装：将/data/app/vmdl{安装会话}.tmp重命名为/data/apppp/包名-suffix,suffix为1、2...
- 第六步：开始intent filter验证
- 第七步：这里根据不同的安装方式进行不同的方式，主要有两种情况
  - 覆盖安装即更新安装：调用replacePackageLI方法进行覆盖安装
  - 首次安装：调用installNewPackageLI方法进行首次安装
- 第八步：安装收尾，调用PackageSetting的queryInstalledUsers设置安装用户

（2）Android 安装一个 APK 的时候首先会解析 APK，而解析 APK 则需要用到一个工具类，这个工具类就是 PackageParser。PackageParse 类主要用来解析手机上的 APK 文件（支持 Single APK 和 MultipleAPK），解析一个 APK 主要是分类两个步骤：1.将APK 解析成 Package：即解析 APK 文件为 Package 对象的规程。2.将 Package 转化为 PackageInfo：即由 Package 对象生成 Package 对象生成 PackageInfo 的过程。

（3）PackageParser.Package 的 parsePackage() 方法会调用 parseBaseApk() 方法，解析一个 apk 并生成一个 Package 对象。

（4）在 PackageParser 的 parseBaseApk() 方法中会详细解析 AndroidManifest 下面的每一个节点（如 application、overlay、key_sets、permission-group、permission、permission-tree、uses-permission 等等节点）。并且也会调用 loadApkIntoAssetManager() 方法将 Android 系统中安装包路径和 AssetManager 关联。

（5）解析完成后回到PackageManagerService 的installPackageLI() 方法中会调用 installNewPackageLI方法进行首次安装。

4. 装载：

（1）PackageManagerService 的 installNewPackageLI() 方法会调用 scanPackageLI() 方法进行安装 APK，并调用 updateSettingsLI() 方法更新 Settings。

（2）PackageManagerService 的 scanPackageLI() 方法调用了 scanPackageDirtyLI() 方法安装 APK。

- 第一步：检查代码路径是否存在，如果不存在则抛出异常。
- 第二步：初始化 PackageParser.Package 的 pkg 的一些属性，主要是 applicationInfo 信息。
- 第三步：设置 ResolverActivity 信息。
- 第四步：如果是系统应用程序，则变更 ResolverActivity 信息。
- 第五步：如果是更新安装（即只安装已经存在的包），检查它的 PackageSetting 信息，如果路径不一致，则抛出异常。
- 第六步：初始化包的安装目录（代码目录与资源目录）。
- 第七步：检查是否需要重命名。
- 第八步：检查所有共享库：并且映射到真实的路径。
- 第九步：如果是升级更新安装，则检查升级更新包的签名，如果是新安装，则验证签名。
- 第十步：检查安装包中的 provider 是不是和现在系统中已经存在包的 provider 冲突
- 第十一步：检查当前安装包对其他包的所拥有的权限（比如系统应用）
- 第十二步：创建 data 目录，并且重新调整 uid，调用 createDataDirsLI 进行包的安装。其中 framework-res.apk 比较特殊，它的 data 目录位于 /data/system/，其他 APK 的 data 目录都位于 /data/data/packageName 下。
- 第十三步：设置 Native Library 的路径。即 so 文件目录。
- 第十四步：创建用户数据，主要是调用 Installer 的 createUserData 方法来实现的。
- 第十五步：对包进行 dex 优化，主要是调用 performDexOpt 方法来进行，最终还是要调用 Install 的 dexopt 函数。
- 第十六步：如果该包已经存在了，需要杀死该进程。

- 第十七步：将一个安装包的内容从 pkg 里面映射到 PackageManagerService 里面。这样一个安装包中的所有组件信息里面主要分为：
  - 1.解析 provider，并映射到 PackageManagerService 的变量 xx 里面
  - 2.解析 service，并映射到 PackageManagerService 的变量 xx 里面
  - 3.解析 receive，并映射到 PackageManagerService 的变量 xx 里面
  - 4.解析 activity，并映射到 PackageManagerService 的变量 xx 里面
  - 5.解析 GroupPermission 与 Permission，并映射到 PackageManagerService 的变量 xx 里面
  - 6.解析 instrumentation，并映射到 PackageManagerService 的变量 xx 里面。

（3）PackageManagerService 的 createDataDirsLI() 方法调用 Installer 的 install 方法进行了包的安装。

（4）

5. 完成：在 PackageManagerService. 的 startCopy() 方法的最后，调用了 handleReturnCode() 方法发送了 POST_INSTALL 的消息到 PackageHandler 通知安装结束。
   - PackageHandler 会发送 ACTION_PACKAGE_ADDED 广播，如果是更新，还会发送 ACTION_PACKAGE_REPLACED 广播，
   - 接着还会强制调用 gc ，触发 JVM 进行垃圾回收操作，删除旧的安装信息，
   - 最后会回调 args.observer.packageInstalled 方法，告诉 PackageInstaller 安装结果，从而实现了安装回调到 UI 层。