**Android Framework 框架系列之 PackageManagerService**

本篇文章主要介绍 Android 开发中的 **PackageManagerService** 部分知识点

前言 SystemServer启动PMS  
一、PackageManagerService 简介  
二、PMS.main入口  
三、PMS 主要作用  
四、PMS 涉及到的模块  
五、PMS 启动过程  
六、PMS 权限管理  
七、PMS 安装 Jar包 、apk  
八、PMS 构造函数  
九、PMS构造函数分析

**本文介绍PackageManagerService启动流程**

相关源码:

frameworks/base/services/core/java/com/android/server/pm/PackageManagerService.java

frameworks/base/services/core/java/com/android/server/pm/PackageInstallerService.java

frameworks/base/services/core/java/com/android/server/pm/Settings.java

frameworks/base/services/core/java/com/android/server/pm/Installer.java

frameworks/base/services/core/java/com/android/server/SystemConfig.java

frameworks/base/core/java/android/content/pm/PackageManager.java

frameworks/base/core/java/android/content/pm/IPackageManager.aidl

frameworks/base/core/java/android/content/pm/PackageParser.java

frameworks/base/core/java/com/android/internal/os/InstallerConnection.java

frameworks/base/cmds/pm/src/com/android/commands/pm/Pm.java

**前言 SystemServer启动PMS**

SystemServer

**本文主要讲解PMS 启动流程，SystemServer 详细可以查看如下文章**

[SystemServer 启动流程](https://mp.weixin.qq.com/s/C4WF6rhx8PeK_pVsOyO1gQ)

**一、PackageManagerService 简介**

**PackageManagerService** 住用来跟踪管理系统所有的apk，AMS中有两个重要的锁( mInstallLock 锁, mPackages锁 )。

**mInstallLock**  
用来保护所有安装apk的访问权限，此操作通常涉及繁重的磁盘数据读写等，并且是单线程操作，故有时候会处理很慢。  
此锁永远不会再已经持有 **mPackages** 锁的情况下获得。  
反过来，在已经持有 **mInstallLock**锁 的情况下，立即获取 **mPackages** 是安全的。

许多内部方法依靠调用者来保存适当的锁，并且此合约通过方法名称后缀表示：

* 1.fooLI()

调用者必须持有**mInstallLock** 锁。

* 2.fooLIF()

调用者必须持有**mInstallLock** 锁 并且被修改的包必须被冻结

**mPackages**  
用来解析内存中所有apk的package 信息及相关状态，其实系统中划分最精细，竞争力最强的锁之一。

* 1.fooLPr()

调用者必须持有**mPackages** read 锁。

* 1. fooLPw()

调用者必须持有**mPackages** write 锁。

**PackageManagerService 继承实现关系**

PMS 继承关系图如下：

PackageManagerService 继承实现关系

PackageSender 接口类

**二、PMS.main入口**

PackageManagerService.main过程主要是创建PMS服务，并注册到ServiceManager大管家

PMS 是在 Systemserver.java 中的startBootstrapServices 方法中启动并调用的。

Systemserver startBootstrapServices

public static PackageManagerService main(Context context, Installer installer,

boolean factoryTest, boolean onlyCore) {

// Self-check for initial settings.

PackageManagerServiceCompilerMapping.checkProperties();

PackageManagerService m = new PackageManagerService(context, installer,

factoryTest, onlyCore);

m.enableSystemUserPackages();

ServiceManager.addService("package", m);

final PackageManagerNative pmn = m.new PackageManagerNative();

ServiceManager.addService("package\_native", pmn);

return m;

}

**三、PMS 主要作用**

**PMS 主要作用如下：**

* 1.管理系统 jar包和apk ，负责系统权限
* 2.负责程序的安装、卸载、更新、解析
* 3.对于其他应用和服务提供安装、卸载服务

**四、PMS 涉及到的模块**

PMS 涉及到如下模块

PMS 涉及到的模块

**五、PMS 启动过程**

**PMS 启动大致过程如下**

* 1. 和installed 进行连接，进行安装、卸载操作。
  2. 创建PackageHandler 线程，处理外部安装、卸载请求。
  3. 处理系统权限相关的文件/system/etc/permission/\*.xml。
  4. 扫描安装jar包 和APK，并获取到安装包的信息。

**六、PMS 权限管理**

PMS 权限管理 大致如下：

PMS 权限管理

**七、PMS 安装 Jar包 、apk**

PMS 安装 Jar包 、apk

**八、PMS 构造函数**

PMS 沟通方法一

PMS构造函数

扫描系统中的apk

只允许开机向导有高优先级

只允许开机向导高优先级

checkDefaultBrowser 检查默认浏览器

checkDefaultBrowser 检查默认浏览器

**九、PMS构造函数分析**

创建PMS对象的过程，就是执行PMS的构造函数，PMS构造函数比较长，我们把这个过程分成几个阶段

* BOOT\_PROGRESS\_PMS\_START,
* BOOT\_PROGRESS\_PMS\_SYSTEM\_SCAN\_START,
* BOOT\_PROGRESS\_PMS\_DATA\_SCAN\_START,
* BOOT\_PROGRESS\_PMS\_SCAN\_END,
* BOOT\_PROGRESS\_PMS\_READY,

**BOOT\_PROGRESS\_PMS\_START 主要工作**

* 构造DisplayMetrics类：描述界面显示，尺寸，分辨率，密度。构造完后并获取默认的信息保存到变量mMetrics中。
* 构造Settings类：这个是Android的全局管理者，用于协助PMS保存所有的安装包信息
* 保存Installer对象
* 获取系统配置信息：SystemConfig构造函数中会通过readPermissions()解析指定目录下的所有xml文件,然后把这些信息保存到systemConfig中，涉及的目录有如下：  
  /system/etc/sysconfig  
  /system/etc/permissions  
  /oem/etc/sysconfig  
  /oem/etc/permissions
* 创建名为PackageManager的handler线程，建立PackageHandler消息循环，用于处理外部的安装请求等消息
* 创建data下的各种目录，比如data/app, data/app-private等。
* 创建用户管理服务UserManagerService
* 把systemConfig关于xml中的标签所指的动态库保存到mSharedLibraries  
  Settings.readLPw扫描解析packages.xml和packages-backup.xml

**PMS\_SYSTEM\_SCAN\_START 主要工作**

* 首先将BOOTCLASSPATH，SYSTEMSERVERCLASSPATH这两个环境变量下的路径加入到不需要dex优化集合alreadyDexOpted中

**SYSTEMSERVERCLASSPATH：**  
主要包括/system/framework目录下services.jar，ethernet-service.jar，wifi-service.jar这3个文件。

**BOOTCLASSPATH：**  
该环境变量内容较多，不同ROM可能有所不同，常见内容包含/system/framework目录下的framework.jar，ext.jar，core-libart.jar，telephony-common.jar，ims-common.jar，core-junit.jar等文件。

* 获取共享库mSharedLibraries，判断是否需要dex优化，如果需要则进行dex优化，并加入到alreadyDexOpted列表中
* 添加framework-res.apk、core-libart.jar两个文件添加到已优化集合alreadyDexOpted中
* 将framework目录下，其他的apk或者jar，进行dex优化并加入已优化集合alreadyDexOpted中
* scanDirLI(): 扫描指定目录下的apk文件，最终调用PackageParser.parseBaseApk来完成AndroidManifest.xml文件的解析，生成Application, activity,service,broadcast, provider等信息
* 删除系统不存在的包 removePackageLI
* 清理安装失败的包 cleanupInstallFailedPackage
* 删除临时文件 deleteTempPackageFiles
* 移除不相干包中的所有共享userID