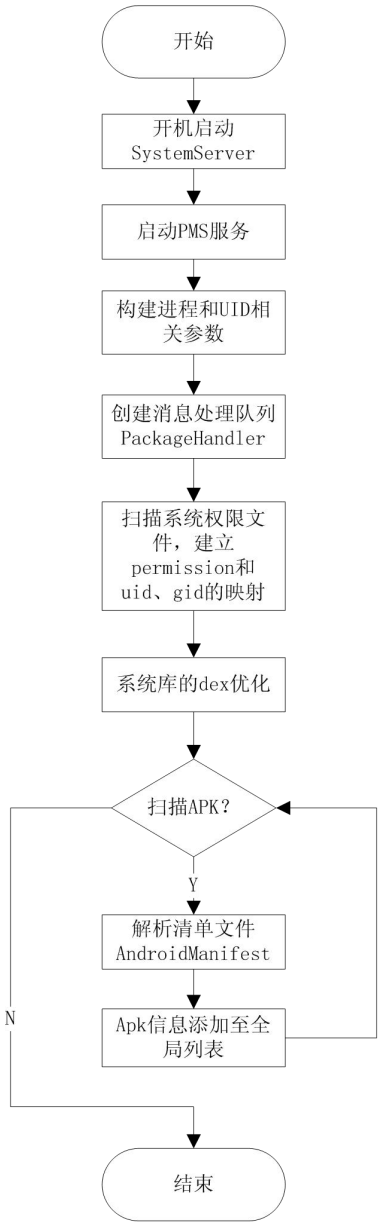
目录

一、PMS(PackageManagerService)启动分析

（1）PMS 介绍

PackageManagerService 是 Android 系统中最常用的服务之一。它负责系统中 Package 的管理，应用程序的安装、卸载、信息查询等。

（2）PMS 启动流程图



（3）PMS 核心代码分析

1、SystemServer 启动 PMS 服务

SystemServer 进程是 Zygote 孵化出的第一个进程，该进程主要的工作是启动 android 系统服务进程，其中包括 PackageManagerService 服务。

```
private void startBootstrapServices() {
    ...
```

```
    mPackageManagerService = PackageManagerService.main(mSystemContext, installer,
            mFactoryTestMode != FactoryTest.FACTORY_TEST_OFF, mOnlyCore);
    mFirstBoot = mPackageManagerService.isFirstBoot();
    mPackageManager = mSystemContext.getPackageManager();
    ...
}
```

2、PMS 的 main 方法实现

main 函数很简单，只有短短几行代码，执行时间却较长（正常开机，创建 PMS 对象需 6-10s），主要原因是 PKMS 在其构造函数中做了很多“体力活”，这也是 Android 启动速度慢的主要原因之一。

```
public static PackageManagerService main(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
    PackageManagerServiceCompilerMapping.checkProperties();
    //实例化 PMS 对象
    PackageManagerService m = new PackageManagerService(context, installer,
            factoryTest, onlyCore);
    //enable 和 disable 某些黑名单应用
    m.enableSystemUserPackages();
    CarrierAppUtils.disableCarrierAppsUntilPrivileged(context.getOpPackageName(), m,
            UserHandle.USER_SYSTEM);
    //将 PMS 注册到 ServiceManager
    ServiceManager.addService("package", m);
    return m;
}
```

3、实例化 PMS--构建进程和 UID 关系

UID 为用户 ID 的缩写，GID 为用户组 ID 的缩写，这两个概念均与 Linux 系统中进程的权限管理有关。一般来说，每一个进程都有一个对应的 UID，表示该进程属于哪个用户，不同用户有不同权限。一个进程也可分属不同的用户组，每个用户组都有对应的权限。

```
public PackageManagerService(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
    ...
    mFactoryTest = factoryTest;//false
    mOnlyCore = onlyCore;//false
    mMetrics = new DisplayMetrics();
    mSettings = new Settings(mPackages);

    mSettings.addSharedUserLPw("android.uid.system", Process.SYSTEM_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//1000
    mSettings.addSharedUserLPw("android.uid.phone", RADIO_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//1001
    mSettings.addSharedUserLPw("android.uid.log", LOG_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//1007
```

```
    mSettings.addSharedUserLPw("android.uid.nfc", NFC_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//1027
    mSettings.addSharedUserLPw("android.uid.bluetooth", BLUETOOTH_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//1002
    mSettings.addSharedUserLPw("android.uid.shell", SHELL_UID,
            ApplicationInfo.FLAG_SYSTEM, ApplicationInfo.PRIVATE_FLAG_PRIVILEGED);//2000
}
```

```
SharedUserSetting addSharedUserLPw(String name, int uid, int pkgFlags, int pkgPrivateFlags) {
    SharedUserSetting s = mSharedUsers.get(name);
    ...
    s = new SharedUserSetting(name, pkgFlags, pkgPrivateFlags);
    s.userId = uid;
    if (addUserIdLPw(uid, s, name)) {
        mSharedUsers.put(name, s);
        return s;
    }
    return null;
}
```

例如：AndroidManifgest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.com.apkinstall_test"
    android:sharedUserId="android.uid.system">
```

普通 APK，

```
P100AN:/ # top | grep "android.com.apkinstall_test"
 2691 u0_a96   20   0   0% S    25 1039180K  68392K  fg android.com.apkinstall_test
^Z[3] + Stopped              top | grep "android.com.apkinstall_test"
P100AN:/ # id u0_a96
uid=10096(u0_a96) gid=10096(u0_a96) groups=10096(u0_a96), context=u:r:su:s0
```

系统 APK，

```
P100AN:/ $ top | grep "android.com.apkinstall_test"
 2391 system   20   0   0% S    24 1027552K  58584K  bg android.com.apkinstall_test
^Z[2] + Stopped              top | grep "android.com.apkinstall_test"
P100AN:/ $ id system
uid=1000(system) gid=1000(system) groups=1000(system), context=u:r:shell:s0
```

在该标签中，声明了一个 android:sharedUserId 的属性，其值为“android.uid.system”。sharedUserId 和 UID 有关，它的作用是：

- 两个或者多个声明了同一种 sharedUserid 的 APK 可共享彼此的数据；
- 通过声明特定的 sharedUserId,该 APK 所在的进程将被赋予指定 UID。

4、实例化 PMS--扫描系统权限文件

扫描系统目录下与系统权限相关的 xml 文件，将其存放到 PKM 中。

```
public PackageManagerService(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
```

```
    ....
    SystemConfig systemConfig = SystemConfig.getInstance();//获取系统配置信息
    mGlobalGids = systemConfig.getGlobalGids();
    mSystemPermissions = systemConfig.getSystemPermissions();
    mAvailableFeatures = systemConfig.getAvailableFeatures();
    ....
}
```

```
SystemConfig() {
     ...
    /**
     * 依次扫描如下路径权限文件
     * etc/sysconfig     etc/permissions
     * odm/etc/sysconfig     odm/etc/permissions
     * oem/etc/sysconfig     oem/etc/permissions
     * vendor/etc/sysconfig     vendor/etc/permissions
     */
    readPermissions(Environment.buildPath(
            Environment.getRootDirectory(), "etc", "sysconfig"), ALLOW_ALL);
    readPermissions(Environment.buildPath(
            Environment.getRootDirectory(), "etc", "permissions"), ALLOW_ALL);
   ...
}
```

例如：etc/permissions/platform.xml，部分代码如下：

```
37    <permission name="android.permission.BLUETOOTH_ADMIN" >
38        <group gid="net_bt_admin" />
39    </permission>
40
41    <permission name="android.permission.BLUETOOTH" >
42        <group gid="net_bt" />
43    </permission>
44
45    <permission name="android.permission.BLUETOOTH_STACK" >
46        <group gid="net_bt_stack" />
47        <group gid="wakelock" />
48    </permission>
49
50    <permission name="android.permission.NET_TUNNELING" >
51        <group gid="vpn" />
52    </permission>
53
54    <permission name="android.permission.INTERNET" >
55        <group gid="inet" />
56    </permission>
```

5、实例化 PMS--创建消息处理队列

创建一个 ThreadHandler 对象，实际就是创建一个带消息队列循环处理的线程，该线程的工作是：程序的安装和卸载等。

```
public PackageManagerService(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
```

```
    ...
    mHandlerThread = new ServiceThread(TAG,
            Process.THREAD_PRIORITY_BACKGROUND, true /*allowlo*/);
    mHandlerThread.start();
    mHandler = new PackageHandler(mHandlerThread.getLooper());
    mProcessLoggingHandler = new ProcessLoggingHandler();
    Watchdog.getInstance().addThread(mHandler, WATCHDOG_TIMEOUT);
    ...
}
```

6、实例化 PMS--扫描 APK

阶段的工作主要是扫描系统中的 APK，由于需要逐个扫描 apk 文件，因此手机上安装的程序越多，PKM 的工作量越大，系统启动速度越慢，也就是开机时间越长。

```
public PackageManagerService(Context context, Installer installer,
        boolean factoryTest, boolean onlyCore) {
    ...
```

```
    /**
     * 依次掃描如下等路径
     * /vendor/overlay/
     * /system/framework/
     * /system/priv-app/
     * /system/app/
     * /vendor/priv-app/
     * /system/vendor/app/
     */
    File frameworkDir = new File(Environment.getRootDirectory(), "framework");
    File vendorOverlayDir = new File(VENDOR_OVERLAY_DIR);
    scanDirTracedLI(vendorOverlayDir, mDefParseFlags
            | PackageParser.PARSE_IS_SYSTEM
            | PackageParser.PARSE_IS_SYSTEM_DIR
            | PackageParser.PARSE_TRUSTED_OVERLAY, scanFlags |
SCAN_TRUSTED_OVERLAY, 0);
    ...
    final File privilegedAppDir = new File(Environment.getRootDirectory(), "priv-app");
    scanDirTracedLI(privilegedAppDir, mDefParseFlags
            | PackageParser.PARSE_IS_SYSTEM
            | PackageParser.PARSE_IS_SYSTEM_DIR
            | PackageParser.PARSE_IS_PRIVILEGED, scanFlags, 0);
    final File systemAppDir = new File(Environment.getRootDirectory(), "app");
    scanDirTracedLI(systemAppDir, mDefParseFlags
            | PackageParser.PARSE_IS_SYSTEM
            | PackageParser.PARSE_IS_SYSTEM_DIR, scanFlags, 0);
        ...
}
```

扫描系统的 APK，每一个 APK 对应一个 Package 对象，主要是扫描 APK 的 AndroidManifest.xml，解析 application 标签及其子标签 actvity、service、recever 等，解析后将它们保存到 Package 对应的数据结构中

```
private Package parseBaseApkCommon(Package pkg, Set<String> acceptedTags, Resources res,
        XmlResourceParser parser, int flags, String[] outError) throws XmlPullParserException,
        IOException {
while ((type = parser.next()) != XmlPullParser.END_DOCUMENT
        && (type != XmlPullParser.END_TAG || parser.getDepth() > outerDepth)) {
    ...
    String tagName = parser.getName();
    ...
    if (tagName.equals(TAG_APPLICATION)) {//application
        ...
```

```
    } else if (tagName.equals(TAG_OVERLAY)) {//overlay
        ...
    } else if (tagName.equals(TAG_KEY_SETS)) {//key-sets
        ...
    } else if (tagName.equals(TAG_PERMISSION_GROUP)) {//permission-group
        ...
    } else if (tagName.equals(TAG_PERMISSION)) {//permission
        ...
    } else if (tagName.equals(TAG_PERMISSION_TREE)) {//permission-tree
        ...
    } else if (tagName.equals(TAG_USES_PERMISSION)) {//uses-permission

        ...
    }
    ...
}
```

依次解析 activity，receiver，service，provider 等标签，其中可以发现，receiver 被当成 activity 来解析了，PKM 通过 PackageParser 类解析后的四大组件保存到对应数据结构中，也就是存放到 PackageParser 的 activities，receivers，providers，services 对象中

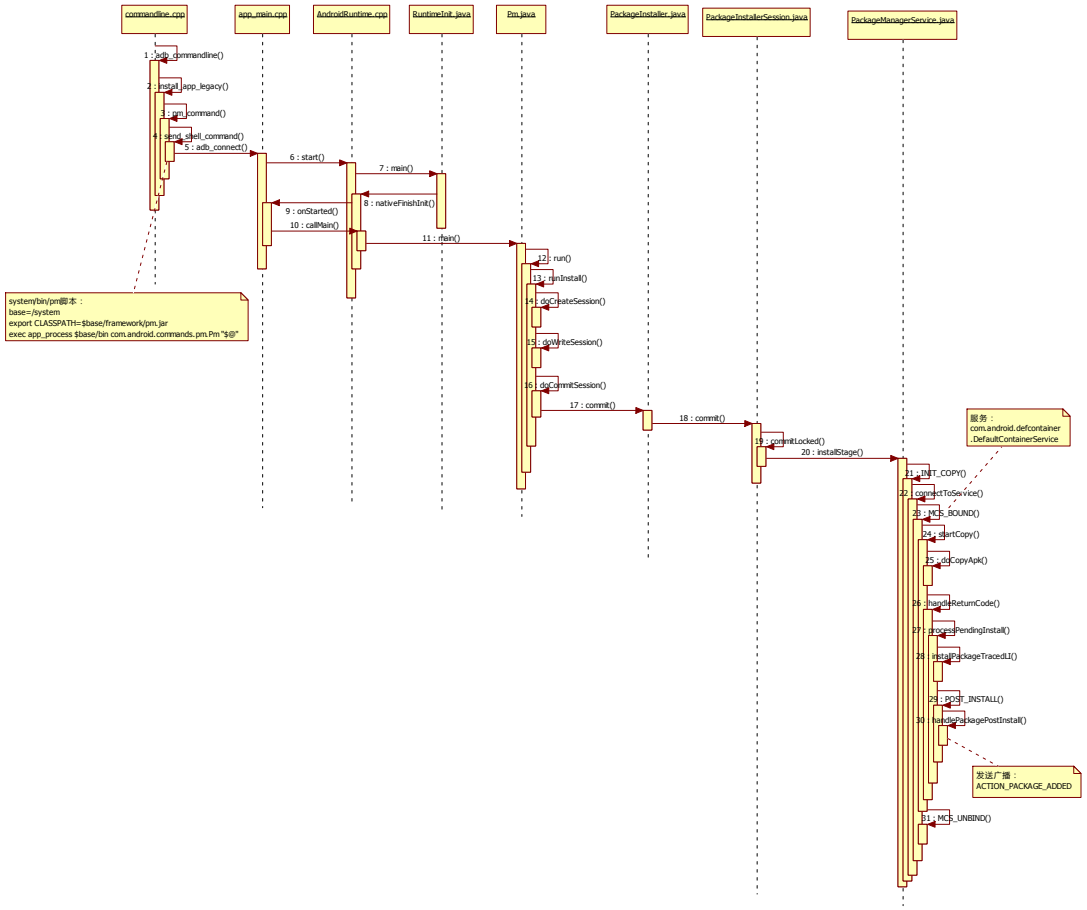
```
private PackageParser.Package scanPackageDirtyLI(PackageParser.Package pkg,
        final int policyFlags, final int scanFlags, long currentTime, UserHandle user)
        throws PackageManagerException {
    ...
    int N = pkg.providers.size();
    for (i=0; i<N; i++) {
        ...
    }
    N = pkg.services.size();
    for (i=0; i<N; i++) {
        ...
    }
    N = pkg.receivers.size();
    for (i=0; i<N; i++) {
        PackageParser.Activity a = pkg.receivers.get(i);
        a.info.processName = fixProcessName(pkg.applicationInfo.processName,
                a.info.processName, pkg.applicationInfo.uid);
        mReceivers.addActivity(a, "receiver");
        ...
    }
    N = pkg.activities.size();
    for (i=0; i<N; i++) {
```

```
        ...
    }
    ...
}
```

二、应用安装分析

（1）adb install 安装

1、流程图



2、代码分析

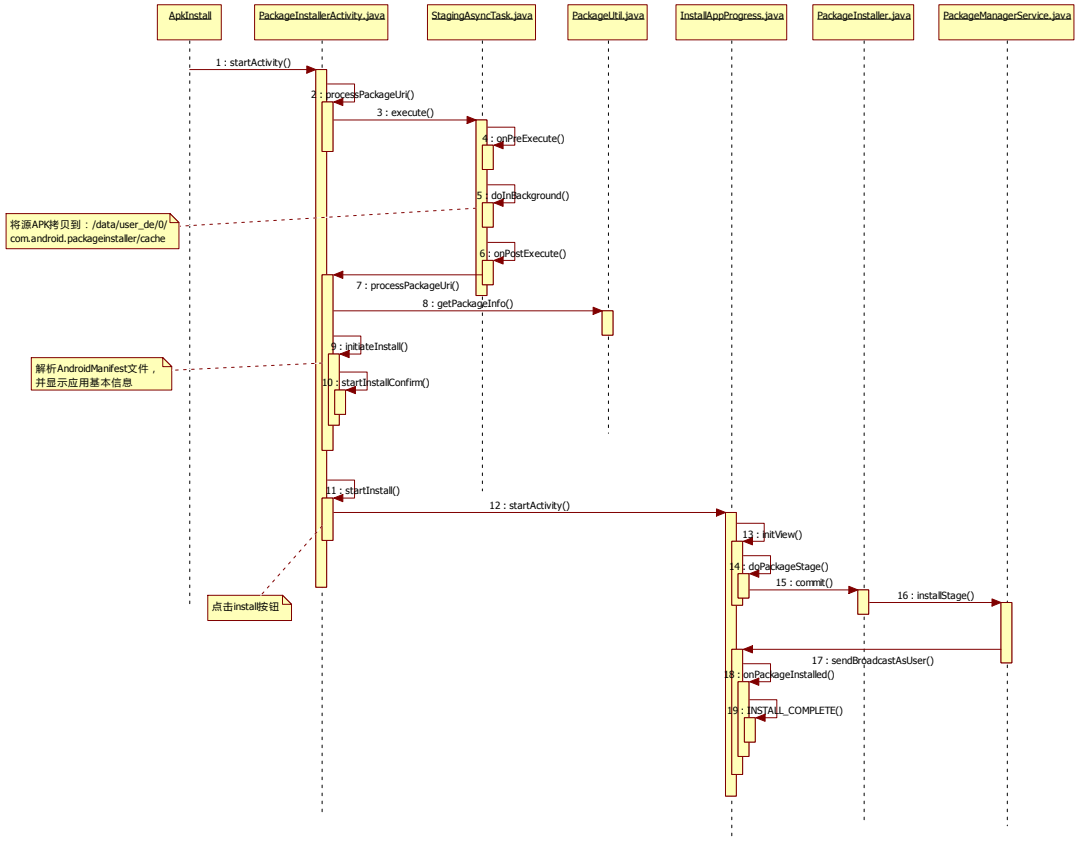
```
class PackageHandler extends Handler {
    private boolean mBound = false;
    final ArrayList<HandlerParams> mPendingInstalls =
        new ArrayList<HandlerParams>();

    private boolean connectToService() {
        Intent service = new Intent().setComponent(DEFAULT_CONTAINER_COMPONENT);
        Process.setThreadPriority(Process.THREAD_PRIORITY_DEFAULT);
        if (mContext.bindServiceAsUser(service, mDefContainerConn,
                Context.BIND_AUTO_CREATE, UserHandle.SYSTEM)) {
```

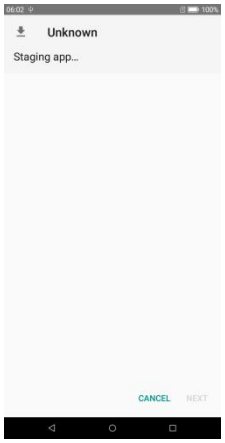
```
            Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
            mBound = true;
            final long DEFCONTAINER_CHECK = 1 * 1000;
            final Message msg = mHandler.obtainMessage(MCS_CHECK);
            mHandler.sendMessageDelayed(msg, DEFCONTAINER_CHECK);
            return true;
        }
        Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
        return false;
    }

    private void disconnectService() {
        mContainerService = null;
        mBound = false;
        mServiceConnected = false;
        if (DEBUG_SD_INSTALL) Log.i(TAG, "disconnectService: " + mServiceConnected);
        Process.setThreadPriority(Process.THREAD_PRIORITY_DEFAULT);
        mContext.unbindService(mDefContainerConn);
        Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND);
    }
    public void handleMessage(Message msg) {
            doHandleMessage(msg);
    }

    void doHandleMessage(Message msg) {
        switch (msg.what) {
            case INIT_COPY: {
                HandlerParams params = (HandlerParams) msg.obj;
                int idx = mPendingInstalls.size();
                if (!mBound) {
                    if (!connectToService()) {
                        ...
                    }
                }
                ...
            }
            case MCS_BOUND: {
                if (mContainerService == null) {
                    ...
                } else if (mPendingInstalls.size() > 0) {
                    HandlerParams params = mPendingInstalls.get(0);
                    if (params != null) {
                        if (params.startCopy()) {
                            if (mPendingInstalls.size() > 0) {
```

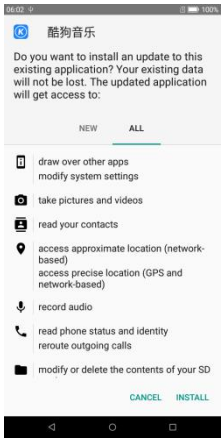
```
                                mPendingInstalls.remove(0);
                            }
                            if (mPendingInstalls.size() == 0) {
                                if (mBound) {
                                    removeMessages(MCS_UNBIND);
                                    Message ubmsg = obtainMessage(MCS_UNBIND);
                                    sendMessageDelayed(ubmsg, 10000);
                                }
                            }
                        }
                    }
                }
            }
            case MCS_CHECK:
            case MCS_RECONNECT:
            case MCS_UNBIND: {
                if (mPendingInstalls.size() == 0 && mPendingVerification.size() == 0) {
                    if (mBound) {
                        disconnectService();
                    }
                }
            }
            case MCS_GIVE_UP:
            case SEND_PENDING_BROADCAST:
            case START_CLEANING_PACKAGE:
            ...
        }
    }
```

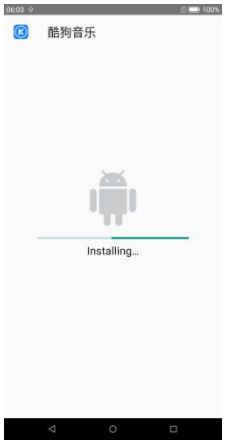
（2）下载管理器（PackageInstaller）安装

```
private void installApk(){
    /**
     * 对于 Android N 的应用，API 禁止向您的应用外公开 file://URI。
     */
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    String path = Environment.getExternalStorageDirectory().getAbsolutePath() +
"/LZQ/com.kugou.android.apk";
    Uri contentUri = FileProvider.getUriForFile(mContext, BuildConfig.APPLICATION_ID +
".fileProvider", new File(path));
    intent.setDataAndType(contentUri, "application/vnd.android.package-archive");
    mContext.startActivity(intent);
}
```


ApkInstall
PackageInstallerActivity.java
StagingAsyncTask.java
PackageUtil.java
InstallAppProgress.java
PackageInstaller.java
PackageManagerService.java
1 : startActivity()
2 : processPackageUri()
3 : execute()
4 : onPreExecute()
5 : doInBackground()
将源APK拷贝到：/data/user_de/0/com.android.packageinstaller/cache
6 : onPostExecute()
7 : processPackageUri()
8 : getPackageInfo()
9 : initiateInstall()
解析AndroidManifest文件，并显示应用基本信息
10 : startInstallConfirm()
11 : startInstall()
12 : startActivity()
13 : initView()
14 : doPackageStage()
15 : commit()
16 : installStage()
17 : sendBroadcastAsUser()
18 : onPackageInstalled()
19 : INSTALL_COMPLETE()
点击install按钮


Unknown
Staging app...
CANCEL
NEXT


酷狗音乐
Do you want to install an update to this existing application? Your existing data will not be lost. The updated application will get access to:
NEW
ALL
draw over other apps
modify system settings
take pictures and videos
read your contacts
access approximate location (network-based)
access precise location (GPS and network-based)
record audio
read phone status and identity
reroute outgoing calls
modify or delete the contents of your SD
CANCEL
INSTALL


酷狗音乐
Installing...