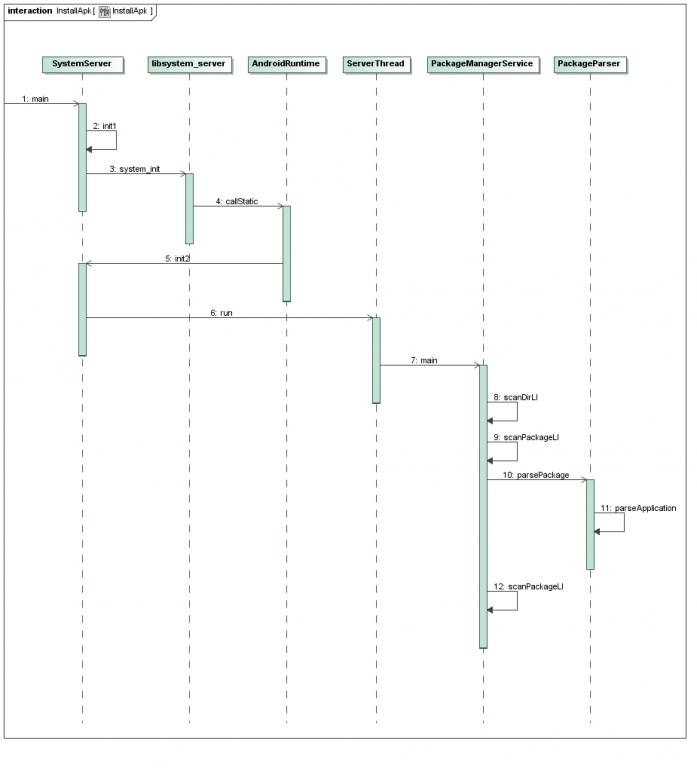
Android系统在启动的过程中，会启动一个应用程序管理服务PackageManagerService，这个服务负责扫描系统中特定的目录，找到里面的应用程序文件，即以Apk为后缀的文件，然后对这些文件进解析，得到应用程序的相关信息，完成应用程序的安装过程，本文将详细分析这个过程。

        应用程序管理服务PackageManagerService安装应用程序的过程，其实就是解析析应用程序配置文件AndroidManifest.xml的过程，并从里面得到得到应用程序的相关信息，例如得到应用程序的组件Activity、Service、Broadcast Receiver和Content Provider等信息，有了这些信息后，通过ActivityManagerService这个服务，我们就可以在系统中正常地使用这些应用程序了。

        应用程序管理服务PackageManagerService是系统启动的时候由SystemServer组件启动的，启后它就会执行应用程序安装的过程，因此，本文将从SystemServer启动PackageManagerService服务的过程开始分析系统中的应用程序安装的过程。

        应用程序管理服务PackageManagerService从启动到安装应用程序的过程如下图所示：



        下面我们具体分析每一个步骤。

        Step 1. SystemServer.main

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. public class SystemServer
2. {
3. ......
5. native public static void init1(String[] args);
7. ......
9. public static void main(String[] args) {
10. ......
12. init1(args);
14. ......
15. }
17. ......
18. }

public class SystemServer

{

......

native public static void init1(String[] args);

......

public static void main(String[] args) {

......

init1(args);

......

}

......

}

        SystemServer组件是由Zygote进程负责启动的，启动的时候就会调用它的main函数，这个函数主要调用了JNI方法init1来做一些系统初始化的工作。

        Step 2. SystemServer.init1

        这个函数是一个JNI方法，实现在 frameworks/base/services/jni/com\_android\_server\_SystemServer.cpp文件中：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. namespace android {
3. extern "C" **int** system\_init();
5. static void android\_server\_SystemServer\_init1(JNIEnv\* env, jobject clazz)
6. {
7. system\_init();
8. }
10. /\*
11. \* JNI registration.
12. \*/
13. static JNINativeMethod gMethods[] = {
14. /\* name, signature, funcPtr \*/
15. { "init1", "([Ljava/lang/String;)V", (void\*) android\_server\_SystemServer\_init1 },
16. };
18. **int** register\_android\_server\_SystemServer(JNIEnv\* env)
19. {
20. return jniRegisterNativeMethods(env, "com/android/server/SystemServer",
21. gMethods, NELEM(gMethods));
22. }
24. }; // namespace android

namespace android {

extern "C" int system\_init();

static void android\_server\_SystemServer\_init1(JNIEnv\* env, jobject clazz)

{

system\_init();

}

/\*

\* JNI registration.

\*/

static JNINativeMethod gMethods[] = {

/\* name, signature, funcPtr \*/

{ "init1", "([Ljava/lang/String;)V", (void\*) android\_server\_SystemServer\_init1 },

};

int register\_android\_server\_SystemServer(JNIEnv\* env)

{

return jniRegisterNativeMethods(env, "com/android/server/SystemServer",

gMethods, NELEM(gMethods));

}

}; // namespace android

        这个函数很简单，只是调用了system\_init函数来进一步执行操作。

        Step 3. libsystem\_server.system\_init

        函数system\_init实现在libsystem\_server库中，源代码位于frameworks/base/cmds/system\_server/library/system\_init.cpp文件中：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. extern "C" status\_t system\_init()
2. {
3. LOGI("Entered system\_init()");
5. sp<ProcessState> proc(ProcessState::self());
7. sp<IServiceManager> sm = defaultServiceManager();
8. LOGI("ServiceManager: %p\n", sm.get());
10. sp<GrimReaper> grim = new GrimReaper();
11. sm->asBinder()->linkToDeath(grim, grim.get(), 0);
13. **char** propBuf[PROPERTY\_VALUE\_MAX];
14. property\_get("system\_init.startsurfaceflinger", propBuf, "1");
15. if (strcmp(propBuf, "1") == 0) {
16. // Start the SurfaceFlinger
17. SurfaceFlinger::instantiate();
18. }
20. // Start the sensor service
21. SensorService::instantiate();

24. // On the simulator, audioflinger et al don't get started the
25. // same way as on the device, and we need to start them here
26. if (!proc->supportsProcesses()) {
27. 82的代码中没有下面这些服务的启动
28. // Start the AudioFlinger
29. AudioFlinger::instantiate();
31. // Start the media playback service
32. MediaPlayerService::instantiate();
34. // Start the camera service
35. CameraService::instantiate();
37. // Start the audio policy service
38. AudioPolicyService::instantiate();
39. }
41. // And now start the Android runtime.  We have to do this bit
42. // of nastiness because the Android runtime initialization requires
43. // some of the core system services to already be started.
44. // All other servers should just start the Android runtime at
45. // the beginning of their processes's main(), before calling
46. // the init function.
47. LOGI("System server: starting Android runtime.\n");
49. AndroidRuntime\* runtime = AndroidRuntime::getRuntime();
51. LOGI("System server: starting Android services.\n");
52. runtime->callStatic("com/android/server/SystemServer", "init2");
54. // If running in our own process, just go into the thread
55. // pool.  Otherwise, call the initialization finished
56. // func to let this process continue its initilization.
57. if (proc->supportsProcesses()) {
58. LOGI("System server: entering thread pool.\n");
59. ProcessState::self()->startThreadPool();
60. IPCThreadState::self()->joinThreadPool();
61. LOGI("System server: exiting thread pool.\n");
62. }
64. return NO\_ERROR;
65. }

extern "C" status\_t system\_init()

{

LOGI("Entered system\_init()");

sp<ProcessState> proc(ProcessState::self());

sp<IServiceManager> sm = defaultServiceManager();

LOGI("ServiceManager: %p\n", sm.get());

sp<GrimReaper> grim = new GrimReaper();

sm->asBinder()->linkToDeath(grim, grim.get(), 0);

char propBuf[PROPERTY\_VALUE\_MAX];

property\_get("system\_init.startsurfaceflinger", propBuf, "1");

if (strcmp(propBuf, "1") == 0) {

// Start the SurfaceFlinger

SurfaceFlinger::instantiate();

}

// Start the sensor service

SensorService::instantiate();

// On the simulator, audioflinger et al don't get started the

// same way as on the device, and we need to start them here

if (!proc->supportsProcesses()) {

// Start the AudioFlinger

AudioFlinger::instantiate();

// Start the media playback service

MediaPlayerService::instantiate();

// Start the camera service

CameraService::instantiate();

// Start the audio policy service

AudioPolicyService::instantiate();

}

// And now start the Android runtime. We have to do this bit

// of nastiness because the Android runtime initialization requires

// some of the core system services to already be started.

// All other servers should just start the Android runtime at

// the beginning of their processes's main(), before calling

// the init function.

LOGI("System server: starting Android runtime.\n");

AndroidRuntime\* runtime = AndroidRuntime::getRuntime();

LOGI("System server: starting Android services.\n");

runtime->callStatic("com/android/server/SystemServer", "init2");

// If running in our own process, just go into the thread

// pool. Otherwise, call the initialization finished

// func to let this process continue its initilization.

if (proc->supportsProcesses()) {

LOGI("System server: entering thread pool.\n");

ProcessState::self()->startThreadPool();

IPCThreadState::self()->joinThreadPool();

LOGI("System server: exiting thread pool.\n");

}

return NO\_ERROR;

}

        这个函数首先会初始化SurfaceFlinger、SensorService、AudioFlinger、MediaPlayerService、CameraService和AudioPolicyService这几个服务，然后就通过系统全局唯一的AndroidRuntime实例变量runtime的callStatic来调用SystemServer的init2函数了。关于这个AndroidRuntime实例变量runtime的相关资料，可能参考前面一篇文章[Android应用程序进程启动过程的源代码](http://blog.csdn.net/luoshengyang/article/details/6747696)分析一文。

        Step 4. AndroidRuntime.callStatic

        这个函数定义在frameworks/base/core/jni/AndroidRuntime.cpp文件中：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. /\*
2. \* Call a static Java Programming Language function that takes no arguments and returns void.
3. \*/
4. status\_t AndroidRuntime::callStatic(const **char**\* className, const **char**\* methodName)
5. {
6. JNIEnv\* env;  //研究一下这个env
7. jclass clazz;
8. jmethodID methodId;
10. env = getJNIEnv();
11. if (env == NULL)
12. return UNKNOWN\_ERROR;
14. clazz = findClass(env, className);
15. if (clazz == NULL) {
16. LOGE("ERROR: could not find class '%s'\n", className);
17. return UNKNOWN\_ERROR;
18. }
19. methodId = env->GetStaticMethodID(clazz, methodName, "()V");
20. if (methodId == NULL) {
21. LOGE("ERROR: could not find method %s.%s\n", className, methodName);
22. return UNKNOWN\_ERROR;
23. }
25. env->CallStaticVoidMethod(clazz, methodId);
27. return NO\_ERROR;
28. }

/\*

\* Call a static Java Programming Language function that takes no arguments and returns void.

\*/

status\_t AndroidRuntime::callStatic(const char\* className, const char\* methodName)

{

JNIEnv\* env;

jclass clazz;

jmethodID methodId;

env = getJNIEnv();

if (env == NULL)

return UNKNOWN\_ERROR;

clazz = findClass(env, className);

if (clazz == NULL) {

LOGE("ERROR: could not find class '%s'\n", className);

return UNKNOWN\_ERROR;

}

methodId = env->GetStaticMethodID(clazz, methodName, "()V");

if (methodId == NULL) {

LOGE("ERROR: could not find method %s.%s\n", className, methodName);

return UNKNOWN\_ERROR;

}

env->CallStaticVoidMethod(clazz, methodId);

return NO\_ERROR;

}

        这个函数调用由参数className指定的java类的静态成员函数，这个静态成员函数是由参数methodName指定的。上面传进来的参数className的值为"com/android/server/SystemServer"，而参数methodName的值为"init2"，因此，接下来就会调用SystemServer类的init2函数了。

        Step 5. SystemServer.init2

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. public class SystemServer
2. {
3. ......
5. public static final void init2() {
6. Slog.i(TAG, "Entered the Android system server!");
7. Thread thr = new ServerThread();
8. thr.setName("android.server.ServerThread");
9. thr.start();
10. }
11. }

public class SystemServer

{

......

public static final void init2() {

Slog.i(TAG, "Entered the Android system server!");

Thread thr = new ServerThread();

thr.setName("android.server.ServerThread");

thr.start();

}

}

        这个函数创建了一个ServerThread线程，PackageManagerService服务就是这个线程中启动的了。这里调用了ServerThread实例thr的start函数之后，下面就会执行这个实例的run函数了。

        Step 6. ServerThread.run

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class ServerThread extends Thread {
2. ......
4. @Override
5. public void run() {  //这个方法相当重要
6. ......
8. IPackageManager pm = null;
10. ......
12. // Critical services...
13. try {
14. ......
16. Slog.i(TAG, "Package Manager");
17. pm = PackageManagerService.main(context,
18. factoryTest != SystemServer.FACTORY\_TEST\_OFF);
20. ......
21. } catch (RuntimeException e) {
22. Slog.e("System", "Failure starting core service", e);
23. }
25. ......
26. }
28. ......
29. }

class ServerThread extends Thread {

......

@Override

public void run() {

......

IPackageManager pm = null;

......

// Critical services...

try {

......

Slog.i(TAG, "Package Manager");

pm = PackageManagerService.main(context,

factoryTest != SystemServer.FACTORY\_TEST\_OFF);

......

} catch (RuntimeException e) {

Slog.e("System", "Failure starting core service", e);

}

......

}

......

}

        这个函数除了启动PackageManagerService服务之外，还启动了其它很多的服务，例如在前面学习Activity和Service的几篇文章中经常看到的ActivityManagerService服务，有兴趣的读者可以自己研究一下。

        Step 7. PackageManagerService.main

        这个函数定义在frameworks/base/services/java/com/android/server/PackageManagerService.java文件中：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
4. public static final IPackageManager main(Context context, boolean factoryTest) {
5. PackageManagerService m = new PackageManagerService(context, factoryTest);
6. ServiceManager.addService("package", m);  //这里要重点参照IPC的机制
7. return m;
8. }
10. ......
11. }

class PackageManagerService extends IPackageManager.Stub {

......

public static final IPackageManager main(Context context, boolean factoryTest) {

PackageManagerService m = new PackageManagerService(context, factoryTest);

ServiceManager.addService("package", m);

return m;

}

......

}

        这个函数创建了一个PackageManagerService服务实例，然后把这个服务添加到ServiceManager中去，ServiceManager是Android系统Binder进程间通信机制的守护进程，负责管理系统中的Binder对象，具体可以参考[浅谈Service Manager成为Android进程间通信（IPC）机制Binder守护进程之路](http://blog.csdn.net/luoshengyang/article/details/6621566)一文。  
        在创建这个PackageManagerService服务实例时，会在PackageManagerService类的构造函数中开始执行安装应用程序的过程：

从这步开始，就要执行apk文件的扫描和解析，都是在PackageManagerService中完成的。

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
3. //构造函数中扫描出了所有apk文件
4. public PackageManagerService(Context context, boolean factoryTest) {
5. ......
7. synchronized (mInstallLock) {
8. synchronized (mPackages) {
9. ......
11. File dataDir = Environment.getDataDirectory();
12. mAppDataDir = new File(dataDir, "data");
13. mSecureAppDataDir = new File(dataDir, "secure/data");
14. mDrmAppPrivateInstallDir = new File(dataDir, "app-private");
16. ......
18. mFrameworkDir = new File(Environment.getRootDirectory(), "framework");
19. mDalvikCacheDir = new File(dataDir, "dalvik-cache");
21. ......
23. // Find base frameworks (resource packages without code).
24. mFrameworkInstallObserver = new AppDirObserver(
25. mFrameworkDir.getPath(), OBSERVER\_EVENTS, true);
26. mFrameworkInstallObserver.startWatching();
27. scanDirLI(mFrameworkDir, PackageParser.PARSE\_IS\_SYSTEM
28. | PackageParser.PARSE\_IS\_SYSTEM\_DIR,
29. scanMode | SCAN\_NO\_DEX, 0);
31. // Collect all system packages.
32. mSystemAppDir = new File(Environment.getRootDirectory(), "app");
33. mSystemInstallObserver = new AppDirObserver(
34. mSystemAppDir.getPath(), OBSERVER\_EVENTS, true);
35. mSystemInstallObserver.startWatching();
36. scanDirLI(mSystemAppDir, PackageParser.PARSE\_IS\_SYSTEM
37. | PackageParser.PARSE\_IS\_SYSTEM\_DIR, scanMode, 0);
39. // Collect all vendor packages.
40. mVendorAppDir = new File("/vendor/app");
41. mVendorInstallObserver = new AppDirObserver(
42. mVendorAppDir.getPath(), OBSERVER\_EVENTS, true);
43. mVendorInstallObserver.startWatching();
44. scanDirLI(mVendorAppDir, PackageParser.PARSE\_IS\_SYSTEM
45. | PackageParser.PARSE\_IS\_SYSTEM\_DIR, scanMode, 0);

48. mAppInstallObserver = new AppDirObserver(
49. mAppInstallDir.getPath(), OBSERVER\_EVENTS, false);
50. mAppInstallObserver.startWatching();
51. scanDirLI(mAppInstallDir, 0, scanMode, 0);
53. mDrmAppInstallObserver = new AppDirObserver(
54. mDrmAppPrivateInstallDir.getPath(), OBSERVER\_EVENTS, false);
55. mDrmAppInstallObserver.startWatching();
56. scanDirLI(mDrmAppPrivateInstallDir, PackageParser.PARSE\_FORWARD\_LOCK,
57. scanMode, 0);
59. ......
60. }
61. }
62. }
64. ......
65. }

class PackageManagerService extends IPackageManager.Stub {

......

public PackageManagerService(Context context, boolean factoryTest) {

......

synchronized (mInstallLock) {

synchronized (mPackages) {

......

File dataDir = Environment.getDataDirectory();

mAppDataDir = new File(dataDir, "data");

mSecureAppDataDir = new File(dataDir, "secure/data");

mDrmAppPrivateInstallDir = new File(dataDir, "app-private");

......

mFrameworkDir = new File(Environment.getRootDirectory(), "framework");

mDalvikCacheDir = new File(dataDir, "dalvik-cache");

......

// Find base frameworks (resource packages without code).

mFrameworkInstallObserver = new AppDirObserver(

mFrameworkDir.getPath(), OBSERVER\_EVENTS, true);

mFrameworkInstallObserver.startWatching();

scanDirLI(mFrameworkDir, PackageParser.PARSE\_IS\_SYSTEM

| PackageParser.PARSE\_IS\_SYSTEM\_DIR,

scanMode | SCAN\_NO\_DEX, 0);

// Collect all system packages.

mSystemAppDir = new File(Environment.getRootDirectory(), "app");

mSystemInstallObserver = new AppDirObserver(

mSystemAppDir.getPath(), OBSERVER\_EVENTS, true);

mSystemInstallObserver.startWatching();

scanDirLI(mSystemAppDir, PackageParser.PARSE\_IS\_SYSTEM

| PackageParser.PARSE\_IS\_SYSTEM\_DIR, scanMode, 0);

// Collect all vendor packages.

mVendorAppDir = new File("/vendor/app");

mVendorInstallObserver = new AppDirObserver(

mVendorAppDir.getPath(), OBSERVER\_EVENTS, true);

mVendorInstallObserver.startWatching();

scanDirLI(mVendorAppDir, PackageParser.PARSE\_IS\_SYSTEM

| PackageParser.PARSE\_IS\_SYSTEM\_DIR, scanMode, 0);

mAppInstallObserver = new AppDirObserver(

mAppInstallDir.getPath(), OBSERVER\_EVENTS, false);

mAppInstallObserver.startWatching();

scanDirLI(mAppInstallDir, 0, scanMode, 0);

mDrmAppInstallObserver = new AppDirObserver(

mDrmAppPrivateInstallDir.getPath(), OBSERVER\_EVENTS, false);

mDrmAppInstallObserver.startWatching();

scanDirLI(mDrmAppPrivateInstallDir, PackageParser.PARSE\_FORWARD\_LOCK,

scanMode, 0);

......

}

}

}

......

}

        这里会调用scanDirLI函数来扫描移动设备上的下面这五个目录中的Apk文件：

***/system/framework***

***/system/app***

***/vendor/app***

***/data/app***

***/data/app-private***

       Step 8. PackageManagerService.scanDirLI  
       这个函数定义在frameworks/base/services/java/com/android/server/PackageManagerService.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
4. private void scanDirLI(File dir, int flags, int scanMode, long currentTime) {
5. String[] files = dir.list();
6. ......
8. int i;
9. for (i=0; i<files.length; i++) {
10. File file = new File(dir, files[i]);
11. if (!isPackageFilename(files[i])) {
12. // Ignore entries which are not apk's
13. continue;
14. }
15. PackageParser.Package pkg = scanPackageLI(file,
16. flags|PackageParser.PARSE\_MUST\_BE\_APK, scanMode, currentTime);
17. // Don't mess around with apps in system partition.
18. if (pkg == null && (flags & PackageParser.PARSE\_IS\_SYSTEM) == 0 &&
19. mLastScanError == PackageManager.INSTALL\_FAILED\_INVALID\_APK) {
20. // Delete the apk
21. Slog.w(TAG, "Cleaning up failed install of " + file);
22. file.delete();
23. }
24. }
25. }

28. ......
29. }

class PackageManagerService extends IPackageManager.Stub {

......

private void scanDirLI(File dir, int flags, int scanMode, long currentTime) {

String[] files = dir.list();

......

int i;

for (i=0; i<files.length; i++) {

File file = new File(dir, files[i]);

if (!isPackageFilename(files[i])) {

// Ignore entries which are not apk's

continue;

}

PackageParser.Package pkg = scanPackageLI(file,

flags|PackageParser.PARSE\_MUST\_BE\_APK, scanMode, currentTime);

// Don't mess around with apps in system partition.

if (pkg == null && (flags & PackageParser.PARSE\_IS\_SYSTEM) == 0 &&

mLastScanError == PackageManager.INSTALL\_FAILED\_INVALID\_APK) {

// Delete the apk

Slog.w(TAG, "Cleaning up failed install of " + file);

file.delete();

}

}

}

......

}

         对于目录中的每一个文件，如果是以后Apk作为后缀名，那么就调用scanPackageLI函数来对它进行解析和安装。

         Step 9. PackageManagerService.scanPackageLI

         这个函数定义在frameworks/base/services/java/com/android/server/PackageManagerService.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
4. private PackageParser.Package scanPackageLI(File scanFile,
5. int parseFlags, int scanMode, long currentTime) {
6. ......
8. String scanPath = scanFile.getPath();
9. parseFlags |= mDefParseFlags;
10. PackageParser pp = new PackageParser(scanPath);
12. ......
14. final PackageParser.Package pkg = pp.parsePackage(scanFile,
15. scanPath, mMetrics, parseFlags);
17. ......
19. return scanPackageLI(pkg, parseFlags, scanMode | SCAN\_UPDATE\_SIGNATURE, currentTime);
20. }
22. ......
23. }

class PackageManagerService extends IPackageManager.Stub {

......

private PackageParser.Package scanPackageLI(File scanFile,

int parseFlags, int scanMode, long currentTime) {

......

String scanPath = scanFile.getPath();

parseFlags |= mDefParseFlags;

PackageParser pp = new PackageParser(scanPath);

......

final PackageParser.Package pkg = pp.parsePackage(scanFile,

scanPath, mMetrics, parseFlags);

......

return scanPackageLI(pkg, parseFlags, scanMode | SCAN\_UPDATE\_SIGNATURE, currentTime);

}

......

}

        这个函数首先会为这个Apk文件创建一个PackageParser实例，接着调用这个实例的parsePackage函数来对这个Apk文件进行解析。这个函数最后还会调用另外一个版本的scanPackageLI函数把来解析后得到的应用程序信息保存在PackageManagerService中。

        Step 10. PackageParser.parsePackage  
        这个函数定义在frameworks/base/core/java/android/content/pm/PackageParser.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. public class PackageParser {
2. ......
4. public Package parsePackage(File sourceFile, String destCodePath,
5. DisplayMetrics metrics, int flags) {
6. ......
8. mArchiveSourcePath = sourceFile.getPath();
10. ......
12. XmlResourceParser parser = null;
13. AssetManager assmgr = null;
14. boolean assetError = true;
15. try {
16. assmgr = new AssetManager();
17. int cookie = assmgr.addAssetPath(mArchiveSourcePath);
18. if(cookie != 0) {
19. parser = assmgr.openXmlResourceParser(cookie, "AndroidManifest.xml");
20. assetError = false;
21. } else {
22. ......
23. }
24. } catch (Exception e) {
25. ......
26. }
28. ......
30. String[] errorText = new String[1];
31. Package pkg = null;
32. Exception errorException = null;
33. try {
34. // XXXX todo: need to figure out correct configuration.
35. Resources res = new Resources(assmgr, metrics, null);
36. pkg = parsePackage(res, parser, flags, errorText);
37. } catch (Exception e) {
38. ......
39. }
41. ......
43. parser.close();
44. assmgr.close();
46. // Set code and resource paths
47. pkg.mPath = destCodePath;
48. pkg.mScanPath = mArchiveSourcePath;
49. //pkg.applicationInfo.sourceDir = destCodePath;
50. //pkg.applicationInfo.publicSourceDir = destRes;
51. pkg.mSignatures = null;
53. return pkg;
54. }
56. ......
57. }

public class PackageParser {

......

public Package parsePackage(File sourceFile, String destCodePath,

DisplayMetrics metrics, int flags) {

......

mArchiveSourcePath = sourceFile.getPath();

......

XmlResourceParser parser = null;

AssetManager assmgr = null;

boolean assetError = true;

try {

assmgr = new AssetManager();

int cookie = assmgr.addAssetPath(mArchiveSourcePath);

if(cookie != 0) {

parser = assmgr.openXmlResourceParser(cookie, "AndroidManifest.xml");

assetError = false;

} else {

......

}

} catch (Exception e) {

......

}

......

String[] errorText = new String[1];

Package pkg = null;

Exception errorException = null;

try {

// XXXX todo: need to figure out correct configuration.

Resources res = new Resources(assmgr, metrics, null);

pkg = parsePackage(res, parser, flags, errorText);

} catch (Exception e) {

......

}

......

parser.close();

assmgr.close();

// Set code and resource paths

pkg.mPath = destCodePath;

pkg.mScanPath = mArchiveSourcePath;

//pkg.applicationInfo.sourceDir = destCodePath;

//pkg.applicationInfo.publicSourceDir = destRes;

pkg.mSignatures = null;

return pkg;

}

......

}

        每一个Apk文件都是一个归档文件，它里面包含了Android应用程序的配置文件AndroidManifest.xml，这里主要就是要对这个配置文件就行解析了，从Apk归档文件中得到这个配置文件后，就调用另一外版本的parsePackage函数对这个应用程序进行解析了：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. public class PackageParser {
2. ......
4. private Package parsePackage(
5. Resources res, XmlResourceParser parser, int flags, String[] outError)
6. throws XmlPullParserException, IOException {
7. ......
9. String pkgName = parsePackageName(parser, attrs, flags, outError);
11. ......
13. final Package pkg = new Package(pkgName);
15. ......
17. int type;
19. ......
21. TypedArray sa = res.obtainAttributes(attrs,
22. com.android.internal.R.styleable.AndroidManifest);
24. ......
26. while ((type=parser.next()) != parser.END\_DOCUMENT
27. && (type != parser.END\_TAG || parser.getDepth() > outerDepth)) {
28. if (type == parser.END\_TAG || type == parser.TEXT) {
29. continue;
30. }
32. String tagName = parser.getName();
33. if (tagName.equals("application")) {
34. ......
36. if (!parseApplication(pkg, res, parser, attrs, flags, outError)) {
37. return null;
38. }
39. } else if (tagName.equals("permission-group")) {
40. ......
41. } else if (tagName.equals("permission")) {
42. ......
43. } else if (tagName.equals("permission-tree")) {
44. ......
45. } else if (tagName.equals("uses-permission")) {
46. ......
47. } else if (tagName.equals("uses-configuration")) {
48. ......
49. } else if (tagName.equals("uses-feature")) {
50. ......
51. } else if (tagName.equals("uses-sdk")) {
52. ......
53. } else if (tagName.equals("supports-screens")) {
54. ......
55. } else if (tagName.equals("protected-broadcast")) {
56. ......
57. } else if (tagName.equals("instrumentation")) {
58. ......
59. } else if (tagName.equals("original-package")) {
60. ......
61. } else if (tagName.equals("adopt-permissions")) {
62. ......
63. } else if (tagName.equals("uses-gl-texture")) {
64. ......
65. } else if (tagName.equals("compatible-screens")) {
66. ......
67. } else if (tagName.equals("eat-comment")) {
68. ......
69. } else if (RIGID\_PARSER) {
70. ......
71. } else {
72. ......
73. }
74. }
76. ......
78. return pkg;
79. }
81. ......
82. }

public class PackageParser {

......

private Package parsePackage(

Resources res, XmlResourceParser parser, int flags, String[] outError)

throws XmlPullParserException, IOException {

......

String pkgName = parsePackageName(parser, attrs, flags, outError);

......

final Package pkg = new Package(pkgName);

......

int type;

......

TypedArray sa = res.obtainAttributes(attrs,

com.android.internal.R.styleable.AndroidManifest);

......

while ((type=parser.next()) != parser.END\_DOCUMENT

&& (type != parser.END\_TAG || parser.getDepth() > outerDepth)) {

if (type == parser.END\_TAG || type == parser.TEXT) {

continue;

}

String tagName = parser.getName();

if (tagName.equals("application")) {

......

if (!parseApplication(pkg, res, parser, attrs, flags, outError)) {

return null;

}

} else if (tagName.equals("permission-group")) {

......

} else if (tagName.equals("permission")) {

......

} else if (tagName.equals("permission-tree")) {

......

} else if (tagName.equals("uses-permission")) {

......

} else if (tagName.equals("uses-configuration")) {

......

} else if (tagName.equals("uses-feature")) {

......

} else if (tagName.equals("uses-sdk")) {

......

} else if (tagName.equals("supports-screens")) {

......

} else if (tagName.equals("protected-broadcast")) {

......

} else if (tagName.equals("instrumentation")) {

......

} else if (tagName.equals("original-package")) {

......

} else if (tagName.equals("adopt-permissions")) {

......

} else if (tagName.equals("uses-gl-texture")) {

......

} else if (tagName.equals("compatible-screens")) {

......

} else if (tagName.equals("eat-comment")) {

......

} else if (RIGID\_PARSER) {

......

} else {

......

}

}

......

return pkg;

}

......

}

        这里就是对AndroidManifest.xml文件中的各个标签进行解析了，各个标签的含义可以参考官方文档<http://developer.android.com/guide/topics/manifest/manifest-intro.html>，这里我们只简单看一下application标签的解析，这是通过调用parseApplication函数来进行的。

        Step 11. PackageParser.parseApplication  
        这个函数定义在frameworks/base/core/java/android/content/pm/PackageParser.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. public class PackageParser {
2. ......
4. private boolean parseApplication(Package owner, Resources res,
5. XmlPullParser parser, AttributeSet attrs, int flags, String[] outError)
6. throws XmlPullParserException, IOException {
7. final ApplicationInfo ai = owner.applicationInfo;
8. final String pkgName = owner.applicationInfo.packageName;
10. TypedArray sa = res.obtainAttributes(attrs,
11. com.android.internal.R.styleable.AndroidManifestApplication);
13. ......
15. int type;
16. while ((type=parser.next()) != parser.END\_DOCUMENT
17. && (type != parser.END\_TAG || parser.getDepth() > innerDepth)) {
18. if (type == parser.END\_TAG || type == parser.TEXT) {
19. continue;
20. }
22. String tagName = parser.getName();
23. if (tagName.equals("activity")) {
24. Activity a = parseActivity(owner, res, parser, attrs, flags, outError, false);
25. ......
27. owner.activities.add(a);
29. } else if (tagName.equals("receiver")) {
30. Activity a = parseActivity(owner, res, parser, attrs, flags, outError, true);
31. ......
33. owner.receivers.add(a);
34. } else if (tagName.equals("service")) {
35. Service s = parseService(owner, res, parser, attrs, flags, outError);
36. ......
38. owner.services.add(s);
39. } else if (tagName.equals("provider")) {
40. Provider p = parseProvider(owner, res, parser, attrs, flags, outError);
41. ......
43. owner.providers.add(p);
44. } else if (tagName.equals("activity-alias")) {
45. Activity a = parseActivityAlias(owner, res, parser, attrs, flags, outError);
46. ......
48. owner.activities.add(a);
49. } else if (parser.getName().equals("meta-data")) {
50. ......
51. } else if (tagName.equals("uses-library")) {
52. ......
53. } else if (tagName.equals("uses-package")) {
54. ......
55. } else {
56. ......
57. }
58. }
60. return true;
61. }
63. ......
64. }

public class PackageParser {

......

private boolean parseApplication(Package owner, Resources res,

XmlPullParser parser, AttributeSet attrs, int flags, String[] outError)

throws XmlPullParserException, IOException {

final ApplicationInfo ai = owner.applicationInfo;

final String pkgName = owner.applicationInfo.packageName;

TypedArray sa = res.obtainAttributes(attrs,

com.android.internal.R.styleable.AndroidManifestApplication);

......

int type;

while ((type=parser.next()) != parser.END\_DOCUMENT

&& (type != parser.END\_TAG || parser.getDepth() > innerDepth)) {

if (type == parser.END\_TAG || type == parser.TEXT) {

continue;

}

String tagName = parser.getName();

if (tagName.equals("activity")) {

Activity a = parseActivity(owner, res, parser, attrs, flags, outError, false);

......

owner.activities.add(a);

} else if (tagName.equals("receiver")) {

Activity a = parseActivity(owner, res, parser, attrs, flags, outError, true);

......

owner.receivers.add(a);

} else if (tagName.equals("service")) {

Service s = parseService(owner, res, parser, attrs, flags, outError);

......

owner.services.add(s);

} else if (tagName.equals("provider")) {

Provider p = parseProvider(owner, res, parser, attrs, flags, outError);

......

owner.providers.add(p);

} else if (tagName.equals("activity-alias")) {

Activity a = parseActivityAlias(owner, res, parser, attrs, flags, outError);

......

owner.activities.add(a);

} else if (parser.getName().equals("meta-data")) {

......

} else if (tagName.equals("uses-library")) {

......

} else if (tagName.equals("uses-package")) {

......

} else {

......

}

}

return true;

}

......

}

        这里就是对AndroidManifest.xml文件中的application标签进行解析了，我们常用到的标签就有activity、service、receiver和provider，各个标签的含义可以参考官方文档<http://developer.android.com/guide/topics/manifest/manifest-intro.html>。

        这里解析完成后，一层层返回到Step 9中，调用另一个版本的scanPackageLI函数把来解析后得到的应用程序信息保存下来。

        Step 12. PackageManagerService.scanPackageLI

        这个函数定义在frameworks/base/services/java/com/android/server/PackageManagerService.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6766010)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
4. // Keys are String (package name), values are Package.  This also serves
5. // as the lock for the global state.  Methods that must be called with
6. // this lock held have the prefix "LP".
7. final HashMap<String, PackageParser.Package> mPackages =
8. new HashMap<String, PackageParser.Package>();
10. ......
12. // All available activities, for your resolving pleasure.
13. final ActivityIntentResolver mActivities =
14. new ActivityIntentResolver();
16. // All available receivers, for your resolving pleasure.
17. final ActivityIntentResolver mReceivers =
18. new ActivityIntentResolver();
20. // All available services, for your resolving pleasure.
21. final ServiceIntentResolver mServices = new ServiceIntentResolver();
23. // Keys are String (provider class name), values are Provider.
24. final HashMap<ComponentName, PackageParser.Provider> mProvidersByComponent =
25. new HashMap<ComponentName, PackageParser.Provider>();
27. ......
29. private PackageParser.Package scanPackageLI(PackageParser.Package pkg,
30. int parseFlags, int scanMode, long currentTime) {
31. ......
33. synchronized (mPackages) {
34. ......
36. // Add the new setting to mPackages
37. mPackages.put(pkg.applicationInfo.packageName, pkg);
39. ......
41. int N = pkg.providers.size();
42. int i;
43. for (i=0; i<N; i++) {
44. PackageParser.Provider p = pkg.providers.get(i);
45. p.info.processName = fixProcessName(pkg.applicationInfo.processName,
46. p.info.processName, pkg.applicationInfo.uid);
47. mProvidersByComponent.put(new ComponentName(p.info.packageName,
48. p.info.name), p);
50. ......
51. }
53. N = pkg.services.size();
54. for (i=0; i<N; i++) {
55. PackageParser.Service s = pkg.services.get(i);
56. s.info.processName = fixProcessName(pkg.applicationInfo.processName,
57. s.info.processName, pkg.applicationInfo.uid);
58. mServices.addService(s);
60. ......
61. }
63. N = pkg.receivers.size();
64. r = null;
65. for (i=0; i<N; i++) {
66. PackageParser.Activity a = pkg.receivers.get(i);
67. a.info.processName = fixProcessName(pkg.applicationInfo.processName,
68. a.info.processName, pkg.applicationInfo.uid);
69. mReceivers.addActivity(a, "receiver");
71. ......
72. }
74. N = pkg.activities.size();
75. for (i=0; i<N; i++) {
76. PackageParser.Activity a = pkg.activities.get(i);
77. a.info.processName = fixProcessName(pkg.applicationInfo.processName,
78. a.info.processName, pkg.applicationInfo.uid);
79. mActivities.addActivity(a, "activity");
81. ......
82. }
84. ......
85. }
87. ......
89. return pkg;
90. }
92. ......
93. }

class PackageManagerService extends IPackageManager.Stub {

......

// Keys are String (package name), values are Package. This also serves

// as the lock for the global state. Methods that must be called with

// this lock held have the prefix "LP".

final HashMap<String, PackageParser.Package> mPackages =

new HashMap<String, PackageParser.Package>();

......

// 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();

// Keys are String (provider class name), values are Provider.

final HashMap<ComponentName, PackageParser.Provider> mProvidersByComponent =

new HashMap<ComponentName, PackageParser.Provider>();

......

private PackageParser.Package scanPackageLI(PackageParser.Package pkg,

int parseFlags, int scanMode, long currentTime) {

......

synchronized (mPackages) {

......

// Add the new setting to mPackages

mPackages.put(pkg.applicationInfo.packageName, pkg);

......

int N = pkg.providers.size();

int i;

for (i=0; i<N; i++) {

PackageParser.Provider p = pkg.providers.get(i);

p.info.processName = fixProcessName(pkg.applicationInfo.processName,

p.info.processName, pkg.applicationInfo.uid);

mProvidersByComponent.put(new ComponentName(p.info.packageName,

p.info.name), p);

......

}

N = pkg.services.size();

for (i=0; i<N; i++) {

PackageParser.Service s = pkg.services.get(i);

s.info.processName = fixProcessName(pkg.applicationInfo.processName,

s.info.processName, pkg.applicationInfo.uid);

mServices.addService(s);

......

}

N = pkg.receivers.size();

r = null;

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++) {

PackageParser.Activity a = pkg.activities.get(i);

a.info.processName = fixProcessName(pkg.applicationInfo.processName,

a.info.processName, pkg.applicationInfo.uid);

mActivities.addActivity(a, "activity");

......

}

......

}

......

return pkg;

}

......

}

        这个函数主要就是把前面解析应用程序得到的package、provider、service、receiver和activity等信息保存在PackageManagerService服务中了。

        这样，在Android系统启动的时候安装应用程序的过程就介绍完了，但是，这些应用程序只是相当于在PackageManagerService服务注册好了，如果我们想要在Android桌面上看到这些应用程序，还需要有一个Home应用程序，负责从PackageManagerService服务中把这些安装好的应用程序取出来，并以友好的方式在桌面上展现出来，例如以快捷图标的形式。在Android系统中，负责把系统中已经安装的应用程序在桌面中展现出来的Home应用程序就是Launcher了，在下一篇文章中，我们将介绍Launcher是如何启动的以及它是如何从PackageManagerService服务中把系统中已经安装好的应用程序展现出来的，敬请期待。