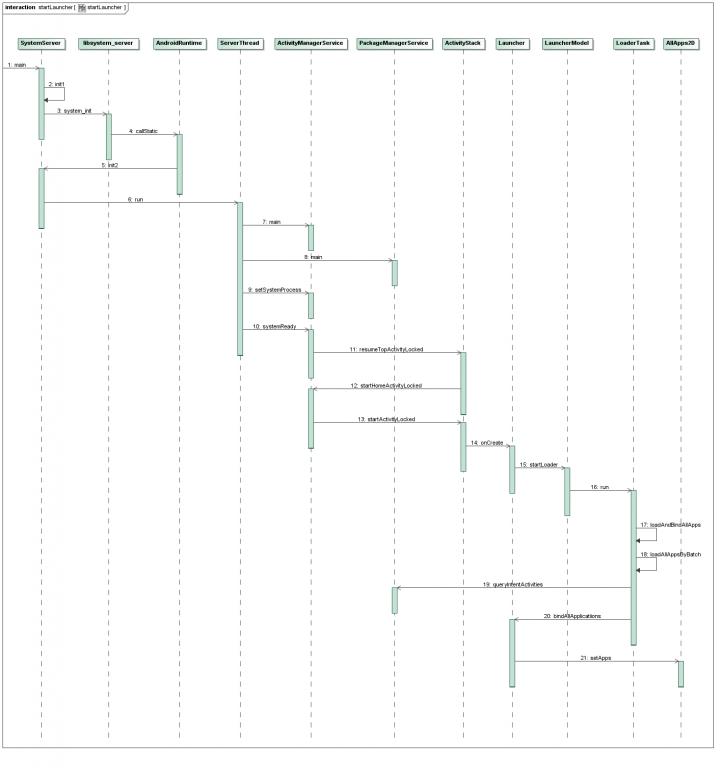
在前面一篇文章中，我们分析了Android系统在启动时安装应用程序的过程，这些应用程序安装好之后，还需要有一个Home应用程序来负责把它们在桌面上展示出来，在Android系统中，这个默认的Home应用程序就是Launcher了，本文将详细分析Launcher应用程序的启动过程。

        Android系统的Home应用程序Launcher是由ActivityManagerService启动的，而ActivityManagerService和PackageManagerService一样，都是在开机时由SystemServer组件启动的，SystemServer组件首先是启动ePackageManagerServic，由它来负责安装系统的应用程序，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)，系统中的应用程序安装好了以后，SystemServer组件接下来就要通过ActivityManagerService来启动Home应用程序Launcher了，Launcher在启动的时候便会通过PackageManagerServic把系统中已经安装好的应用程序以快捷图标的形式展示在桌面上，这样用户就可以使用这些应用程序了，整个过程如下图所示：



[点击查看大图](http://hi.csdn.net/attachment/201109/11/0_1315757033NNGT.gif)

        下面详细分析每一个步骤。

        Step 1. SystemServer.main

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 1。

        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，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 2。

        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，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 3。

        Step 4. AndroidRuntime.callStatic

        这个函数定义在frameworks/base/core/jni/AndroidRuntime.cpp文件中，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 4。

        Step 5. SystemServer.init2

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 5。

        Step 6. ServerThread.run

        这个函数定义在frameworks/base/services/java/com/android/server/SystemServer.java文件中，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 6。

        Step 7. ActivityManagerService.main

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

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class ActivityManagerService extends ActivityManagerNative
2. implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
3. ......
5. public static final Context main(int factoryTest) {
6. AThread thr = new AThread();
7. thr.start();
9. synchronized (thr) {
10. while (thr.mService == null) {
11. try {
12. thr.wait();
13. } catch (InterruptedException e) {
14. }
15. }
16. }
18. ActivityManagerService m = thr.mService;
19. mSelf = m;
20. ActivityThread at = ActivityThread.systemMain();
21. mSystemThread = at;
22. Context context = at.getSystemContext();
23. m.mContext = context;
24. m.mFactoryTest = factoryTest;
25. m.mMainStack = new ActivityStack(m, context, true);
27. m.mBatteryStatsService.publish(context);
28. m.mUsageStatsService.publish(context);
30. synchronized (thr) {
31. thr.mReady = true;
32. thr.notifyAll();
33. }
35. m.startRunning(null, null, null, null);
37. return context;
38. }
40. ......
41. }

public final class ActivityManagerService extends ActivityManagerNative

implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {

......

public static final Context main(int factoryTest) {

AThread thr = new AThread();

thr.start();

synchronized (thr) {

while (thr.mService == null) {

try {

thr.wait();

} catch (InterruptedException e) {

}

}

}

ActivityManagerService m = thr.mService;

mSelf = m;

ActivityThread at = ActivityThread.systemMain();

mSystemThread = at;

Context context = at.getSystemContext();

m.mContext = context;

m.mFactoryTest = factoryTest;

m.mMainStack = new ActivityStack(m, context, true);

m.mBatteryStatsService.publish(context);

m.mUsageStatsService.publish(context);

synchronized (thr) {

thr.mReady = true;

thr.notifyAll();

}

m.startRunning(null, null, null, null);

return context;

}

......

}

        这个函数首先通过AThread线程对象来内部创建了一个ActivityManagerService实例，然后将这个实例保存其成员变量mService中，接着又把这个ActivityManagerService实例保存在ActivityManagerService类的静态成员变量mSelf中，最后初始化其它成员变量，就结束了。

        Step 8. PackageManagerService.main

        这个函数定义在frameworks/base/services/java/com/android/server/PackageManagerService.java文件中，具体可以参考前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)的Step 7。执行完这一步之后，系统中的应用程序的所有信息都保存在PackageManagerService中了，后面Home应用程序Launcher启动起来后，就会把PackageManagerService中的应用程序信息取出来，然后以快捷图标的形式展示在桌面上，后面我们将会看到这个过程。

        Step 9. ActivityManagerService.setSystemProcess

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

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class ActivityManagerService extends ActivityManagerNative
2. implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
3. ......
5. public static void setSystemProcess() {
6. try {
7. ActivityManagerService m = mSelf;
9. ServiceManager.addService("activity", m);
10. ServiceManager.addService("meminfo", new MemBinder(m));
11. if (MONITOR\_CPU\_USAGE) {
12. ServiceManager.addService("cpuinfo", new CpuBinder(m));
13. }
14. ServiceManager.addService("permission", new PermissionController(m));
16. ApplicationInfo info =
17. mSelf.mContext.getPackageManager().getApplicationInfo(
18. "android", STOCK\_PM\_FLAGS);
19. mSystemThread.installSystemApplicationInfo(info);
21. synchronized (mSelf) {
22. ProcessRecord app = mSelf.newProcessRecordLocked(
23. mSystemThread.getApplicationThread(), info,
24. info.processName);
25. app.persistent = true;
26. app.pid = MY\_PID;
27. app.maxAdj = SYSTEM\_ADJ;
28. mSelf.mProcessNames.put(app.processName, app.info.uid, app);
29. synchronized (mSelf.mPidsSelfLocked) {
30. mSelf.mPidsSelfLocked.put(app.pid, app);
31. }
32. mSelf.updateLruProcessLocked(app, true, true);
33. }
34. } catch (PackageManager.NameNotFoundException e) {
35. throw new RuntimeException(
36. "Unable to find android system package", e);
37. }
38. }
39. ......
40. }

public final class ActivityManagerService extends ActivityManagerNative

implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {

......

public static void setSystemProcess() {

try {

ActivityManagerService m = mSelf;

ServiceManager.addService("activity", m);

ServiceManager.addService("meminfo", new MemBinder(m));

if (MONITOR\_CPU\_USAGE) {

ServiceManager.addService("cpuinfo", new CpuBinder(m));

}

ServiceManager.addService("permission", new PermissionController(m));

ApplicationInfo info =

mSelf.mContext.getPackageManager().getApplicationInfo(

"android", STOCK\_PM\_FLAGS);

mSystemThread.installSystemApplicationInfo(info);

synchronized (mSelf) {

ProcessRecord app = mSelf.newProcessRecordLocked(

mSystemThread.getApplicationThread(), info,

info.processName);

app.persistent = true;

app.pid = MY\_PID;

app.maxAdj = SYSTEM\_ADJ;

mSelf.mProcessNames.put(app.processName, app.info.uid, app);

synchronized (mSelf.mPidsSelfLocked) {

mSelf.mPidsSelfLocked.put(app.pid, app);

}

mSelf.updateLruProcessLocked(app, true, true);

}

} catch (PackageManager.NameNotFoundException e) {

throw new RuntimeException(

"Unable to find android system package", e);

}

}

......

}

        这个函数首先是将这个ActivityManagerService实例添加到ServiceManager中去托管，这样其它地方就可以通过ServiceManager.getService接口来访问这个全局唯一的ActivityManagerService实例了，接着又通过调用mSystemThread.installSystemApplicationInfo函数来把应用程序框架层下面的android包加载进来 ，这里的mSystemThread是一个ActivityThread类型的实例变量，它是在上面的Step 7中创建的，后面就是一些其它的初始化工作了。

        Step 10.  ActivityManagerService.systemReady

        这个函数是在上面的Step 6中的ServerThread.run函数在将系统中的一系列服务都初始化完毕之后才调用的，它定义在frameworks/base/services/java/com/android/server/am/ActivityManagerServcie.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class ActivityManagerService extends ActivityManagerNative
2. implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
3. ......
5. public void systemReady(final Runnable goingCallback) {
6. ......
8. synchronized (this) {
9. ......
11. mMainStack.resumeTopActivityLocked(null);
12. }
13. }
15. ......
16. }

public final class ActivityManagerService extends ActivityManagerNative

implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {

......

public void systemReady(final Runnable goingCallback) {

......

synchronized (this) {

......

mMainStack.resumeTopActivityLocked(null);

}

}

......

}

        这个函数的内容比较多，这里省去无关的部分，主要关心启动Home应用程序的逻辑，这里就是通过mMainStack.resumeTopActivityLocked函数来启动Home应用程序的了，这里的mMainStack是一个ActivityStack类型的实例变量。

        Step 11. ActivityStack.resumeTopActivityLocked

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

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class ActivityStack {
2. ......
4. final boolean resumeTopActivityLocked(ActivityRecord prev) {
5. // Find the first activity that is not finishing.
6. ActivityRecord next = topRunningActivityLocked(null);
8. ......
10. if (next == null) {
11. // There are no more activities!  Let's just start up the
12. // Launcher...
13. if (mMainStack) {
14. return mService.startHomeActivityLocked();
15. }
16. }
18. ......
19. }
21. ......
22. }

public class ActivityStack {

......

final boolean resumeTopActivityLocked(ActivityRecord prev) {

// Find the first activity that is not finishing.

ActivityRecord next = topRunningActivityLocked(null);

......

if (next == null) {

// There are no more activities! Let's just start up the

// Launcher...

if (mMainStack) {

return mService.startHomeActivityLocked();

}

}

......

}

......

}

        这里调用函数topRunningActivityLocked返回的是当前系统Activity堆栈最顶端的Activity，由于此时还没有Activity被启动过，因此，返回值为null，即next变量的值为null，于是就调用mService.startHomeActivityLocked语句，这里的mService就是前面在Step 7中创建的ActivityManagerService实例了。

        Step 12. ActivityManagerService.startHomeActivityLocked

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

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class ActivityManagerService extends ActivityManagerNative
2. implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
3. ......
5. boolean startHomeActivityLocked() {
6. ......
8. Intent intent = new Intent(
9. mTopAction,
10. mTopData != null ? Uri.parse(mTopData) : null);
11. intent.setComponent(mTopComponent);
12. if (mFactoryTest != SystemServer.FACTORY\_TEST\_LOW\_LEVEL) {
13. intent.addCategory(Intent.CATEGORY\_HOME);
14. }
15. ActivityInfo aInfo =
16. intent.resolveActivityInfo(mContext.getPackageManager(),
17. STOCK\_PM\_FLAGS);
18. if (aInfo != null) {
19. intent.setComponent(new ComponentName(
20. aInfo.applicationInfo.packageName, aInfo.name));
21. // Don't do this if the home app is currently being
22. // instrumented.
23. ProcessRecord app = getProcessRecordLocked(aInfo.processName,
24. aInfo.applicationInfo.uid);
25. if (app == null || app.instrumentationClass == null) {
26. intent.setFlags(intent.getFlags() | Intent.FLAG\_ACTIVITY\_NEW\_TASK);
27. mMainStack.startActivityLocked(null, intent, null, null, 0, aInfo,
28. null, null, 0, 0, 0, false, false);
29. }
30. }
32. return true;
33. }
35. ......
36. }

public final class ActivityManagerService extends ActivityManagerNative

implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {

......

boolean startHomeActivityLocked() {

......

Intent intent = new Intent(

mTopAction,

mTopData != null ? Uri.parse(mTopData) : null);

intent.setComponent(mTopComponent);

if (mFactoryTest != SystemServer.FACTORY\_TEST\_LOW\_LEVEL) {

intent.addCategory(Intent.CATEGORY\_HOME);

}

ActivityInfo aInfo =

intent.resolveActivityInfo(mContext.getPackageManager(),

STOCK\_PM\_FLAGS);

if (aInfo != null) {

intent.setComponent(new ComponentName(

aInfo.applicationInfo.packageName, aInfo.name));

// Don't do this if the home app is currently being

// instrumented.

ProcessRecord app = getProcessRecordLocked(aInfo.processName,

aInfo.applicationInfo.uid);

if (app == null || app.instrumentationClass == null) {

intent.setFlags(intent.getFlags() | Intent.FLAG\_ACTIVITY\_NEW\_TASK);

mMainStack.startActivityLocked(null, intent, null, null, 0, aInfo,

null, null, 0, 0, 0, false, false);

}

}

return true;

}

......

}

        函数首先创建一个CATEGORY\_HOME类型的Intent，然后通过Intent.resolveActivityInfo函数向PackageManagerService查询Category类型为HOME的Activity，这里我们假设只有系统自带的Launcher应用程序注册了HOME类型的Activity（见packages/apps/Launcher2/AndroidManifest.xml文件）：

**[html]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. **<manifest**
2. xmlns:android="http://schemas.android.com/apk/res/android"
3. package="com.android.launcher"
4. android:sharedUserId="@string/sharedUserId"
5. **>**
7. ......
9. **<application**
10. android:name="com.android.launcher2.LauncherApplication"
11. android:process="@string/process"
12. android:label="@string/application\_name"
13. android:icon="@drawable/ic\_launcher\_home"**>**
15. **<activity**
16. android:name="com.android.launcher2.Launcher"
17. android:launchMode="singleTask"
18. android:clearTaskOnLaunch="true"
19. android:stateNotNeeded="true"
20. android:theme="@style/Theme"
21. android:screenOrientation="nosensor"
22. android:windowSoftInputMode="stateUnspecified|adjustPan"**>**
23. **<intent-filter>**
24. **<action** android:name="android.intent.action.MAIN" **/>**
25. **<category** android:name="android.intent.category.HOME" **/>**
26. **<category** android:name="android.intent.category.DEFAULT" **/>**
27. **<category** android:name="android.intent.category.MONKEY"**/>**
28. **</intent-filter>**
29. **</activity>**
31. ......
32. **</application>**
33. **</manifest>**

<manifest

xmlns:android="http://schemas.android.com/apk/res/android"

package="com.android.launcher"

android:sharedUserId="@string/sharedUserId"

>

......

<application

android:name="com.android.launcher2.LauncherApplication"

android:process="@string/process"

android:label="@string/application\_name"

android:icon="@drawable/ic\_launcher\_home">

<activity

android:name="com.android.launcher2.Launcher"

android:launchMode="singleTask"

android:clearTaskOnLaunch="true"

android:stateNotNeeded="true"

android:theme="@style/Theme"

android:screenOrientation="nosensor"

android:windowSoftInputMode="stateUnspecified|adjustPan">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.HOME" />

<category android:name="android.intent.category.DEFAULT" />

<category android:name="android.intent.category.MONKEY"/>

</intent-filter>

</activity>

......

</application>

</manifest>

        因此，这里就返回com.android.launcher2.Launcher这个Activity了。由于是第一次启动这个Activity，接下来调用函数getProcessRecordLocked返回来的ProcessRecord值为null，于是，就调用mMainStack.startActivityLocked函数启动com.android.launcher2.Launcher这个Activity了，这里的mMainStack是一个ActivityStack类型的成员变量。

        Step 13.  ActivityStack.startActivityLocked

        这个函数定义在frameworks/base/services/java/com/android/server/am/ActivityStack.java文件中，具体可以参考[Android应用程序启动过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6689748)一文，这里就不详述了，在我们这个场景中，从这步往下经过层层调用，这个函数的最后结果就是把com.android.launcher2.Launcher启动起来，接着调用它的onCreate函数。（在[Android应用程序启动过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6689748)中是从step8-step35）

        Step 14. Launcher.onCreate

        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/Launcher.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class Launcher extends Activity
2. implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {
3. ......
5. @Override
6. protected void onCreate(Bundle savedInstanceState) {
7. ......
9. if (!mRestoring) {
10. mModel.startLoader(this, true);
11. }
13. ......
14. }
16. ......
17. }

public final class Launcher extends Activity

implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {

......

@Override

protected void onCreate(Bundle savedInstanceState) {

......

if (!mRestoring) {

mModel.startLoader(this, true);

}

......

}

......

}

        这里的mModel是一个LauncherModel类型的成员变量，这里通过调用它的startLoader成员函数来执行加应用程序的操作。

        Step 15. LauncherModel.startLoader

        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/LauncherModel.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class LauncherModel extends BroadcastReceiver {
2. ......
4. public void startLoader(Context context, boolean isLaunching) {
5. ......
7. synchronized (mLock) {
8. ......
10. // Don't bother to start the thread if we know it's not going to do anything
11. if (mCallbacks != null && mCallbacks.get() != null) {
12. // If there is already one running, tell it to stop.
13. LoaderTask oldTask = mLoaderTask;
14. if (oldTask != null) {
15. if (oldTask.isLaunching()) {
16. // don't downgrade isLaunching if we're already running
17. isLaunching = true;
18. }
19. oldTask.stopLocked();
20. }
21. mLoaderTask = new LoaderTask(context, isLaunching);
22. sWorker.post(mLoaderTask);
23. }
24. }
25. }
27. ......
28. }

public class LauncherModel extends BroadcastReceiver {

......

public void startLoader(Context context, boolean isLaunching) {

......

synchronized (mLock) {

......

// Don't bother to start the thread if we know it's not going to do anything

if (mCallbacks != null && mCallbacks.get() != null) {

// If there is already one running, tell it to stop.

LoaderTask oldTask = mLoaderTask;

if (oldTask != null) {

if (oldTask.isLaunching()) {

// don't downgrade isLaunching if we're already running

isLaunching = true;

}

oldTask.stopLocked();

}

mLoaderTask = new LoaderTask(context, isLaunching);

sWorker.post(mLoaderTask);

}

}

}

......

}

        这里不是直接加载应用程序，而是把加载应用程序的操作作为一个消息来处理。这里的sWorker是一个Handler，通过它的post方式把一个消息放在消息队列中去，然后系统就会调用传进去的参数mLoaderTask的run函数来处理这个消息，这个mLoaderTask是LoaderTask类型的实例，于是，下面就会执行LoaderTask类的run函数了。

        Step 16. LoaderTask.run

        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/LauncherModel.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class LauncherModel extends BroadcastReceiver {
2. ......
4. private class LoaderTask implements Runnable {
5. ......
7. public void run() {
8. ......
10. keep\_running: {
11. ......
13. // second step
14. if (loadWorkspaceFirst) {
15. ......
16. loadAndBindAllApps();
17. } else {
18. ......
19. }
21. ......
22. }
24. ......
25. }
27. ......
28. }
30. ......
31. }

public class LauncherModel extends BroadcastReceiver {

......

private class LoaderTask implements Runnable {

......

public void run() {

......

keep\_running: {

......

// second step

if (loadWorkspaceFirst) {

......

loadAndBindAllApps();

} else {

......

}

......

}

......

}

......

}

......

}

        这里调用loadAndBindAllApps成员函数来进一步操作。

        Step 17. LoaderTask.loadAndBindAllApps  
        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/LauncherModel.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class LauncherModel extends BroadcastReceiver {
2. ......
4. private class LoaderTask implements Runnable {
5. ......
7. private void loadAndBindAllApps() {
8. ......
10. if (!mAllAppsLoaded) {
11. loadAllAppsByBatch();
12. if (mStopped) {
13. return;
14. }
15. mAllAppsLoaded = true;
16. } else {
17. onlyBindAllApps();
18. }
19. }

22. ......
23. }
25. ......
26. }

public class LauncherModel extends BroadcastReceiver {

......

private class LoaderTask implements Runnable {

......

private void loadAndBindAllApps() {

......

if (!mAllAppsLoaded) {

loadAllAppsByBatch();

if (mStopped) {

return;

}

mAllAppsLoaded = true;

} else {

onlyBindAllApps();

}

}

......

}

......

}

        由于还没有加载过应用程序，这里的mAllAppsLoaded为false，于是就继续调用loadAllAppsByBatch函数来进一步操作了。

        Step 18. LoaderTask.loadAllAppsByBatch  
        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/LauncherModel.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class LauncherModel extends BroadcastReceiver {
2. ......
4. private class LoaderTask implements Runnable {
5. ......
7. private void loadAllAppsByBatch() {
8. ......
10. final Intent mainIntent = new Intent(Intent.ACTION\_MAIN, null);
11. mainIntent.addCategory(Intent.CATEGORY\_LAUNCHER);
13. final PackageManager packageManager = mContext.getPackageManager();
14. List<ResolveInfo> apps = null;
16. int N = Integer.MAX\_VALUE;
18. int startIndex;
19. int i=0;
20. int batchSize = -1;
21. while (i < N && !mStopped) {
22. if (i == 0) {
23. mAllAppsList.clear();
24. ......
25. apps = packageManager.queryIntentActivities(mainIntent, 0);
27. ......
29. N = apps.size();
31. ......
33. if (mBatchSize == 0) {
34. batchSize = N;
35. } else {
36. batchSize = mBatchSize;
37. }
39. ......
41. Collections.sort(apps,
42. new ResolveInfo.DisplayNameComparator(packageManager));
43. }
45. startIndex = i;
46. for (int j=0; i<N && j<batchSize; j++) {
47. // This builds the icon bitmaps.
48. mAllAppsList.add(new ApplicationInfo(apps.get(i), mIconCache));
49. i++;
50. }
52. final boolean first = i <= batchSize;
53. final Callbacks callbacks = tryGetCallbacks(oldCallbacks);
54. final ArrayList<ApplicationInfo> added = mAllAppsList.added;
55. mAllAppsList.added = new ArrayList<ApplicationInfo>();
57. mHandler.post(new Runnable() {
58. public void run() {
59. final long t = SystemClock.uptimeMillis();
60. if (callbacks != null) {
61. if (first) {
62. callbacks.bindAllApplications(added);
63. } else {
64. callbacks.bindAppsAdded(added);
65. }
66. ......
67. } else {
68. ......
69. }
70. }
71. });
73. ......
74. }
76. ......
77. }
79. ......
80. }
82. ......
83. }

public class LauncherModel extends BroadcastReceiver {

......

private class LoaderTask implements Runnable {

......

private void loadAllAppsByBatch() {

......

final Intent mainIntent = new Intent(Intent.ACTION\_MAIN, null);

mainIntent.addCategory(Intent.CATEGORY\_LAUNCHER);

final PackageManager packageManager = mContext.getPackageManager();

List<ResolveInfo> apps = null;

int N = Integer.MAX\_VALUE;

int startIndex;

int i=0;

int batchSize = -1;

while (i < N && !mStopped) {

if (i == 0) {

mAllAppsList.clear();

......

apps = packageManager.queryIntentActivities(mainIntent, 0);

......

N = apps.size();

......

if (mBatchSize == 0) {

batchSize = N;

} else {

batchSize = mBatchSize;

}

......

Collections.sort(apps,

new ResolveInfo.DisplayNameComparator(packageManager));

}

startIndex = i;

for (int j=0; i<N && j<batchSize; j++) {

// This builds the icon bitmaps.

mAllAppsList.add(new ApplicationInfo(apps.get(i), mIconCache));

i++;

}

final boolean first = i <= batchSize;

final Callbacks callbacks = tryGetCallbacks(oldCallbacks);

final ArrayList<ApplicationInfo> added = mAllAppsList.added;

mAllAppsList.added = new ArrayList<ApplicationInfo>();

mHandler.post(new Runnable() {

public void run() {

final long t = SystemClock.uptimeMillis();

if (callbacks != null) {

if (first) {

callbacks.bindAllApplications(added);

} else {

callbacks.bindAppsAdded(added);

}

......

} else {

......

}

}

});

......

}

......

}

......

}

......

}

        函数首先构造一个CATEGORY\_LAUNCHER类型的Intent：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. final Intent mainIntent = new Intent(Intent.ACTION\_MAIN, null);
2. mainIntent.addCategory(Intent.CATEGORY\_LAUNCHER);

final Intent mainIntent = new Intent(Intent.ACTION\_MAIN, null);

mainIntent.addCategory(Intent.CATEGORY\_LAUNCHER);

        接着从mContext变量中获得PackageManagerService的接口：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. final PackageManager packageManager = mContext.getPackageManager();

final PackageManager packageManager = mContext.getPackageManager();

       下一步就是通过这个PackageManagerService.queryIntentActivities接口来取回所有Action类型为Intent.ACTION\_MAIN，并且Category类型为Intent.CATEGORY\_LAUNCHER的Activity了。

       我们先进入到PackageManagerService.queryIntentActivities函数中看看是如何获得这些Activity的，然后再回到这个函数中来看其余操作。

       Step 19. PackageManagerService.queryIntentActivities

       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/6767736)

1. class PackageManagerService extends IPackageManager.Stub {
2. ......
4. public List<ResolveInfo> queryIntentActivities(Intent intent,
5. String resolvedType, int flags) {
6. ......
8. synchronized (mPackages) {
9. String pkgName = intent.getPackage();
10. if (pkgName == null) {
11. return (List<ResolveInfo>)mActivities.queryIntent(intent,
12. resolvedType, flags);
13. }
15. ......
16. }
18. ......
19. }
21. ......
22. }

class PackageManagerService extends IPackageManager.Stub {

......

public List<ResolveInfo> queryIntentActivities(Intent intent,

String resolvedType, int flags) {

......

synchronized (mPackages) {

String pkgName = intent.getPackage();

if (pkgName == null) {

return (List<ResolveInfo>)mActivities.queryIntent(intent,

resolvedType, flags);

}

......

}

......

}

......

}

        回忆前面一篇文章[Android应用程序安装过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6766010)，系统在前面的Step 8中启动PackageManagerService时，会把系统中的应用程序都解析一遍，然后把解析得到的Activity都保存在mActivities变量中，这里通过这个mActivities变量的queryIntent函数返回符合条件intent的Activity，这里要返回的便是Action类型为Intent.ACTION\_MAIN，并且Category类型为Intent.CATEGORY\_LAUNCHER的Activity了。

        回到Step 18中的 LoaderTask.loadAllAppsByBatch函数中，从queryIntentActivities函数调用处返回所要求的Activity后，便调用函数tryGetCallbacks(oldCallbacks)得到一个返CallBack接口，这个接口是由Launcher类实现的，接着调用这个接口的.bindAllApplications函数来进一步操作。注意，这里又是通过消息来处理加载应用程序的操作的。

        Step 20. Launcher.bindAllApplications

        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/Launcher.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class Launcher extends Activity
2. implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {
3. ......
5. private AllAppsView mAllAppsGrid;
7. ......
9. public void bindAllApplications(ArrayList<ApplicationInfo> apps) {
10. mAllAppsGrid.setApps(apps);
11. }
13. ......
14. }

public final class Launcher extends Activity

implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {

......

private AllAppsView mAllAppsGrid;

......

public void bindAllApplications(ArrayList<ApplicationInfo> apps) {

mAllAppsGrid.setApps(apps);

}

......

}

        这里的mAllAppsGrid是一个AllAppsView类型的变量，它的实际类型一般就是AllApps2D了。

        Step 21. AllApps2D.setApps（这步与实际代码有些差异）

        这个函数定义在packages/apps/Launcher2/src/com/android/launcher2/AllApps2D.java文件中：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class AllApps2D
2. extends RelativeLayout
3. implements AllAppsView,
4. AdapterView.OnItemClickListener,
5. AdapterView.OnItemLongClickListener,
6. View.OnKeyListener,
7. DragSource {
9. ......
11. public void setApps(ArrayList<ApplicationInfo> list) {
12. mAllAppsList.clear();
13. addApps(list);
14. }
16. public void addApps(ArrayList<ApplicationInfo> list) {
17. final int N = list.size();
19. for (int i=0; i<N; i++) {
20. final ApplicationInfo item = list.get(i);
21. int index = Collections.binarySearch(mAllAppsList, item,
22. LauncherModel.APP\_NAME\_COMPARATOR);
23. if (index < 0) {
24. index = -(index+1);
25. }
26. mAllAppsList.add(index, item);
27. }
28. mAppsAdapter.notifyDataSetChanged();
29. }
31. ......
32. }

public class AllApps2D

extends RelativeLayout

implements AllAppsView,

AdapterView.OnItemClickListener,

AdapterView.OnItemLongClickListener,

View.OnKeyListener,

DragSource {

......

public void setApps(ArrayList<ApplicationInfo> list) {

mAllAppsList.clear();

addApps(list);

}

public void addApps(ArrayList<ApplicationInfo> list) {

final int N = list.size();

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

final ApplicationInfo item = list.get(i);

int index = Collections.binarySearch(mAllAppsList, item,

LauncherModel.APP\_NAME\_COMPARATOR);

if (index < 0) {

index = -(index+1);

}

mAllAppsList.add(index, item);

}

mAppsAdapter.notifyDataSetChanged();

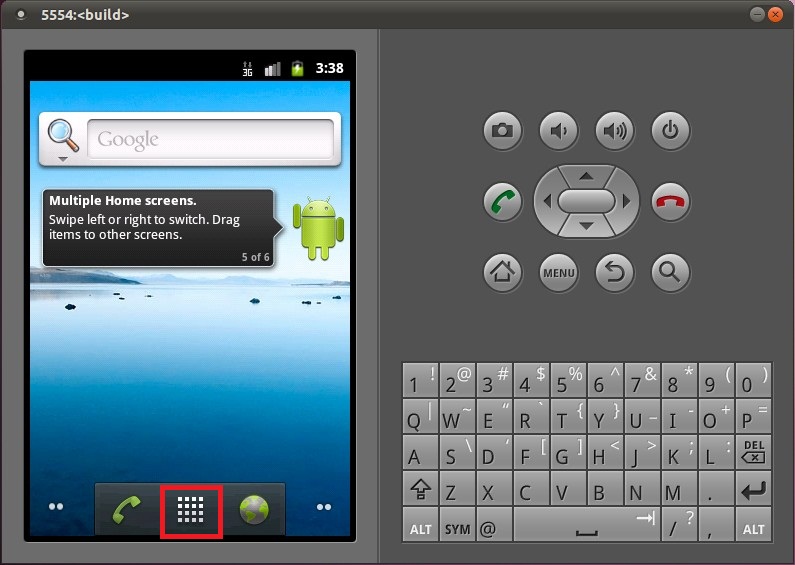
}

......

}

        函数setApps首先清空mAllAppsList列表，然后调用addApps函数来为上一步得到的每一个应用程序创建一个ApplicationInfo实例了，有了这些ApplicationInfo实例之后，就可以在桌面上展示系统中所有的应用程序了。

        到了这里，系统默认的Home应用程序Launcher就把PackageManagerService中的应用程序加载进来了，当我们在屏幕上点击下面这个图标时，就会把刚才加载好的应用程序以图标的形式展示出来了：



        点击这个按钮时，便会响应Launcher.onClick函数：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class Launcher extends Activity
2. implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {
3. ......
5. public void onClick(View v) {
6. Object tag = v.getTag();
7. if (tag instanceof ShortcutInfo) {
8. ......
9. } else if (tag instanceof FolderInfo) {
10. ......
11. } else if (v == mHandleView) {
12. if (isAllAppsVisible()) {
13. ......
14. } else {
15. showAllApps(true);
16. }
17. }
18. }
20. ......
21. }

public final class Launcher extends Activity

implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {

......

public void onClick(View v) {

Object tag = v.getTag();

if (tag instanceof ShortcutInfo) {

......

} else if (tag instanceof FolderInfo) {

......

} else if (v == mHandleView) {

if (isAllAppsVisible()) {

......

} else {

showAllApps(true);

}

}

}

......

}

        接着就会调用showAllApps函数显示应用程序图标：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public final class Launcher extends Activity
2. implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {
3. ......
5. void showAllApps(boolean animated) {
6. mAllAppsGrid.zoom(1.0f, animated);
8. ((View) mAllAppsGrid).setFocusable(true);
9. ((View) mAllAppsGrid).requestFocus();
11. // TODO: fade these two too
12. mDeleteZone.setVisibility(View.GONE);
13. }
15. ......
16. }

public final class Launcher extends Activity

implements View.OnClickListener, OnLongClickListener, LauncherModel.Callbacks, AllAppsView.Watcher {

......

void showAllApps(boolean animated) {

mAllAppsGrid.zoom(1.0f, animated);

((View) mAllAppsGrid).setFocusable(true);

((View) mAllAppsGrid).requestFocus();

// TODO: fade these two too

mDeleteZone.setVisibility(View.GONE);

}

......

}

        这样我们就可以看到系统中的应用程序了：

  
  
        当点击上面的这些应用程序图标时，便会响应AllApps2D.onItemClick函数：

**[java]** [view plaincopyprint?](http://blog.csdn.net/luoshengyang/article/details/6767736)

1. public class AllApps2D
2. extends RelativeLayout
3. implements AllAppsView,
4. AdapterView.OnItemClickListener,
5. AdapterView.OnItemLongClickListener,
6. View.OnKeyListener,
7. DragSource {
9. ......
11. public void onItemClick(AdapterView parent, View v, int position, long id) {
12. ApplicationInfo app = (ApplicationInfo) parent.getItemAtPosition(position);
13. mLauncher.startActivitySafely(app.intent, app);
14. }

17. ......
18. }

public class AllApps2D

extends RelativeLayout

implements AllAppsView,

AdapterView.OnItemClickListener,

AdapterView.OnItemLongClickListener,

View.OnKeyListener,

DragSource {

......

public void onItemClick(AdapterView parent, View v, int position, long id) {

ApplicationInfo app = (ApplicationInfo) parent.getItemAtPosition(position);

mLauncher.startActivitySafely(app.intent, app);

}

......

}

        这里的成员变量mLauncher的类型为Launcher，于是就调用Launcher.startActivitySafely函数来启动应用程序了，这个过程具体可以参考[Android应用程序启动过程源代码分析](http://blog.csdn.net/luoshengyang/article/details/6689748)一文。