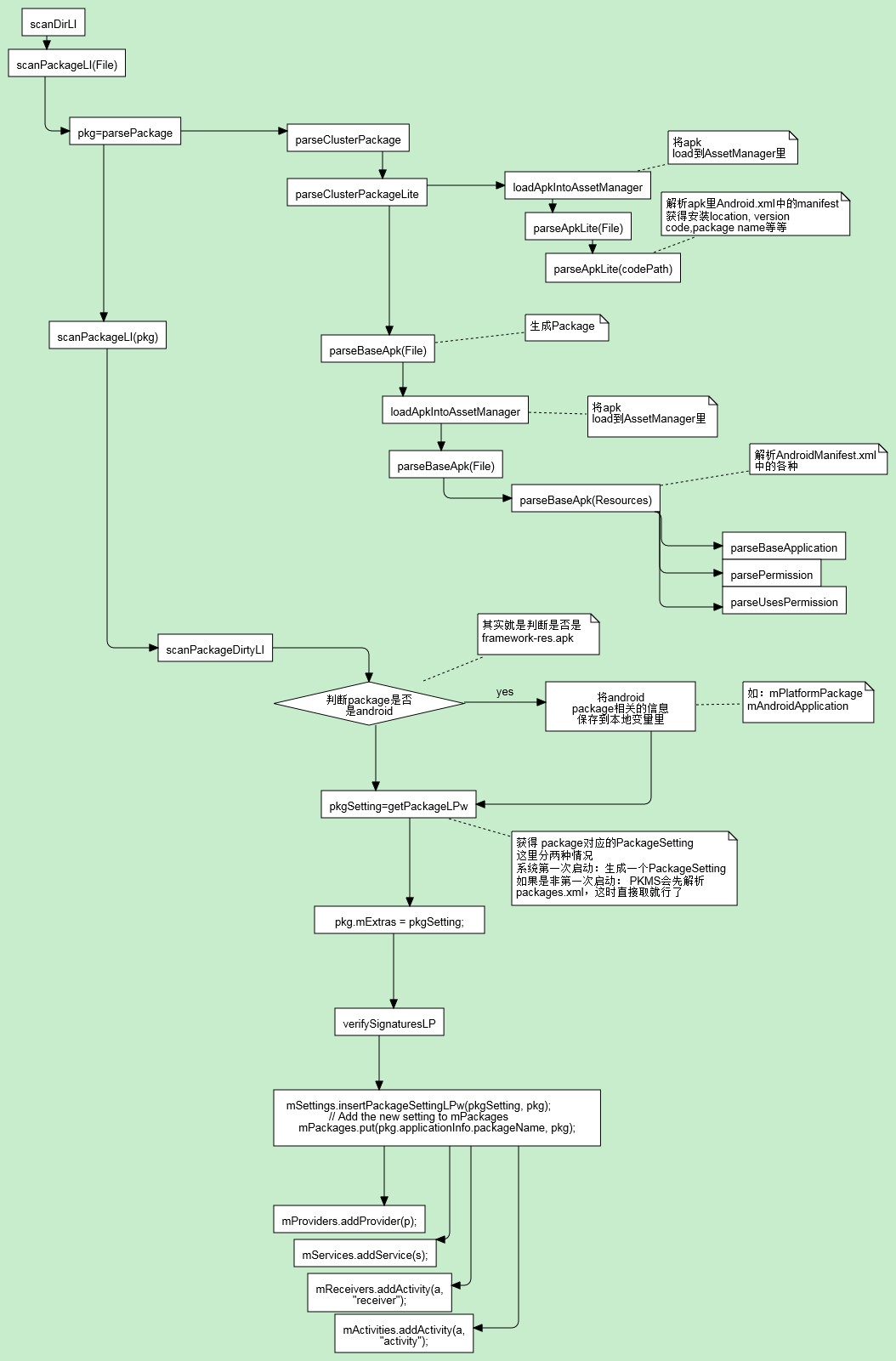
Android 组件图

一、apk解析流程



从 scanDirLI可以看出，解析android组件的函数是由 parseBaseApk(Resource …)实现的

//每个apk都对应一个Package, 为每个apk生成一个Package

**final** Package pkg = **new** Package(pkgName);

//解析 <manifest> 获得package的version code/version name/shared uid …

pkg.mVersionCode = pkg.applicationInfo.versionCode= sa.getInteger(  
 com.android.internal.R.styleable.AndroidManifest\_versionCode, 0);

//解析 <application>

**boolean** parseBaseApplication(Package owner, …) {

//构造出application的className

ai.className = *buildClassName*(pkgName, name, outError);

//构造 processName, 这个app会默认运行在哪个进程里, 如果 process以”:” 开头，那么 processName将会构造成 package\_name + processName, 如果是以非:开头，那 processName直接为process所定义的值, 如package name为 com.bobby.example, process: “:hello”, 那么构造出来的processName为 com.bobby.example:hello

pname = sa.getNonConfigurationString(  
 com.android.internal.R.styleable.***AndroidManifestApplication\_process***,  
 Configuration.***NATIVE\_CONFIG\_VERSION***);

ai.processName = *buildProcessName*(ai.packageName, **null**, pname,  
 flags, mSeparateProcesses, outError);

//解析 activity

Activity a = parseActivity();

owner.activities.add(a);

//解析 alias activity

Activity a = parseActivityAlias();

owner.activities.add(a);

//解析 receiver, 可见 receiver和activity是用同一个类来表示

Activity a = parseActivity();

owner.receivers.add(a);

//解析 service

Service s = parseService();

owner.services.add(s);

//解析provider

Provider p = parseProvider();

owner.providers.add(p);

//解析library

owner.libraryNames = ArrayUtils.*add*(owner.libraryNames, lname);

//解析 use-library

owner.usesLibraries = ArrayUtils.*add*(owner.usesLibraries, lname);

}

以activity的解析过程为例

Activity parseActivity(Package owner, …) {

//生成Activity, 并继续解析activity, 这里的info是 ActivityInfo类

Activity a = **new** Activity(mParseActivityArgs, **new** ActivityInfo());

a.info.launchMode = sa.getInt(  
 R.styleable.AndroidManifestActivity\_launchMode, ActivityInfo.***LAUNCH\_MULTIPLE***);

a.info.softInputMode = sa.getInt(  
 R.styleable.AndroidManifestActivity\_windowSoftInputMode, 0);

//解析 intent-filter, 这个可以定义由哪些intent启动

**if** (parser.getName().equals("intent-filter")) {  
 ActivityIntentInfo intent = **new** ActivityIntentInfo(a);  
 **if** (!parseIntent(res, parser, attrs, **true**, **true**, intent, outError)) {  
 **return null**;  
 }

a.intents.add(intent); // 加入到intents里

}

}

同理可以分析 service/provider的解析

但是解析 activity-alias 会有稍许不一样

如：

<activity-alias android:name="EmergencyOutgoingCallBroadcaster"

android:enabled="false"

android:targetActivity="OutgoingCallBroadcaster"

android:permission="android.permission.CALL\_PRIVILEGED">

<intent-filter android:priority="1000">

<action android:name="android.intent.action.CALL\_EMERGENCY" />

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

<data android:scheme="tel" />

Activity parseActivityAlias(Package owner, …) {

//根据android:targetActivity 找出target activity的别名，然后找出它对应的Activity.

String targetActivity = sa.getNonConfigurationString(  
 com.android.internal.R.styleable.***AndroidManifestActivityAlias\_targetActivity***,  
 Configuration.***NATIVE\_CONFIG\_VERSION***);  
  
targetActivity = *buildClassName*(owner.applicationInfo.packageName,  
 targetActivity, outError);  
  
Activity target = **null**;  
  
**final int** NA = owner.activities.size();  
**for** (**int** i=0; i<NA; i++) {  
 Activity t = owner.activities.get(i);  
 **if** (targetActivity.equals(t.info.name)) {  
 target = t;  
 **break**;  
 }  
}

//将target的ActivityInfo信息赋值给alias的activity

ActivityInfo info = **new** ActivityInfo();

info.targetActivity = targetActivity;  
info.configChanges = target.info.configChanges;

…

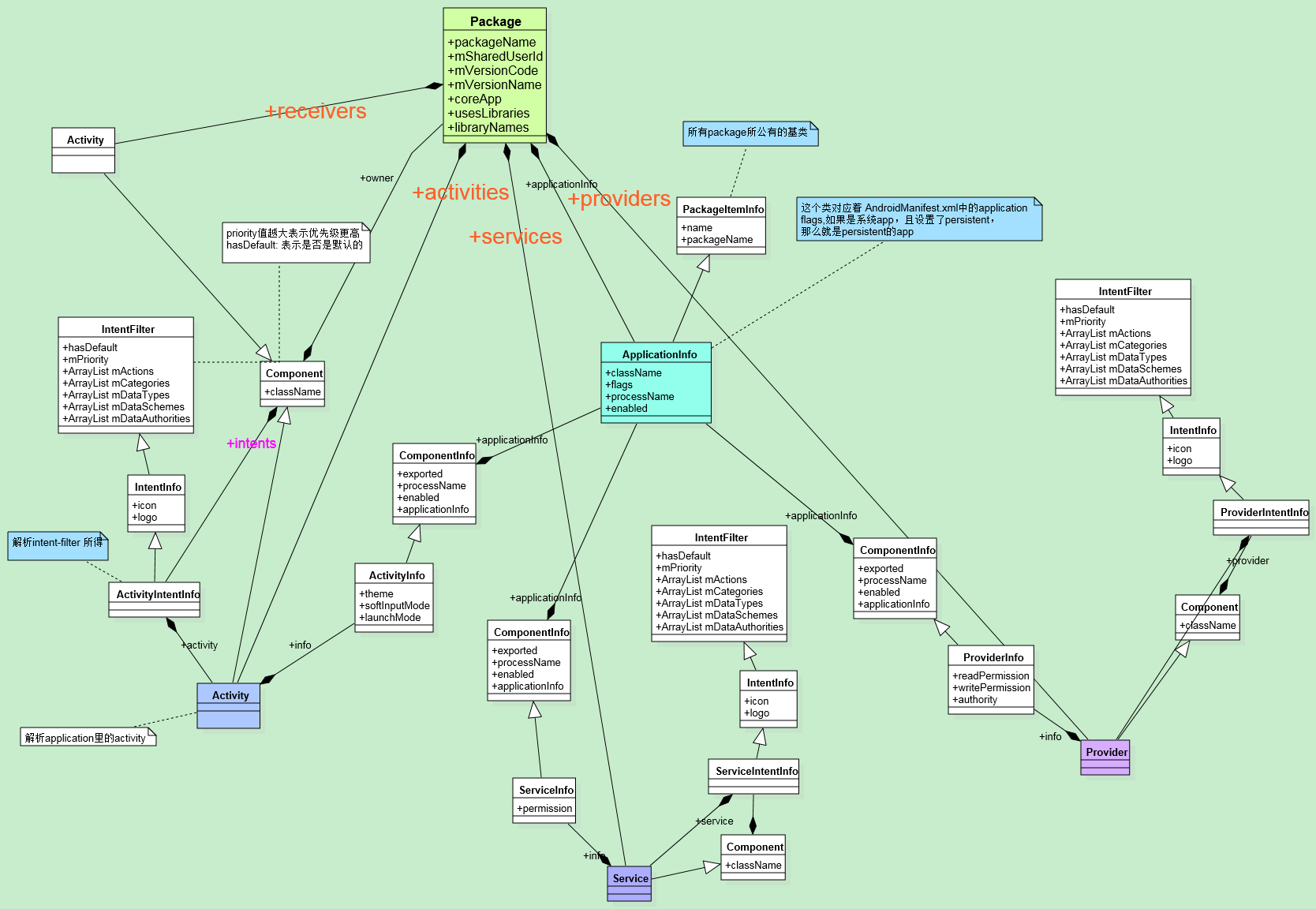
//接着解析该 <activity-alias>，这个就和解析activity一样了，

//最后加入到package里的activities list中

}

可以看出来 activity-alias尽管是用的同一个Activity代码，但是通过 activity-alias后，实际上是新创建了一个Activity. 只不过里面的某些字段可能会有限制了。

最后解析后的android component图为



二、将所有的组件加入到PKMS里

在 scanPackageDirtyLI的最后，会将所有的activity/provider/service/receiver加入到PKMS里。

**==============================================================================**

**int** N = pkg.providers.size();  
StringBuilder r = **null**;  
**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);  
 mProviders.addProvider(p);

}

**public final void** addProvider(PackageParser.Provider p) {  
 **if** (mProviders.containsKey(p.getComponentName())) {  
 Slog.*w*(***TAG***, "Provider " + p.getComponentName() + " already defined; ignoring");  
 **return**;  
 }  
  
 mProviders.put(p.getComponentName(), p);

//加入所有的provider的intent-filter到mFilters里  
 **final int** NI = p.intents.size();  
 **int** j;  
 **for** (j = 0; j < NI; j++) {  
 PackageParser.ProviderIntentInfo intent = p.intents.get(j);  
 addFilter(intent);  
 }  
}

**==============================================================================**

N = pkg.services.size();  
r = **null**;  
**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");

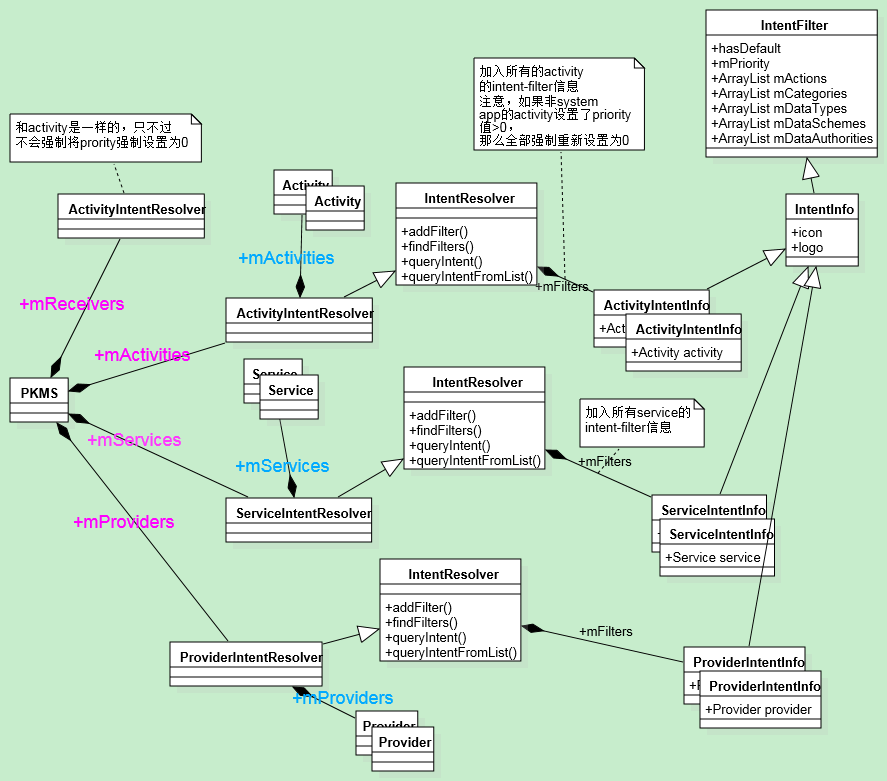
}

**==============================================================================**

N = pkg.activities.size();  
r = **null**;  
**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");

}

PKMS与组件相关的uml图



三、PKMS如何查询组件

AMS最后都会调用到 packagemanagerservice里的queryIntentActivities

**public** List<ResolveInfo> queryIntentActivities(Intent intent,  
 String resolvedType, **int** flags, **int** userId)

该函数分为三种情况

//scenario 1, 有component的情况

一般是 new Intent(TestaActivity.this, TestbActivity.class)这种情况

**public** Intent(Context packageContext, Class<?> cls) {  
 mComponent = **new** ComponentName(packageContext, cls);  
}

ComponentName comp = intent.getComponent();  
**if** (comp != **null**) {  
 **final** List<ResolveInfo> list = **new** ArrayList<ResolveInfo>(1);  
 **final** ActivityInfo ai = getActivityInfo(comp, flags, userId);  
 **if** (ai != **null**) {  
 **final** ResolveInfo ri = **new** ResolveInfo();  
 ri.activityInfo = ai;  
 list.add(ri);  
 }  
 **return** list;  
}

**public** ActivityInfo getActivityInfo(ComponentName component, **int** flags, **int** userId) {  
  
 **synchronized** (mPackages) {  
 PackageParser.Activity a = mActivities.mActivities.get(component);  
  
 **if** (a != **null** && mSettings.isEnabledLPr(a.info, flags, userId)) {  
 PackageSetting ps = mSettings.mPackages.get(component.getPackageName());  
 **if** (ps == **null**) **return null**;  
 **return** PackageParser.*generateActivityInfo*(a, flags, ps.readUserState(userId),  
 userId);  
 }  
 }  
 **return null**;  
}

**public static final** ActivityInfo generateActivityInfo(Activity a, **int** flags,  
 PackageUserState state, **int** userId) {  
 **if** (a == **null**) **return null**;

// 如果可以不需要copy的话，直接返回  
 **if** (!*copyNeeded*(flags, a.owner, state, a.metaData, userId)) {  
 **return** a.info;  
 }

//如果需要copy的话，就生成一个新的对象  
 *// Make shallow copies so we can store the metadata safely* ActivityInfo ai = **new** ActivityInfo(a.info);  
 ai.metaData = a.metaData;  
 ai.applicationInfo = *generateApplicationInfo*(a.owner, flags, state, userId);  
 **return** ai;  
}

可以看出 scenario 1的是直接从 PKMS.mActivities.mActivities里拿到的。

Scenario 2, 非指定具体的class

*// Check for results in the current profile.*List<ResolveInfo> result = mActivities.queryIntent(  
 intent, resolvedType, flags, userId);

**public** List<R> queryIntent(Intent intent, String resolvedType, **boolean** defaultOnly,  
 **int** userId) {

//根据 intent里的action在 mActionToFilter里查找是否有对应的ActivityIntentInfo, 并组装成一个数组，用于后面的buildResolvedList使用**if** (resolvedType == **null** && scheme == **null** && intent.getAction() != **null**) {  
 firstTypeCut = mActionToFilter.get(intent.getAction());  
 **if** (debug) Slog.*v*(***TAG***, "Action list: " + Arrays.*toString*(firstTypeCut));  
}  
  
FastImmutableArraySet<String> categories = *getFastIntentCategories*(intent);  
**if** (firstTypeCut != **null**) {  
 buildResolveList(intent, categories, debug, defaultOnly,  
 resolvedType, scheme, firstTypeCut, finalList, userId);  
}  
**if** (secondTypeCut != **null**) {  
 buildResolveList(intent, categories, debug, defaultOnly,  
 resolvedType, scheme, secondTypeCut, finalList, userId);  
}  
**if** (thirdTypeCut != **null**) {  
 buildResolveList(intent, categories, debug, defaultOnly,  
 resolvedType, scheme, thirdTypeCut, finalList, userId);  
}  
**if** (schemeCut != **null**) {  
 buildResolveList(intent, categories, debug, defaultOnly,  
 resolvedType, scheme, schemeCut, finalList, userId);  
}

同理可以分析出 secondTypeCut/thirdTypeCut/schemeCut …

Scenario 3, 如果 intent中的package name存在的情况.

final PackageParser.Package pkg = mPackages.get(pkgName);  
if (pkg != null) {  
 return filterIfNotPrimaryUser(  
 mActivities.queryIntentForPackage(  
 intent, resolvedType, flags, pkg.activities, userId),  
 userId);  
}  
return new ArrayList<ResolveInfo>();

//查询该package里所有的activity intent-filter

public List<ResolveInfo> queryIntentForPackage(Intent intent, String resolvedType,  
 int flags, ArrayList<PackageParser.Activity> packageActivities, int userId) {  
 if (!sUserManager.exists(userId)) return null;  
 if (packageActivities == null) {  
 return null;  
 }  
 mFlags = flags;  
 final boolean defaultOnly = (flags&PackageManager.MATCH\_DEFAULT\_ONLY) != 0;  
 final int N = packageActivities.size();  
 ArrayList<PackageParser.ActivityIntentInfo[]> listCut =  
 new ArrayList<PackageParser.ActivityIntentInfo[]>(N);  
  
 ArrayList<PackageParser.ActivityIntentInfo> intentFilters;  
 for (int i = 0; i < N; ++i) {  
 intentFilters = packageActivities.get(i).intents;  
 if (intentFilters != null && intentFilters.size() > 0) {  
 PackageParser.ActivityIntentInfo[] array =  
 new PackageParser.ActivityIntentInfo[intentFilters.size()];  
 intentFilters.toArray(array);  
 listCut.add(array);  
 }  
 }  
 return super.queryIntentFromList(intent, resolvedType, defaultOnly, listCut, userId);  
}

特别注意， queryIntent 和 queryIntentFromList都会将查询出来的 list， 进行priority 由大到小进行排序

Collections.*sort*(results, ***mResolvePrioritySorter***);

**private static final** Comparator mResolvePrioritySorter = **new** Comparator() {  
 **public int** compare(Object o1, Object o2) {  
 **final int** q1 = ((IntentFilter) o1).getPriority();  
 **final int** q2 = ((IntentFilter) o2).getPriority();  
 **return** (q1 > q2) ? -1 : ((q1 < q2) ? 1 : 0);  
 }  
};

注意如果是非system app，当哪个intent-filters里定义了该module的priority时，PKMS会强制将它们的priority转换为 0, 可以非常好理解，因为当查询一个组件时，查询出来的list是按照priority进行由大到小进行排序的， 比如 getHomeIntent 去获得home的activity时，最后后调用 chooseBestActivity 进行选择最适合的activity, 如果自己的应用能够定义priority的话，那么自己写的apk可以随便定义priority了，这样就会导致系统被 hack 了