**Launcher Design Doc**

**编 写 人： 梁晓玲**

版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 作者 | 操作 | 日期 | 说明 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 概述 3](#_Toc404672931)

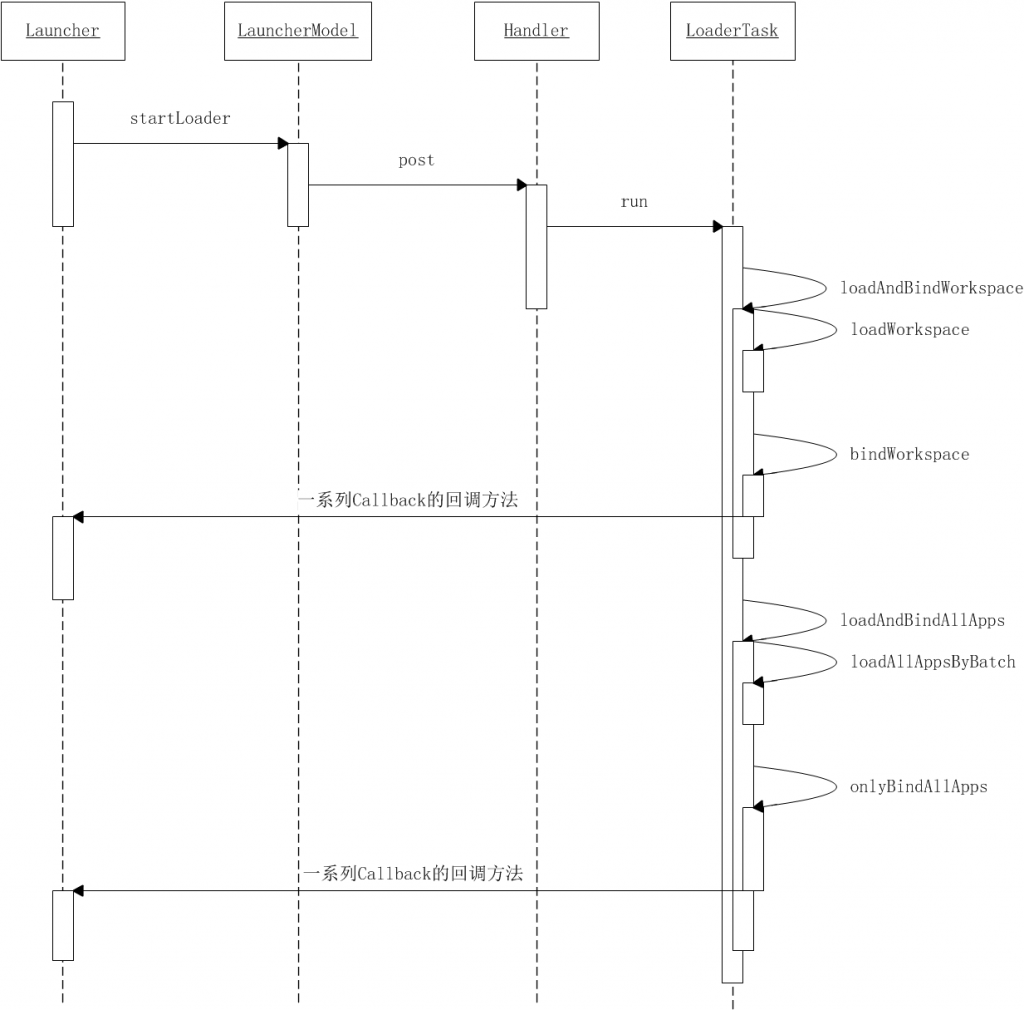
[2 类图，框架 4](#_Toc404672932)

[3 代码分析 4](#_Toc404672933)

### 1 概述

Widget的加载流程，以及简单系统widget的解析。

### 2 类图，框架



网上拷的一个加载流程图。

[Android开机时桌面Widget的载入流程](http://www.apkbus.com/android-55738-1-1.html)：

AndroidWidgetService.java是Widget framework的核心类，承担了所有的Widget的管理工作。  
开机之后在system server启动之后（此进程会管理其他Service）  
会创建AppWidgetService的实例，并调用AppWidgetService的SystemReady方法  
在此方法中完成了Widget的初始化和载入工作，主要是做了三件事情  
1.遍历所有安装包，找到符合条件的（ACTION=ACTION\_APPWIDGET\_UPDATE）和<meta\_data android:name="android.appwidget.provider"/>的Intent receiver  
   解析相关信息并保存到本地变量中。  
2.读取文本数据 /data/system/appwidgets.xml, 所有已安装到桌面的widget信息都保存在这个文件中，并把结果保存到本地。  
3.注册接收三个消息：ACTION\_BOOT\_COMPLETED（系统启动到launcher时会广播此消息）、ACTION\_PACKAGE\_ADDED（新的APK安装到系统时会广播此消息）、ACTION\_PACKAGE\_REMOVER（有应用被删除时会广播此消息）  
当系统启动到桌面后，AppWidgetService接收到ACTION\_BOOT\_COMPLETED消息，它会去检查本地数据，如果有已经安装到launcher的widget，它会发送ACTION\_APPPWIDGET\_ENABLE和ACTION\_APPWIDGET\_UPDATE消息去启动Widget。  
另：如果有Widget设置了updatePeriodMills属性，会每隔一段时间上报ACTION\_APPWIDGET\_UPDATE消息来定时更新此Widget。

简单widget解析：

AppWidget本质就是一个广播接收器。在其中实现了各种方法：

**onUpdate()** : 当AppWidget需要更新时，会触发这个方法，我们需要重写这个方法，在里面实现更新的操作。如果没有定义configuration Activity，那么在添加一个AppWidget

时，也会触发此方法。

**onDelete(Context , int[] )**:当AppWidget从AppWidgetHost中删除时，会触发此方法。

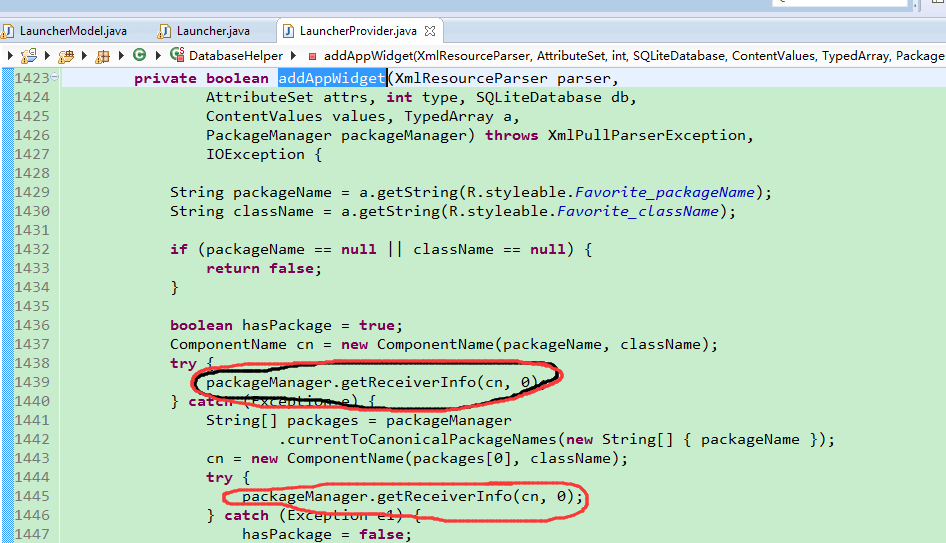
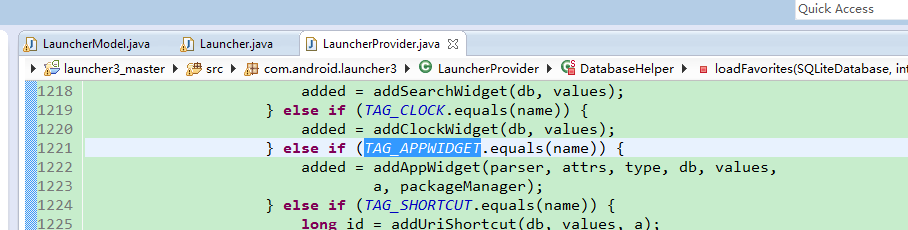
**onEnabled(Context )**:如果为一个应用添加了多个AppWidget，只有在第一个AppWidget被添加时，此方法才会被调用。

**onDisabled(Context )**:当一个应用的最后一个AppWidget从AppWidgetHost中删除时，会触发此方法。

**onReceive(Context , Intent )**:这实际上就是BroadcastReceiver中的方法，当任何一个Broadcast被接收到时，会调用此方法。

### 3 代码分析

Widget加载代码流程：



**private** **void** bindWorkspaceItems(**final** Callbacks oldCallbacks,

**final** ArrayList<ItemInfo> workspaceItems,

**final** ArrayList<LauncherAppWidgetInfo> appWidgets,

**final** HashMap<Long, FolderInfo> folders,

ArrayList<Runnable> deferredBindRunnables) {

**final** **boolean** postOnMainThread = (deferredBindRunnables != **null**);

// Bind the workspace items

**int** N = workspaceItems.size();

**for** (**int** i = 0; i < N; i += *ITEMS\_CHUNK*) {

**final** **int** start = i;

**final** **int** chunkSize = (i + *ITEMS\_CHUNK* <= N) ? *ITEMS\_CHUNK*

: (N - i);

**final** Runnable r = **new** Runnable() {

@Override

**public** **void** run() {

Callbacks callbacks = tryGetCallbacks(oldCallbacks);

**if** (callbacks != **null**) {

callbacks.bindItems(workspaceItems, start, start

+ chunkSize, **false**);

}

}

};

**if** (postOnMainThread) {

deferredBindRunnables.add(r);

} **else** {

runOnMainThread(r, *MAIN\_THREAD\_BINDING\_RUNNABLE*);

}

}

// Bind the folders

**if** (!folders.isEmpty()) {

**final** Runnable r = **new** Runnable() {

**public** **void** run() {

Callbacks callbacks = tryGetCallbacks(oldCallbacks);

**if** (callbacks != **null**) {

callbacks.bindFolders(folders);

}

}

};

**if** (postOnMainThread) {

deferredBindRunnables.add(r);

} **else** {

runOnMainThread(r, *MAIN\_THREAD\_BINDING\_RUNNABLE*);

}

}

// Bind the widgets, one at a time

N = appWidgets.size();

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

**final** LauncherAppWidgetInfo widget = appWidgets.get(i);

**final** Runnable r = **new** Runnable() {

**public** **void** run() {

Callbacks callbacks = tryGetCallbacks(oldCallbacks);

**if** (callbacks != **null**) {

callbacks.bindAppWidget(widget);

}

}

};

**if** (postOnMainThread) {

deferredBindRunnables.add(r);

} **else** {

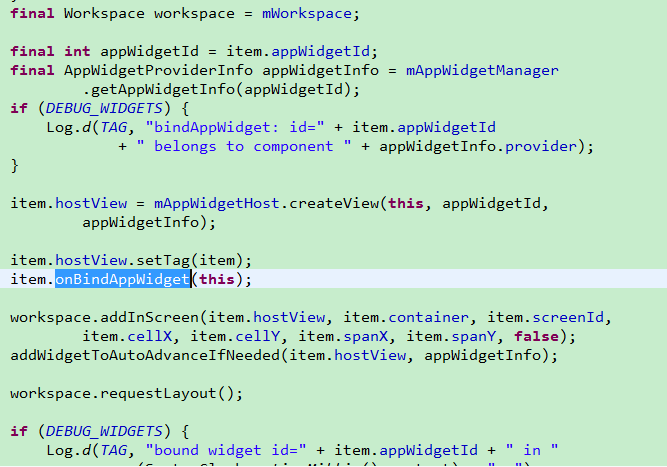
runOnMainThread(r, *MAIN\_THREAD\_BINDING\_RUNNABLE*);

}

}

}

会走到launcher的bindAppWidget方法



**public** **class** MyAppWidgetProvider **extends** AppWidgetProvider {

@Override

**public** **void** onDeleted(Context context, **int**[] appWidgetIds) {

**super**.onDeleted(context, appWidgetIds);

}

@Override

**public** **void** onDisabled(Context context) {

**super**.onDisabled(context);

}

@Override

**public** **void** onEnabled(Context context) {

**super**.onEnabled(context);

}

@Override

**public** **void** onReceive(Context context, Intent intent) {

String action = intent.getAction();

**if** (action.equals("com.xiangying.imageclick")) {

Toast.*makeText*(context, "你点击到我了", Toast.*LENGTH\_SHORT*).show();

}

**super**.onReceive(context, intent);

}

@Override

**public** **void** onUpdate(Context context, AppWidgetManager appWidgetManager,

**int**[] appWidgetIds) {

**super**.onUpdate(context, appWidgetManager, appWidgetIds);

Intent intent = **new** Intent(context, TestAppWidget.**class**);

PendingIntent pendingIntent = PendingIntent.*getActivity*(context, 0,

intent, 0);

RemoteViews remoteViews = **new** RemoteViews(context.getPackageName(),

R.layout.*appwidget*);

remoteViews.setOnClickPendingIntent(R.id.*button\_in\_appwidget*,

pendingIntent);

Intent intents = **new** Intent("com.xiangying.imageclick");

PendingIntent pend = PendingIntent.*getBroadcast*(context, 0, intents, 0);

remoteViews.setOnClickPendingIntent(R.id.*textview\_in\_imageview*, pend);

appWidgetManager.updateAppWidget(appWidgetIds, remoteViews);

}

}

注册：

<receiver android:name=*".MyAppWidgetProvider"*>

<intent-filter>

<action android:name=*"android.appwidget.action.APPWIDGET\_UPDATE"* />

<action android:name=*"com.xiangying.imageclick"*/>

</intent-filter>

<meta-data android:name=*"android.appwidget.provider"*

android:resource=*"@xml/my\_appwidget\_info"* />

</receiver>

**configure** 属性是当用户添加一个App Widget前启动的Activity，这个Activity的作用就是配置App Widget的属性。(Optional)

**updatePeriodMillis** 属性定义了App Widget应该多久向AppWidgetProvider请求更新。实际更新的时间未必能保证及时更新，并且建议尽量不要频繁更新---比如一小时一次去更新电量。也可以容许用户去自定义更新的时间。

重大消息！！！SDK1.5之后此方法**android:updatePeriodMillis就失效了**，要自己创建service更新。

<appwidget-provider xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:minWidth=*"294dp"*

android:minHeight=*"144dp"*

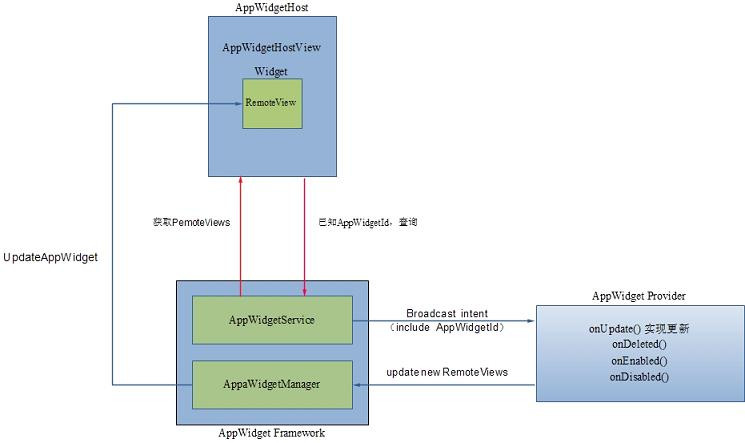
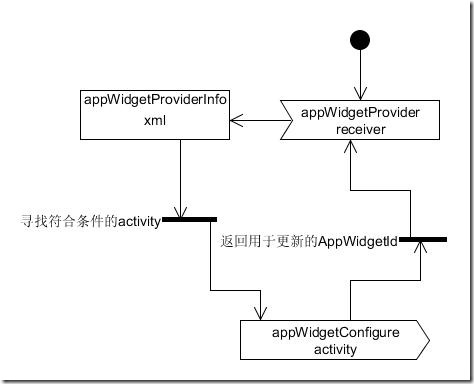
android:updatePeriodMillis=*"86400000"*

android:initialLayout=*"@layout/appwidget"*

android:configure=*"com.example.appwidgettest.MyAppWidgetConfigure"*

>

</appwidget-provider>



**RemoteViews支持的layout和view如下：**

Layout – FrameLayout 、LinearLayout、 RelativeLayou

View -- Analog、Clock、 Button、 Chronometer 、ImageButton、 ImageView、 ProgressBar 、TextView

注意：继承这些类的子类同样不支持。

