About Widget 2

上一篇是对 Dev Guide 中关于 SDK 的翻译，这篇文章的目的就是对 AppWidget 有一个深入的介绍。

一、首先介绍一下在开机过程中系统对 AppWidget 做了什么

当 SystemServer.java 运行到 init2()的时候，通过 SystemManager.addService(Context.AppWidget_Service,appWidget)将 AppWidgetService 服务加到服务队列里面，当所有服务加载完毕后会调用 appWidgetF.systemReady(safeMode)进入到 AppWidgetService.java，在这个方法中做了三件事：

1、遍历所有的安装包，找到符合条件的 ACTION：ACTION_APPWIDGET_UPDATE 和<meta-data android:name="android.appwidget.provider"/>的 receiver，解析相关信息，保存到本地数据成员中。

2、读取本地文件数据：/data/system/appwidgets.xml，所有已安装到桌面的 widget 的信息都保存在这个文件里。读出来，也保存到本地数据成员里。

3、注册了三个消息：ACTION_BOOT_COMPLETED（系统启动到桌面就会发送此消息），ACTION_PACKAGE_ADDED（有新 APK 包安装到系统），ACTION_PACKAGE_REMOVED（有 APK 包被删除）。当系统启动到桌面后，AppWidgetService 接收 到了 ACTION_BOOT_COMPLETED 消息，它会去检查本地数据成员，如果有已经安装到桌面的 widget,它会上发 ACTION_APPWIDGET_ENABLED 和 ACTION_APPWIDGET_UPDATE 消息。如果有 widget 设置了 updatePeriodMillis 的属性，它就会开始计时（这个是通过 AlarmManager 来实现的），到时间时，就会再次发送 ACTION_APPWIDGET_UPDATE 消息。

二、与 AppWidget 相关的类有：

RemoteViews.java

```
 * A class that describes a view hierarchy that can be displayed in
 * another process. The hierarchy is inflated from a layout resource
 * file, and this class provides some basic operations for modifying
 * the content of the inflated hierarchy
```

上面是 Google 给的关于 RemoteViews 的解释，大家不要被它的名字给欺骗了，说白了它就是作为一个描述 view 信息的载体，通过它可以在进程间传递，在另一个进程中由 AppWidgetHostView 去获取 RemoteViews 所承载的信息并且显示出来。

AppWidgetProvider.java

这个类继承 BroadcastReceiver，并且重写了它里面的方法，里面通常使用的是 onUpdate()方法对 AppWidget 更新

AppWidgetManager.java

```
 * Updates AppWidget state; gets information about installed AppWidget
providers and other
```

```
 * AppWidget related state.
```

AppWidgetHost.java

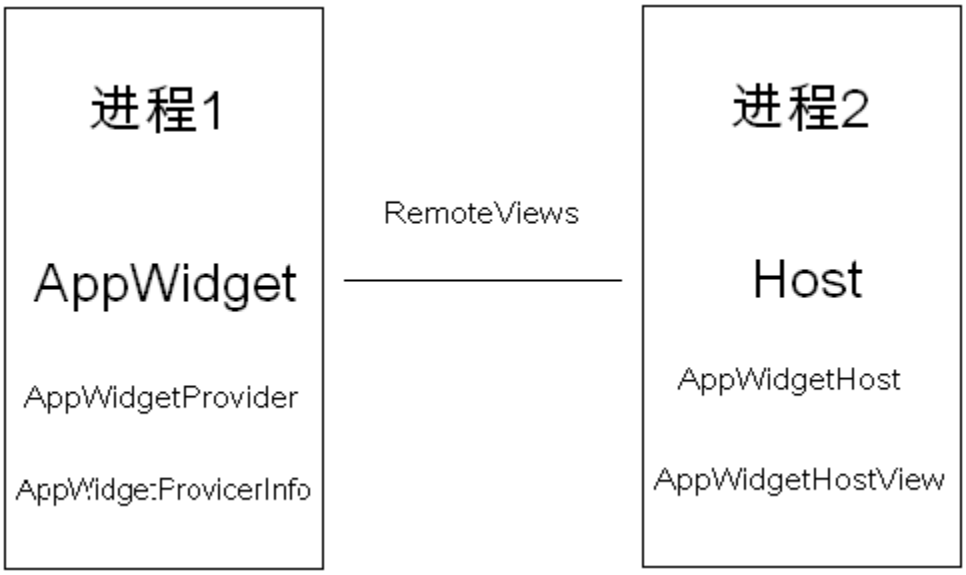
```
* AppWidgetHost provides the interaction with the AppWidget service for
apps,
 * like the home screen, that want to embed AppWidgets in their UI.
```

AppWidgetHostView.java

```
 * Provides the glue to show AppWidget views. This class offers
 automatic animation
 * between updates, and will try recycling old views for each incoming
 * {@link RemoteViews}.
```

AppWidgetService.java

具体实现 AppWidgetHost 和 AppWidgetManager 中的方法。

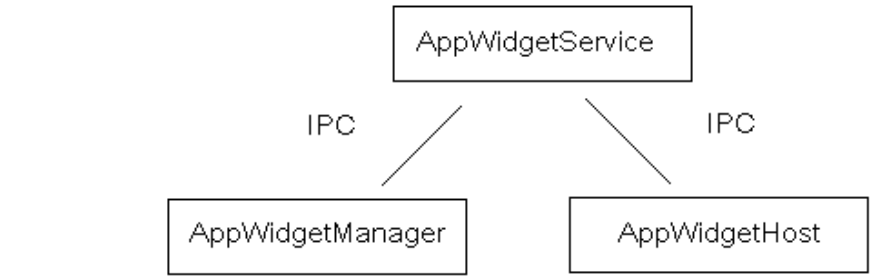


对上图的一个解释：

当我们把一个 AppWidget 放在桌面的时候其实这个 AppWidget 是停靠在 Launcher 的一个 View 上面，被停靠的这个 Activity 我理解为“宿主”，而 AppWidget 是运行在一个独立的进程中，所以 AppWidget 要与这个“宿主”通信的话就需要 IPC。

当 RemoteViews 把 AppWidget 的 View 信息传递“宿主”的时候，通过 AppWidgetHost 获得 AppWidgetHostView 的实例，这样 按照 RemoteViews 中的信息将 AppWidget 的 View 绘制到“宿主”中来。至此，我们的 Widget 就显示在“宿主”上了。

下图是对 AppWidgetManager 和 AppWidgetHost 做的解释，作为管理类，各自完成不同的管理任务。



AppWidgetManager

UpdateAppWidgetState
gets information about installed AppWidget providers and other AppWidget related state

AppWidgetHost

AppWidgetHost provides the interaction with the AppWidget service for apps
like the home screen, that want to embed AppWidgets in their UI

在网上看到这么一段关于 AppWidget 的比喻，贴来大家看看

Android AppWidget 框架妄析： Android 中的借尸还魂

Android, AppWidget, 借尸还魂

由于初识 Android 不久，所以一切分析皆可有误，故而只能为之妄析。 题目起的比较恐怖，然非我本意。 只是实在找不到更加贴切的，可以对 AppWidget 框架一针而见血的比喻了。闲话少说，且看如何个借尸还魂。

首看魂者何来。 大家都知道 Widget 的宗旨，就是要在同一屏幕（界面上）运行多个具有独立功能的小插件，从而丰富功能的同时简化操作。那么，在 Android 的 4 大组件中，何人可以充当该角色，抑或需要再独立设计一个组件？ Activity？ 非也！！ Activity 是 UI 呈现和用户交互的一个组件，具有独特的 Task 管理机制，同一时刻，框架只允许一个 Activity 与用户交互并呈现。 而 Widget 的特点是，多实例的并发交互性。 所以， Activity 不能满足，不能满足同时多个 Widget 的并发交互和呈现。 既然不能前台，那么只能在后台 Running， Service or BroadCastReceiver？ 由于 Widget 需要处理众多的事件交互，所以，BroadCastReceiver 更加合适。 既然找到了合适的，那么也就没有必要再创造新的。 够用

就可以，不是越多越好，这也是软件设计的准则。 OK， AppWidget 的魂已经找到，BroadCastReceive 也， 所以，Android 中的 AppWidget 其本质就是一个 BroadCastReceive 组件。

再看尸者何来。 尸者，阳间之物也。 虽已死（本身无用），却能见光（呈现）也。 任何一个期望在其之上运行 Widget 的前台的应用（Activity），其实就是一个 Widget 宿主。 其本身而言，无任何 Widget 功能，但却可以和用户交互并呈现，从此点而言，可谓尸也。 Android 中的 AppHost 即为尸也。

最后我们看如何还魂。 AppWidget 为魂，功能强大，为所欲为，但却始终位于阴间（后台运行），无法见日，故而众人不可观之。 AppHost 为尸，虽见天日，却已无所可为。 我们何不将此二者互补那？？ 但是，阴阳两隔，必须使用特殊的方式， 此即为还魂术。 通过还魂术，可使得魂寄于尸而见天日。 还魂术就是阴间通往阳间的大道。 Android 中的还魂术即为 RemoteView。 在 Android 中，由于进程边界的存在，使得 AppWidget 与 AppHost 也阴阳两隔，默认是无法直接沟通的。 采用 RemoteView，让 AppWidget 将一切需要呈现的描述构建到 RemoteView 中，AppHost 中再基于该描述，重新创建于属于自己进程中的 View 进而显示。

By Vicki