KONKA 插件开发 V0.4 版说明文档

赵曈

一、简介

1、适用范围

适用于规划中存在许多小功能点，每个小功能点之间相互独立，且后续会不断增加功能点的应用。

宿主与插件的关系为：插件提供自身的视图（ View）以及对各类事件做响应处理，类似于提供了一个完整封装的控件。宿主获取插件提供的 View，在自身页面需要的位置显示。

2、简单原理

a.覆写 Android 原生 DexClassLoader，自定义的 ClassLoader 会先从插件 APK 中初始化，如果没有则从宿主中初始化。这样修改的目的首先是为了解决两个 APK 都引用了同一个库，却因为 ClassLoader 不同导致使用库中同一个类互相赋值时会提示 ClassCastException；其次是因为要实例化插件中的类，必须使用插件的 ClassLoader，否则是找不到相应类的。

b.覆写 Activity/Service 中继承自 Context 的 getAssets,getResources,getTheme 方法，在绘制插件 View 时，使用插件 APK 中获取到的相关对象。这样修改的目的是为了在宿主中能够解析插件中 res 目录下的资源，这样插件开发就可以直接使用 XML 布局。同时此功能也可以实现宿主通过获取插件 APK 中资源来换肤。

c.使用自定义的 ClassLoader（即从插件 APK 中初始化的 ClassLoader），然后就可以通过反射将插件中特定的 class 反射出实例，由于插件和宿主都包含同一类库切插件提供的特定 class 必须实现类库中的特定接口（ IPlugin），因此我们可以将反射出的实例向上转型为 IPlugin，来调用其中的方法，取得插件中提供的视图，供宿主来使用。

3、目前支持功能

支持在 Activity 中添加，通过 Activity/BroadcastReceiver 启动 Service 并新建窗体控件（如 Dialog）在其中添加。

可以同一宿主页面添加多个不同插件提供的 View。

插件可以使用 XML 布局，可以使用自定义控件。但事件处理必须自行封装。

二、使用说明

1、宿主端

前提：将 DynamicPlugin.jar 加入 libs 目录。

具体使用参照 Host 工程。

第一步：新建一个继承 HostApplication 的类，将其作为工程的自定义 Application 类。如下图

```
public class TestApplication extends HostApplication {

    @Override
    public void onAppCreate() {
        // TODO Auto-generated method stub

    }
```

```
<application
    android:name="com.zt.host.TestApplication"
    android:allowBackup="true"
```

第二步：根据实际，根据需要选择继承 HostActivity/HostService。也可以通过 Activity/Service 启动弹窗（如 Dialog,PopupWindow）一类控件。

第三步：对本地插件进行初始化，可以调用以下两个接口之一：

```
void initPlugins(Context context) throws FileNotFoundException;

void asyncInitPlugins(Context context, IAsyncListener listener);
```

此接口目的在于对本地插件进行初始化记录。如果本地目录没有插件，则将宿主应用中 assets/plugins/目录下的插件 apk 释放至本地存储目录并初始化。

第四步：获取不同状态的插件。用户可以调用以下接口获取三种状态的插件

1. 获取所有本地插件

```
List<PluginInfo> getAllRecordedPlugins();
```

2. 获取已安装插件

```
List<PluginInfo> getInstalledPlugins();
```

3. 获取已启用插件

```
List<PluginInfo> getEnablePlugins();
```

第五步：对插件进行管理。用户可以调用 Install/Uninstall/Enable/Disable 四类方法对插件的生命周期进行管理。我们规定一个插件如果需要显示在宿主上，则必须 Enable，如果需要 Enable/Disable，则必须 Install。用户可在满足此生命周期规定的条件下自行组织交互。Host 例子提供了一种简单的标准交互方式。

第六步：显示插件。用户启用相关插件后，通过第四步中介绍的接口获取到已启用的所有插件，即可调用以下接口拿到插件提供的 View 视图，之后即可随意添加于宿主中。

```
View getPluginView(Context context, PluginInfo pluginInfo);
```

2、插件端

前提：将 DynamicPlugin.jar 加入构建路径，但不要拷贝至 libs 文件夹，因为此 jar 只在编译时使用，不可以导出，导出的话会与宿主中包含的相同 jar 包产生类冲突。

插件端也是普通的 Android APP。此 APP 可以包含 Activity 用于自己运行调试，也可以不包含。具体导入和使用可参照 PluginBrightness 工程。

第一步：插件端要有一个实现类继承 BasePlugin 虚类，作为与宿主交换数据，提供 View 的实现类。

第二步：BasePlugin 提供三个需要实现的虚函数，用于插件视图生命周期管理。如下：

```
/**
 * 插件被宿主创建，请在此进行初始化的操作。
 */
public abstract void onCreate();

/**
 * 插件显示在宿主时回调，可以在此进行{@code Service},{@code BroadcastReceiver}等的绑定
 */
public abstract void onAttach();

/**
 * 插件从宿主中消失时回调，可以在此进行{@code Service},{@code BroadcastReceiver}等的解绑
 */
public abstract void onDetach();
```

第三步：用户通过以下方法解析 XML：
解析 rootView：

```
final LinearLayout layout = (LinearLayout) inflateRootView(R.layout.main);
```

解析子控件：

```
mBrightnessLayout = (LinearLayout) findViewById(layout, R.id.brightness_layout);
```

请务必使用此处提供的方法，否则会导致视图回调和自定义插件无法正常使用。

第四步：设置以下函数的返回值，将插件视图提供给宿主。

```
View getPluginView();
```

第五步：修改插件的 AndroidManifest.xml 中 android:versionCode 字段，用于标识插件版本供升级检测；在 AndroidManifest.xml 中 application 标签下增加 android:description 字段，用于指定插件实现了 BasePlugin 的实现类的完整 className，供宿主解析，如下，如果此字

```
android:versionCode="1"
android:versionName="1.0" >

<uses-sdk
    android:minSdkVersion="17"
    android:targetSdkVersion="19" />

<application
    android:allowBackup="true"
    android:description="@string/plugin_class_name"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
</application>
```

段指定有误，则宿主无法正确获取插件视图。

3、调试

第一次运行宿主 APK，将默认带有的插件放入宿主 assets/plugins 目录下编译宿主 APK，如果初始化正确，启动后会在以下目录看到默认打包携带的插件 APK:

/data/misc/konka/plugins/plugin 如果无此目录，则在
/data/data/hostPackageName/app_plugins 目录下。

后续修改插件后想更新，则手动将其拷贝到上述对应目录并改权限使其可读写，即可生效。

三、库工程后续计划

1、插件在线升级系统。