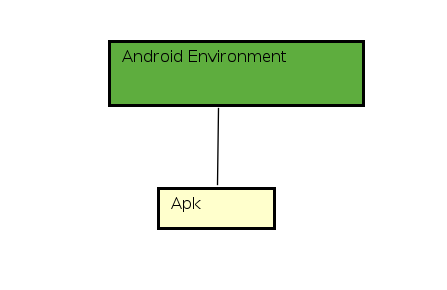
插件说明文档

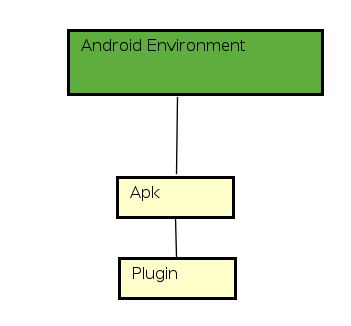
--by lqp

1. 插件和普通apk的不同点

正常情况下一个可以安装运行的apk主要包括代码（classes.dex）资源(res), 资源在代码中是以id的形式进行访问的, 在生成apk文件时打包工具appt会建立各个资源和id之间的对应关系(对应apk中的resources.arsc包含了这种对应关系). 一个独立运行的apk在运行时除了访问自身代码和资源外,还需要访问android系统的代码和资源, 其中系统代码就是我们调用的sdk api对应的class代码, 在打包成的apk中并不包含这部分代码, 在运行apk的过程中, 虚拟机在解析到相应的调用时, 比如Log.d(…), 会从虚拟机环境中找到该类, 然后进行调用; 同样, 我们还可以通过android.R的形式引用系统的资源, 这部分资源也不打包到apk中. 这样做的好处主要有2个, 一是减少apk包的大小, 这是显而易见的, 二是提供灵活性, 每个系统都可以对api接口的行为进行定制, 只要保证函数签名一致即可, 而对于资源, 形如android.R.drawble.back, 在运行时会对应到rom里的资源, 从而展现出不同的back图标.



上面的这种方式可以看出, apk的运行本身就有点类似于插件的, apk的行为并不完全由自身决定, 其中有一部分是来自于运行环境(适配问题的根源). 由此引出我们的插件方案.



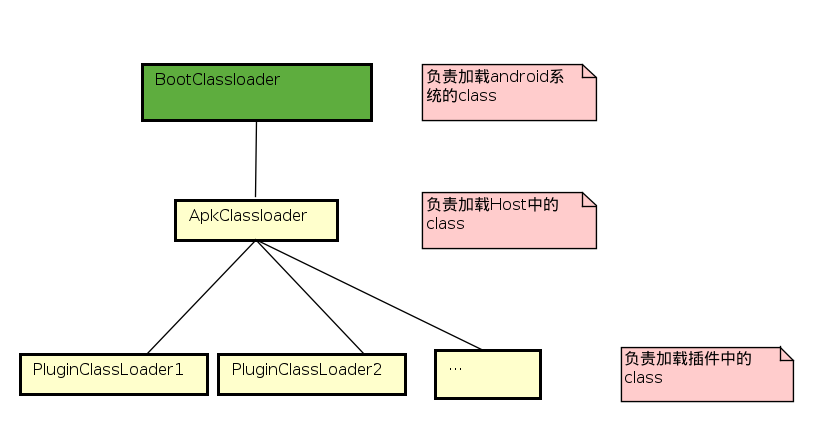
从上图来看, 插件的运行依赖于宿主(Host), 宿主成为了一个运行环境, 这样, 插件访问宿主中的资源(包括图片,字符串等, 代码也可以理解为资源); 插件不能独立运行, 因为部分代码位于host中.

1. 实现细节

下面的过程中主要围绕代码和资源来展开

A. 代码的拆分

基本思路就是将公共的代码放到common库, 它被插件和host引用, 但只打包进host中, 而属于插件逻辑的代码放到plugin中, plugin中可以调用common库的代码 . 类比于apk中调用android sdk 的接口.

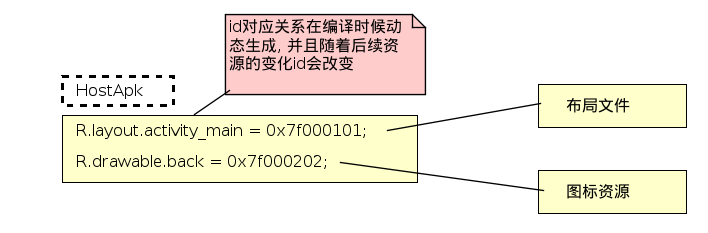


如上图中的层级关系所示, PluginClassLoader只负责加载plugin中的class, 对于不在plugin中的class, 则交给它的父类ApkClassloader, ApkClassloader只知道host中定义类(其中包括common代码的实现)如何加载, 而对于android系统中的类, 则交给bootclassloader去完成. 因此代码意义层面上的拆分比价简单, 需要在打包apk时, 只打包插件自己的代码即可. 当然, google并没有提供这样现成的打包方式, 所以这需要我们自己来实现.

B. 资源的拆分

资源的拆分比代码要复杂得多, 并且由于资源拆分带来的问题会影响到代码的拆分和编码.

先看一下id和资源的关系:



上图是id和资源的对应关系, 这个对应关系在打包资源的时候确定并和最终apk打包在一起, 并且, 代码中的id引用都会被替换为具体的整数值, 也就是写死在编译好的代码里面,同样, xml布局文件中

android:background="@color/view\_bg\_1"

在编译后也变成了形如: android:background=@23242414143 的形式.

从代码访问上来说, 访问资源都是通过Context提供的接口来访问的, 而Context又是通过Resources这个类来访问, 这个类会解析apk中id和资源的对应关系. 从而, id和资源的对应关系是跟着apk走的. Hostapk中的id只能用来访问host中的资源(通过Host的Context), plugin中的id只能用来访问plugin中的资源(通过Plugin的Context). 一个Context只能关联到一个apk. 在生成资源的过程中, 必须保证所有的引用都可以被找到, 因此, 插件xml中的资源引用不能引用host的资源(common资源也在这里), 因为打包插件过程中, common的资源并不参与打包. 而对于代码, 我们只需要把插件中编译好的class文件提取出来单独打包进apk即可.

公共控件中对资源的访问(common代码)

公共控件是放在common代码中的, 上面提到, common代码只打包进host. 一般来说控件的构造方法中会和一个Context产生关系, 从而从Context中去访问需要的布局, 字符串, 图片等. 用的是谁的Context, 则在对应的apk下面寻找id对应的资源. 由于common代码打包进host中,它可以自由的访问host中所有的资源. 但是存在一种情况, 在插件中调用inflater从插件布局中初始化一个view时, 使用的是插件的Context, 这个Context会传入到公共控件的构造方法中, 如果通过这个Context去访问控件库里定义的资源, 是访问不到的, 这个时候, 需要通过插件的Context获取Application的Context来访问.

3. 插件的开发模式

在代码和资源满足插件拆分约束的前提下, 插件作为一个app Module存在, 可以单独运行这个app进行测试. 其次, 由于activity使用需要注册, 而插件是动态由host加载的, 因此插件的界面逻辑不能使用activity, 而选择基于Fragment进行开发. 插件单独运行测试通过后, 由插件打包工具打包成必须通过host加载才能运行的插件. 关于插件开发中更多的规则和限制, 请参考 “插件编码规范和限制.docx”