Android动态加载Activity原理

Android动态加载Activity原理

标签： 动态加载activity免安装插件activity
2016-04-07 18:18 5744人阅读 评论(1) 收藏 举报

分类：
Android（26）

目录(?)[+]

activity的启动流程

加载一个Activity肯定不会像加载一般的类那样，因为activity作为系统的组件有自己的生命周期，有系统的很多回调控制，所以自定义一个DexClassLoader类加载器来加载插件中的Activity肯定是不可以的。

首先不得不了解一下activity的启动流程，当然只是简单的看一下，太详细的话很难研究清楚。

通过startActivity启动后，最终通过AMS进行跨进程回调到ApplicationThread的scheduleLaunchActivity，这时会创建一个ActivityClientRecord对象，这个对象表示一个Acticity以及他的相关信息，比如activityInfo字段包括了启动模式等，还有loadedApk，顾名思义指的是加载过了的APK，他会被放在一个Map中，应用包名到LoadedApk的键值对，包含了一个应用的相关信息。然后通过Handler切换到主线程执performLaunchActivity

[java] view plain copy

```java
private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {
    ActivityInfo aInfo = r.activityInfo;
    // 1.创建ActivityClientRecord对象时没有对他的packageInfo赋值，所以它是null
    if (r.packageInfo == null) {
        r.packageInfo = getPackageInfo(aInfo.applicationInfo, r.compatInfo, Context.CONTEXT_INCLUDE_CODE);
    }
    // ...
    Activity activity = null;
    try {
        // 2.非常重要！！这个ClassLoader保存于LoadedApk对象中，它是用来加载我们写的activity的加载器
        java.lang.ClassLoader cl = r.packageInfo.getClassLoader();
        // 3.用加载器来加载activity类，这个会根据不同的intent加载匹配的activity
        activity = mInstrumentation.newActivity(cl, component.getClassName(), r.intent);
        StrictMode.incrementExpectedActivityCount(activity.getClass());
        r.intent.setExtrasClassLoader(cl);
        if (r.state != null) {
            r.state.setClassLoader(cl);
        }
    } catch (Exception e) {
        // 4.这里的异常也是非常非常重要的！！！后面就根据这个提示找到突破口。。。
        if (!mInstrumentation.onException(activity, e)) {
                throw new RuntimeException(
                    "Unable to instantiate activity " + component
                    + ": " + e.toString(), e);
            }
    }
        if (activity != null) {
            Context appContext = createBaseContextForActivity(r, activity);
            CharSequence title = r.activityInfo.loadLabel(appContext.getPackageManager());
            Configuration config = new Configuration(mCompatConfiguration);
            // 从这里就会执行到我们通常看到的activity的生命周期的onCreate里面
            mInstrumentation.callActivityOnCreate(activity, r.state);
            // 省略的是根据不同的状态执行生命周期
        }
        r.paused = true;
        mActivities.put(r.token, r);
    } catch (SuperNotCalledException e) {
        throw e;
    } catch (Exception e) {
        // ...
    }
    return activity;
}
```

1.getPackageInfo方法最终返回一个LoadedApk对象，它会从一个HashMap的数据结构中取，mPackages维护了包名和LoadedApk的对应关系，即每一个应用有一个键值对对应。如果为null，就新创建一个LoadedApk对象，并将其添加到Map中，重点是这个对象的ClassLoader字段为null！

[java] view plain copy

```java
  public final LoadedApk getPackageInfo(ApplicationInfo ai, CompatibilityInfo compatInfo,
          int flags) {
    // 为true
      boolean includeCode = (flags&Context.CONTEXT_INCLUDE_CODE) != 0;
      boolean securityViolation = includeCode && ai.uid != 0
              && ai.uid != Process.SYSTEM_UID && (mBoundApplication != null
                      ? !UserHandle.isSameApp(ai.uid, mBoundApplication.appInfo.uid)
                      : true);
// ...
      // includeCode为true
      // classloader为null！！！
```

```
        return getPackageInfo(ai, compatInfo, null, securityViolation, includeCode);
    }

    private LoadedApk getPackageInfo(ApplicationInfo aInfo, CompatibilityInfo compatInfo,
            ClassLoader baseLoader, boolean securityViolation, boolean includeCode) {
        synchronized (mPackages) {
            WeakReference<LoadedApk> ref;
            if (includeCode) {
              // includeCode为true
                ref = mPackages.get(aInfo.packageName);
            } else {
                ref = mResourcePackages.get(aInfo.packageName);
            }
            LoadedApk packageInfo = ref != null ? ref.get() : null;
            if (packageInfo == null || (packageInfo.mResources != null &&
!packageInfo.mResources.getAssets().isUpToDate())) {
                if (localLOGV) // ...
                // packageInfo为null，创建一个LoadedApk，并且添加到mPackages里面
                packageInfo = new LoadedApk(this, aInfo, compatInfo, this, baseLoader, securityViolation, includeCode
&&
                            (aInfo.flags&ApplicationInfo. ) != 0);
                if (includeCode) {
                    mPackages.put(aInfo.packageName, new WeakReference<LoadedApk>(packageInfo));
                } else {
                    mResourcePackages.put(aInfo.packageName, new WeakReference<LoadedApk>(packageInfo));
                }
            }
            return packageInfo;
        }
    }
```

2.获取这个activity对应的类加载器，由于上面说过，mClassLoader为null，那么就会执行到ApplicationLoaders#getClassLoader(zip, libraryPath, mBaseClassLoader)方法。

[java] view plain copy

```
public ClassLoader getClassLoader() {
    synchronized (this) {
        if (mClassLoader != null) {
            return mClassLoader;
        }
        // ...
        // 创建加载器，创建默认的加载器
        // zip为Apk的路径，libraryPath也就是JNI的路径
        mClassLoader = ApplicationLoaders.getDefault().getClassLoader(zip, libraryPath, mBaseClassLoader);
        initializeJavaContextClassLoader();
        StrictMode.setThreadPolicy(oldPolicy);
        } else {
            if (mBaseClassLoader == null) {
                mClassLoader = ClassLoader.getSystemClassLoader();
            } else {
                mClassLoader = mBaseClassLoader;
            }
        }
        return mClassLoader;
    }
}
```

ApplicationLoaders使用单例它的getClassLoader方法根据传入的zip路径事实上也就是Apk的路径来创建加载器，返回的是一个PathClassLoader。并且PathClassLoader只能加载安装过的APK。这个加载器创建的时候传入的是当前应用APK的路径，理所应当的，想加载其他的APK就构造一个传递其他APK的类加载器。

3.用该类加载器加载我们要启动的activity，并反射创建一个activity实例

[java] view plain copy

```
public Activity newActivity(ClassLoader cl, String className,Intent intent) throws InstantiationException,
IllegalAccessException, ClassNotFoundException {
    return (Activity)cl.loadClass(className).newInstance();
}
```

总结一下上面的思路就是,当我们启动一个activity时，通过系统默认的PathClassLoader来加载这个activity，当然默认情况下只能加载本应用里面的activity，然后就由系统调用到这个activity的生命周期中。

4.这个地方的异常在后面的示例中会出现，到时候分析到原因后就可以找出我们动态加载Activity的思路了。

动态加载**Activity**：修改系统类加载器

按照这个思路，做这样的一个示例，按下按钮，打开插件中的Activity。

插件项目

plugin.dl.pluginactivity

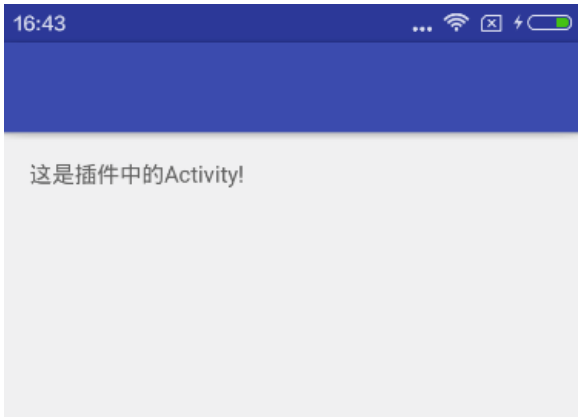
|--MainActivity.java

内容很简单，就是一个布局上面写了这是插件中的Activity!并重写了他的onStart和onDestroy方法。

[java] view plain copy

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        // 加载到宿主程序中之后，这个R.layout.activity_main就是宿主程序中的R.layout.activity_main了
        setContentView(R.layout.activity_main);
    }
    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this,"onStart", 0).show();
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this,"onDestroy", 0).show();
    }
}
```



宿主项目

host.dl.hostactivity

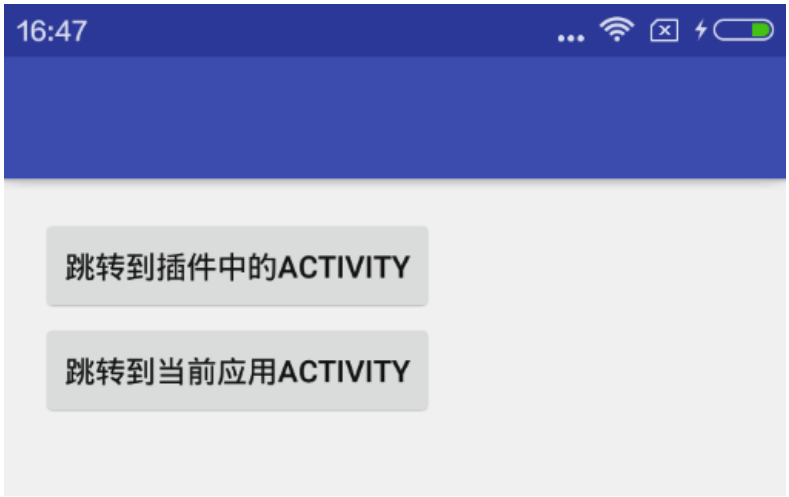
|--MainActivity.java

包括两个按钮，第一个按钮跳转到插件中的MainActivity.java，第二个按钮调转到本应用中的MainActivity.java

[java] view plain copy

```
private Button btn;
    private Button btn1;
    DexClassLoader loader;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button) findViewById(R.id.btn);
        btn1 = (Button) findViewById(R.id.btn1);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Class activity = null;
                String dexPath = "/PluginActivity.apk";
                loader = new DexClassLoader(dexPath, MainActivity.this.getApplicationInfo().dataDir, null,
getClass().getClassLoader());
                try {
                    activity = loader.loadClass("plugin.dl.pluginactivity.MainActivity");
                }catch (ClassNotFoundException e) {
                    Log.i("MainActivity", "ClassNotFoundException");
                }
                Intent intent = new Intent(MainActivity.this, activity);
                MainActivity.this.startActivity(intent);
            }
        });

        btn1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, MainActivity2.class);
                MainActivity.this.startActivity(intent);
            }
        });
```



首先我们要将该activity在宿主工程的额AndroidManifest里面注册。点击按钮打开插件中的activity，发现报错

[html] view plain copy

```
1. java.lang.RuntimeException: Unable to instantiate activity
   ComponentInfo{host.dl.hostactivity/plugin.dl.pluginactivity.MainActivity}: java.lang.ClassNotFoundException:
   plugin.dl.pluginactivity.MainActivity
```

#已经使用自定义的加载器，当startActivity时为什么提示找不到插件中的activity？

前面第四点说过这个异常。其实这个异常就是在performLaunchActivity中抛出的，仔细看这个异常打印信息，发现它说plugin.dl.pluginactivity.MainActivity类找不到，可是我们不是刚刚定义了一个DexClassLoader，成功加载了这个类的吗？？怎么这里又提示这个类找不到？

实际上，确实是这样的，还记得前面说过，系统默认的类加载器PathClassLoader吗？（因为LoadedApk对象的mClassLoader变量为null，就调用到ApplicationLoaders#getClassLoader方法，即根据当前应用的路径返回一个默认的PathClassLoader），当执行到mPackages.get(aInfo.packageName);时从Map获取的LoadedApk中未指定mClassLoader，因此会使用系统默认的类加载器。于是当执行这一句 mInstrumentation.newActivity(cl, component.getClassName(), r.intent);时，由于**这个类加载器找不到我们插件工程中的类**，因此报错了。

现在很清楚了，原因就是使用**系统默认的这个类加载器不包含插件工程路径，无法正确加载我们想要的activity**造成的。

于是考虑替换系统的类加载器。

[java] view plain copy

```
1.  private void replaceClassLoader(DexClassLoader loader) {
2.      try {
3.          Class clazz_Ath = Class.forName("android.app.ActivityThread");
4.          Class clazz_LApk = Class.forName("android.app.LoadedApk");
5.          Object currentActivityThread = clazz_Ath.getMethod("currentActivityThread").invoke(null);
6.          Field field1 = clazz_Ath.getDeclaredField("mPackages");
7.          field1.setAccessible(true);
8.          Map mPackages = (Map) field1.get(currentActivitead);
9.          String packageName = MainActivity.this.getPackageName();
10.         WeakReference ref = (WeakReference) mPackages.get(packageName);
11.         Field field2 = clazz_LApk.getDeclaredField("mClassLoader");
12.         field2.setAccessible(true);
13.         field2.set(ref.get(), loader);
14.     } catch (Exception e) {
15.         e.printStackTrace();
16.     }
17. }
```

这段代码的思路是将ActivityThread类中的mPackages变量中保存的以当前包名为键的LoadedApk值的mClassLoader替换成我们自定义的类加载器。当下一次要加载存放在别的地方的插件中的某个Activity时，直接在mPackages变量中能取到，因此用的就是我们修改了的类加载器了。

因此，在打开插件中的activity之前调用replaceClassLoader(loader);方法替换系统的类加载器，就可以了。

效果如下

此时发现可以启动插件中的activity，因为执行到了他的onStart方法，并且关闭的时候执行了onDestroy方法，但是奇怪的是界面上的控件貌似没有变化？和启动他的界面一模一样，还不能点击。这是什么原因呢？

显然，我们只是把插件中的MainActivity类加载过来了，当执行到他的onCreate方法时，在里面调用setContentView使用的布局参数是R.layout.activity_main，因为文件名是一样的，他们的id也是一样的，当然使用的就是当前应用的资源了。

##已经替换了系统的类加载器为什么加载本应用的activity却能正常运行？

不过在修正这个问题之前，有没有发现一个很奇怪的现象，当加载过插件中的activity后，再次启动本地的activity也是能正常启动的？这是为什么呢？前面已经替换了默认的类加载器了，并且可以在打开插件中的activity后再点击第二个按钮打开本应用的activity之前查看使用的activity，确实是我们已经替换了的类加载器。那这里为什么还能正常启动本应用的activity呢？玄机就在我们创建DexClassLoader时的第四个参数，父加载器！设置父加载器为当前类的加载器，就能**保证类的双亲委派模型不被破坏**，在加载类时都是先由父加载器来加载，加载不成功时在由自己加载。不信可以在new这个加载器的时候父加载器的参数设置成其他值，比如系统类加载器，那么当运行activity时肯定会报错。

接下来解决前面出现的，跳转到插件activity中界面显示不对的问题。这个现象出现的原因已经解释过了，就是因为使用了本地的资源所导致的，因此需要在setContentView时，使用插件中的资源布局。因此在插件Activity中作如下修改。也可以传递一个宿主Activity的引用作为Context，调用它的setContentView方法，这样在找ID的时候就会找插件中的资源，前提是在宿主中加载过插件资源并且重写getResources方法。

[java] view plain copy

```java
public class MainActivity2 extends Activity {
    private static View view;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
// 加载到宿主程序中之后，这个R.layout.activity_main就是宿主程序中的R.layout.activity_main了
//        setContentView(R.layout.activity_main);
        if (view != null)
        setContentView(view);
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this,"onStart", 0).show();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Toast.makeText(this,"onDestroy", 0).show();
    }

    private static void setLayout(View v){
        view = v;
    }
}
```

然后在宿主Activity中获取插件资源并将布局填充成View，然后设置给插件中的activity，作为它的ContentView的内容。

[java] view plain copy

```java
Class<?> layout = loader.loadClass("plugin.dl.pluginactivity.R$layout");
Field field = layout.getField("activity_main");
Integer obj = (Integer) field.get(null);
// 使用包含插件APK的Resources对象来获取这个布局才能正确获取插件中定义的界面效果
//View view = LayoutInflater.from(MainActivity.this).inflate(resources.getLayout(obj),null);
// 或者这样，但一定要重写getResources方法，才能这样写
```

```
View view = LayoutInflater.from(MainActivity.this).inflate(obj, null);
Method method = activity.getDeclaredMethod("setLayout", View.class);
method.setAccessible(true);
method.invoke(activity, view);
```

完整的代码

[java] view plain copy

```
public class MainActivity extends Activity {
    private Resources resources;
    protected AssetManager assetManager;
    private Button btn;
    private Button btn1;
    DexClassLoader loader;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button) findViewById(R.id.btn);
        btn1 = (Button) findViewById(R.id.btn1);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String dexPath = "/PluginActivity.apk";
                loader = new DexClassLoader(dexPath, MainActivity.this.getApplicationInfo().dataDir, null, getClass().getClassLoader());
                Class<?> activity = null;
                Class<?> layout = null;
                try {
                    activity = loader.loadClass("plugin.dl.pluginactivity.MainActivity");
                    layout = loader.loadClass("plugin.dl.pluginactivity.R$layout");
                }catch (ClassNotFoundException e) {
                    Log.i("MainActivity", "ClassNotFoundException");
                }
                replaceClassLoader(loader);
                loadRes(dexPath);
                try {
                    Field field = layout.getField("activity_main");
                    Integer obj = (Integer) field.get(null);
                    // 使用包含插件APK的Resources对象来获取这个布局才能正确获取插件中定义的界面效果
                  View view = LayoutInflater.from(MainActivity.this).inflate(resources.getLayout(obj),null);
                    // 或者这样，但一定要重写getResources方法，才能这样写
//                    View view = LayoutInflater.from(MainActivity.this).inflate(obj, null);
                    Method method = activity.getDeclaredMethod("setLayout", View.class);
                    method.setAccessible(true);
                    method.invoke(activity, view);

                } catch (Exception e) {
                    e.printStackTrace();
                }
                Intent intent = new Intent(MainActivity.this, activity);
                MainActivity.this.startActivity(intent);
            }
        });

        btn1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, MainActivity2.class);
                MainActivity.this.startActivity(intent);
            }
        });
    }

    public void loadRes(String path){
        try {
            assetManager = AssetManager.class.newInstance();
            Method addAssetPath = AssetManager.class.getMethod("addAssetPath", String.class);
            addAssetPath.invoke(assetManager, path);
        } catch (Exception e) {
        }
        resources = new Resources(assetManager, super.getResources().getDisplayMetrics(), super.getResources().getConfiguration());
        // 也可以根据资源获取主题
    }

    private void replaceClassLoader(DexClassLoader loader){
        try {
            Class clazz_Ath = Class.forName("android.app.ActivityThread");
            Class clazz_LApk = Class.forName("android.app.LoadedApk");

            Object currentActivityThread = clazz_Ath.getMethod("currentActivityThread").invoke(null);
            Field field1 = clazz_Ath.getDeclaredField("mPackages");
            field1.setAccessible(true);
            Map mPackages = (Map)field1.get(currentActivityThread);

            String packageName = MainActivity.this.getPackageName();
            WeakReference ref = (WeakReference) mPackages.get(packageName);
            Field field2 = clazz_LApk.getDeclaredField("mClassLoader");
            field2.setAccessible(true);
            field2.set(ref.get(), loader);
        } catch (Exception e){
            System.out.println("-----------------------------------" + "click");
            e.printStackTrace();
        }
    }

```

```
88.     @Override
89.     public Resources getResources() {
90.         return resources == null ? super.getResources() : resources;
91.     }
92.
93.     @Override
94.     public AssetManager getAssets() {
95.         return assetManager == null ? super.getAssets() : assetManager;
96.     }
97. }
```

效果

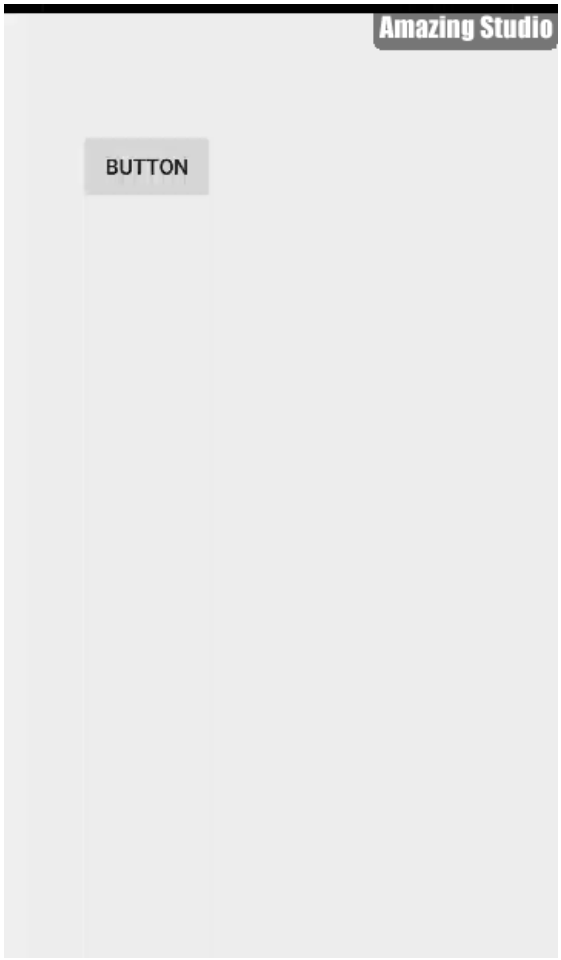
代码点此下载

动态加载**Activity**：使用代理

还有一种方式启动插件中的activity的方式就是将插件中的activity当做一个一般的类，不把它当成组件activity，于是在启动的时候启动一个代理ProxyActivity，它才是真正的Activity，他的生命周期由系统管理，我们在它里面调用插件Activity里的函数即可。同时，在插件Activity里面保存一个代理Activity的引用，把这个引用当做上下文环境Context理解。

这里插件Activity的生命周期函数均由代理Activity调起，**ProxyActivity其实就是一个真正的我们启动的Activity，而不是启动插件中的Activity，插件中的“要启动”的Activity就当做一个很普通的类看待，当成一个包含了一些函数的普通类来理解，只是这个类里面的函数名字起的有些“奇怪”罢了。**涉及到访问资源和更新UI相关的时候通过当前上下文环境，即保存的proxyActivity引用来获取。

以下面这个Demo为例



宿主项目

com.dl.host

|--MainActivity.java

|--ProxyActivity.java

MainActivity包括一个按钮，按下按钮跳转到插件Activity

[java] view plain copy

```java
public class MainActivity extends Activity{

    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = (Button)findViewById(R.id.btn);
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                MainActivity.this.startActivity(new Intent(MainActivity.this, ProxyActivity.class));
            }
        });
    }
}
```

ProxyActivity就是我们要启动的插件Activity的一个傀儡，代理。是系统维护的Activity。

[java] view plain copy

```java
public class ProxyActivity extends Activity{

    private DexClassLoader loader;
    private Activity activity;
    private Class<?> clazz = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        loader = new DexClassLoader("/Plugin.apk", getApplicationInfo().dataDir, null,
getClass().getClassLoader());
        try {
            clazz = loader.loadClass("com.dl.plugin.MainActivity");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        // 设置插件activity的代理
        try {
            Method setProxy = clazz.getDeclaredMethod("setProxy", Activity.class);
            setProxy.setAccessible(true);
```

```
20.
21.             activity = (Activity)clazz.newInstance();
22.             setProxy.invoke(activity, this);
23.
24.             Method onCreate = clazz.getDeclaredMethod("onCreate", Bundle.class);
25.             onCreate.setAccessible(true);
26.             onCreate.invoke(activity, savedInstanceState);
27.         } catch (Exception e) {
28.             e.printStackTrace();
29.         }
30.     }
31.     @Override
32.     protected void onStart() {
33.         super.onStart();
34.         // 调用插件activity的onStart方法
35.         Method onStart = null;
36.         try {
37.             onStart = clazz.getDeclaredMethod("onStart");
38.             onStart.setAccessible(true);
39.             onStart.invoke(activity);
40.         } catch (Exception e) {
41.             e.printStackTrace();
42.         }
43.     }
44.     @Override
45.     protected void onDestroy() {
46.         super.onStart();
47.         // 调用插件activity的onDestroy方法
48.         Method onDestroy = null;
49.         try {
50.             onDestroy = clazz.getDeclaredMethod("onDestroy");
51.             onDestroy.setAccessible(true);
52.             onDestroy.invoke(activity);
53.         } catch (Exception e) {
54.             e.printStackTrace();
55.         }
56.     }
57. }
```

可以看到，ProxyActivity其实就是一个真正的Activity，我们启动的就是这个Activity，而不是插件中的Activity。

插件项目

com.dl.plugin

|--MainActivity.java

保存了一个代理Activity的引用，值得注意的是，由于访问插件中的资源需要额外的操作，要加载资源，因此这里未使用插件项目里面的资源，所以我使用代码添加的TextView，但原理和前面讲的内容是一样的。

[java] view plain copy

```
1.  public class MainActivity extends Activity {
2.      private Activity proxyActivity;
3.      public void setProxy(Activity proxyActivity) {
4.          this.proxyActivity = proxyActivity;
5.      }
6.
7.      // 里面的所有操作都由代理activity来操作
8.      @Override
9.      protected void onCreate(Bundle savedInstanceState) {
10.         TextView tv = new TextView(proxyActivity);
11.         tv.setText("插件Activity");
12.         proxyActivity.setContentView(tv,new FrameLayout.LayoutParams(LayoutParams.WRAP_CONTENT,
    LayoutParams.WRAP_CONTENT));
13.     }
14.
15.     @Override
16.     protected void onStart() {
17.         Toast.makeText(proxyActivity, "插件onStart", 0).show();
18.     }
19.     @Override
20.     protected void onDestroy() {
21.         Toast.makeText(proxyActivity, "插件onDestroy", 0).show();
22.     }
23. }
```

这种方法相比较前面修改系统加载器的方法需要自己维护生命周期，比较麻烦，前一种方式由系统自己维护，并且启动的就是插件中实实在在的Activity。

前一种方式要在宿主的AndroidManifest里面声明插件Activity，这样当activity太多时就要声明很多，比较繁琐，不过也可以不声明逃过系统检查。后面这种方式就只需要一个代理ProxyActivity类即可。在他的onCreate里面根据传递的值选择加载插件中的哪个Activity即可。