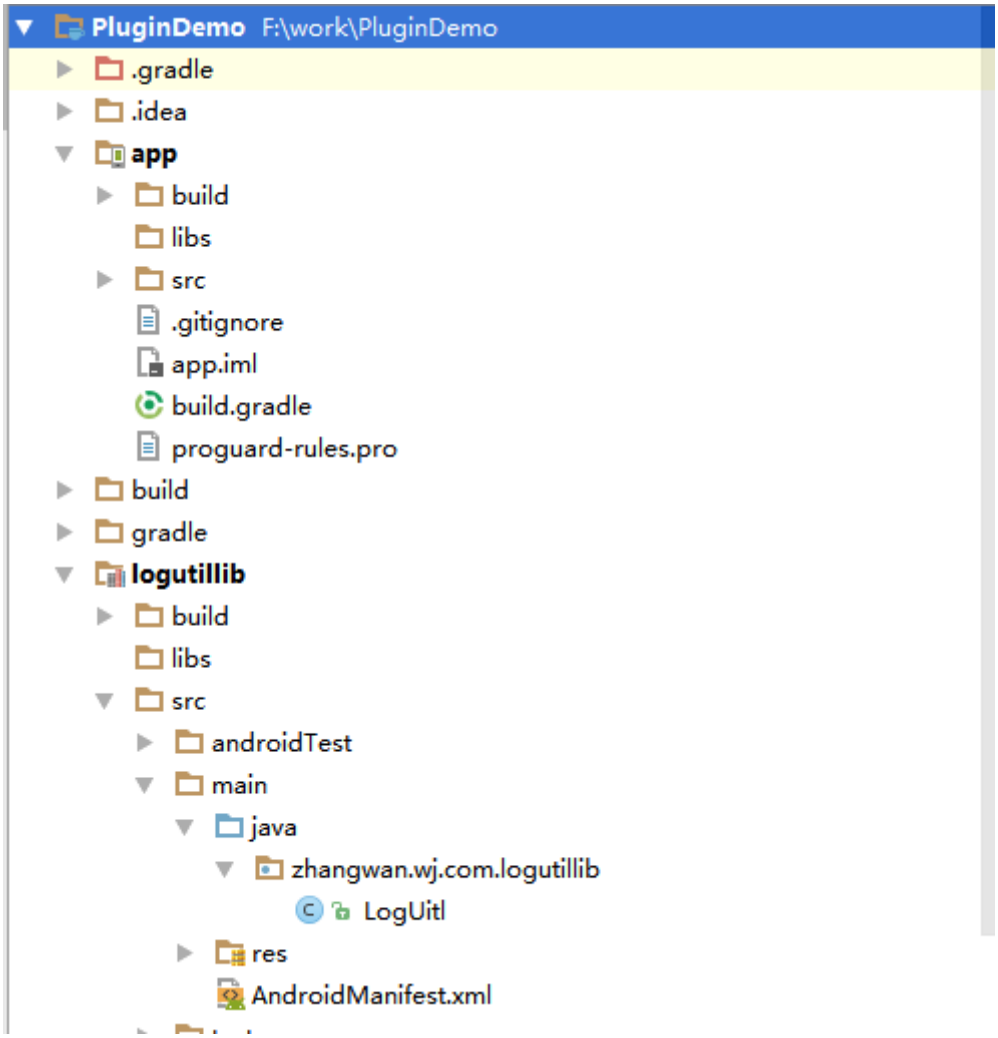
Android系统使用了ClassLoader机制来进行Activity等组件的加载；apk被安装之后，APK文件的代码以及资源会被系统存放在固定的目录（比如/data/app/package_name/1.apk)系统在进行类加载的时候，会自动去这一个或者几个特定的路径来寻找这个类；但是系统并不知道存在于插件中的Activity组件的信息，插件可以是任意位置，甚至是网络，系统无法提前预知，因此正常情况下系统无法加载我们插件中的类；因此也没有办法创建Activity的对象，更不用谈启动组件了。这个时候就需要使用动态加载技术了，关于Activity如何插件化，后面系列在说，本文讲了一个应用程序换肤的故事，虽然老套，但是对于理解动态加载技术很实用，读完之后你可以知道如何解决插件之中的资源加载问题。

一、动态加载dex的技术

Android使用Dalvik虚拟机加载可执行程序，所以不能直接加载基于class的jar，而是需要将class转化为dex字节码，从而执行代码。优化后的字节码文件可以存在一个.jar中，只要其内部存放的是.dex即可使用。

我们现在要实现的一个需求是：如何调用一个非本应用的java程序，如下：



app 与loutillib两个模块没有任何的依耐关系，在Module App中，我们想调用Loutillib中的LogUitl输出一条log。LogUitl如下，so easy。

```
public class LogUitl {
    public static final String TAG="LogUitl";
    private void  printLog(){
        Log.e(TAG,"这是来自另外一个dex中的log");
    }
}
```

所以我们要在运行时把LogUitl动态加载到app这个进程中， Android支持动态加载的两种方式是：DexClassLoader和PathClassLoader，DexClassLoader可加载jar/apk/dex，且支持从SD卡加载；PathClassLoader只能加载已经安装在Android系统内APK文件（/data/app 目录下），其它位置的文件加载的时候都会出现 ClassNotFoundException。 因为 PathClassLoader 会去读取 /data/dalvik-cache 目录下的经过 Dalvik 优化过的 dex 文件，这个目录的 dex 文件是在安装 apk 包的时候由 Dalvik 生成的，没有安装的时候，自然没有生成那个文件。

这里我们用DexClassLoader来加载，LogUitl所生成的dex文件。首先用gradle打出LogUtil的jar包。

```
task makeJar(type:Copy){
    delete 'build/libs/log.jar'
    from('build/intermediates/bundles/release/')
    into('build/libs/')
    include('classes.jar')
    rename ('classes.jar', 'log.jar')
    exclude('test/','BuildConfig.class','R.class')
    exclude{it.name.startsWith('R$');}
}

makeJar.dependsOn(build)
```

注意，这个jar还不能被加载，这个是基于class的jar,Dalvik虚拟机加载的是dex字节码，所以需要将class转化为dex字节码。这个需要用到dx命令，这个可以在Android\sdk\build-tools\23.0.0中找到，把log.jar拷贝到这个目录下，执行

```
dx --dex --output=new_log.jar log.jar
```

在执行

```
adb  push  new_log.jar  sdcard/
```

把这个new_log放进SDCARD中，这样dex的准备工作就OK了。以下是用DexClassLoader动态加载的代码。

```
public class MainActivity extends Activity {

    @Override
    protected void attachBaseContext(Context newBase) {
        super.attachBaseContext(newBase);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

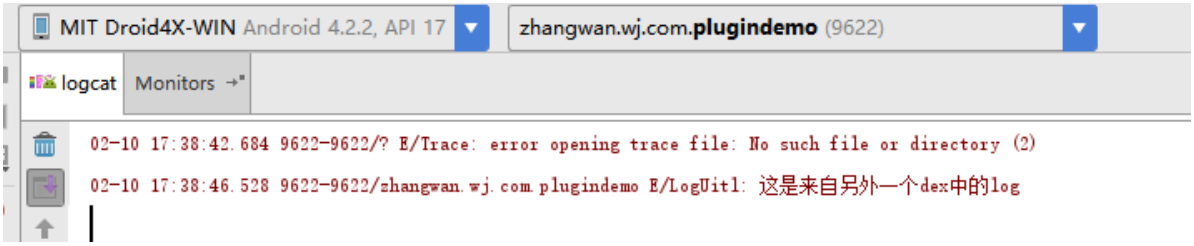
```
    public void start(View view) {
        //dex解压释放后的目录
        final File dexOutPutDir = getDir("dex", 0);
        //dex所在目录
        final String dexPath =
Environment.getExternalStorageDirectory().toString() + File.separator +
"new_log.jar";

        //第一个参数：是dex压缩文件的路径
        //第二个参数：是dex解压缩后存放的目录
        //第三个参数：是C/C++依赖的本地库文件目录,可以为null
        //第四个参数：是上一级的类加载器
        DexClassLoader classLoader=new
DexClassLoader(dexPath,dexOutPutDir.getAbsolutePath(),null,getClassLoader());

        try {
            final Class<?> loadClazz =
classLoader.loadClass("zhangwan.wj.com.logutillib.LogUitl");
            final Object o = loadClazz.newInstance();
            final Method printLogMethod =
loadClazz.getDeclaredMethod("printLog");
            printLogMethod.setAccessible(true);
            printLogMethod.invoke(o);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

执行结果：



发现成功的调用了printLog方法。有上面的基础，现在实现一个难一点的，如何给应用程序换肤，这个难体现在资源加载上。通常各种各样的皮肤都是一个个的apk文件，当用户需要哪个皮肤，就下载到本地，然后动态加载，但是当宿主程序调起未安装的皮肤插件apk的时候，插件中以R开头的资源都不能被访问，程序会抛出异常，无法找到某某id所对应的资源。这是因为加载资源都是通过Resourse来实现的，Resource对象是由Context得到的，我们知道一个app的工程的资源文件都会隐射到R文件中，而这个R文件的包名则是这个应用的包名，所以一个包名一般对应一个Context。宿主与皮肤插件的包名是不一样的，所以宿主Context找不到皮肤插件的资源。

二、应用换肤

1、皮肤程序准备

- **sky.apk**
- **children.apk**

准备两个apk,sky.apk中有一张名字为skin_one的背景图，显示的是蓝色的天空；children.apk中也有一张名字为skin_one的背景图，显示的是一个小孩。将这两个apk都push到SD里面，两套皮肤准备完成。

2、资源加载问题怎么解决

通过分析系统资源加载了解到，系统是通过ContextImpl中的getAssets与getResources加载资源的

```
/**
     * Returns an AssetManager instance for the application's package.
     * <p>
     * <strong>Note:</strong> Implementations of this method should return
     * an AssetManager instance that is consistent with the Resources instance
     * returned by {@link #getResources()}. For example, they should share the
     * same {@link Configuration} object.
     *
     * @return an AssetManager instance for the application's package
     * @see #getResources()
     */
    public abstract AssetManager getAssets();

    /**
     * Returns a Resources instance for the application's package.
     * <p>
     * <strong>Note:</strong> Implementations of this method should return
     * a Resources instance that is consistent with the AssetManager instance
     * returned by {@link #getAssets()}. For example, they should share the
     * same {@link Configuration} object.
     *
     * @return a Resources instance for the application's package
     * @see #getAssets()
     */
    public abstract Resources getResources();
```

ContextImpl中，也就是说，只要实现这两个方法，就可以解决资源问题了。

```
/**
     * 获取AssetManager    用来加载插件资源
     * @param pFilePath  插件的路径
     * @return
     */
    private AssetManager createAssetManager(String pFilePath) {
        try {
            final AssetManager assetManager = AssetManager.class.newInstance();
            final Class<?> assetManagerClazz =
Class.forName("android.content.res.AssetManager");
            final Method addAssetPathMethod =
assetManagerClazz.getDeclaredMethod("addAssetPath", String.class);
            addAssetPathMethod.setAccessible(true);
            addAssetPathMethod.invoke(assetManager, pFilePath);
            return assetManager;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
    //这个Resources就可以加载非宿主apk中的资源
    private Resources  createResources(String pFilePath){
        final AssetManager assetManager = createAssetManager(pFilePath);
        Resources superRes = this.getResources();
        return new Resources(assetManager, superRes.getDisplayMetrics(),
superRes.getConfiguration());
    }
```

3、动态加载皮肤apk

```
public class MainActivity extends Activity {

    private TextView  mSkinTv;

    private  boolean mChange=false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proxy);
        mSkinTv= (TextView) findViewById(R.id.skin_bg);
    }

    /**
     * 获取未安装apk的信息
     * @param context
     * @param pApkFilePath apk文件的path
     * @return
     */
    private String getUninstallApkPkgName(Context context, String pApkFilePath)
{
        PackageManager pm = context.getPackageManager();
        PackageInfo pkgInfo = pm.getPackageArchiveInfo(pApkFilePath,
PackageManager.GET_ACTIVITIES);
        if (pkgInfo != null) {
            ApplicationInfo appInfo = pkgInfo.applicationInfo;
            return appInfo.packageName;
        }
        return "";
    }

    public void switchSkin(View view) {
        String skinType="";
        if(!mChange){
            skinType= "sky.apk";
            mChange=true;
        }else {
            skinType= "children.apk";
            mChange=false;
        }
        final String path = Environment.getExternalStorageDirectory() +
File.separator + skinType;
        final String pkgName = getUninstallApkPkgName(this, path);
        dynamicLoadApk(path,pkgName);
    }
```

```
    private  void dynamicLoadApk(String pApkFilePath,String pApkPacketName){
        File file=getDir("dex", Context.MODE_PRIVATE);
        //第一个参数：是dex压缩文件的路径
        //第二个参数：是dex解压缩后存放的目录
        //第三个参数：是C/C++依赖的本地库文件目录,可以为null
        //第四个参数：是上一级的类加载器
        DexClassLoader  classLoader=new
DexClassLoader(pApkFilePath,file.getAbsolutePath(),null,getClassLoader());
        try {
            final Class<?> loadClazz = classLoader.loadClass(pApkPacketName +
".R$drawable");
          //插件中皮肤的名称是skin_one
            final Field skinOneField = loadClazz.getDeclaredField("skin_one");
            skinOneField.setAccessible(true);
            //反射获取skin_one的resousreId
            final int resousreId = (int) skinOneField.get(R.id.class);
            //可以加载插件资源的Resources
            final Resources resources = createResources(pApkFilePath);
            if (resources != null) {
                final Drawable drawable = resources.getDrawable(resousreId);
                mSkinTv.setBackground(drawable);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```