Android的动态加载

版本：2018/3/18-1(22:53)

1、动态加载技术是什么?

> 1. 也叫做 插件化技术
> 2. 在项目很庞大时，通过 插件化 能减轻应用的内存和CPU。
> 3. 也能实现 热插拔 ， 即在不发布新版本的情况下更新某些模块。

2、动态加载技术面临的三个基础性问题

> 1. 资源访问
> 2. Activity生命周期的管理
> 3. ClassLoader的管理

3、宿主和插件的区别?

> 1. 宿主 是指普通的apk
> 2. 插件 一般是指经过处理的 dex 或者 apk

4、插件化框架的大概思路?

> 1. 大多都是采用经过 特殊处理 的 apk 作为插件，处理方式和 编译以及打包环节 有关
> 2. 一般都需要使用 代理Activity ， 插件 的 Activity 的启动大多是借助一个 代理Activity 来实现的。

5、资源访问的相关概念?

> 1. 资源访问的问题 ： 宿主apk 调用 插件apk ， 是无法访问 插件 中的资源，因为R开头的资源在 宿主apk 中是没有的。
> 2. 不好方案1： 插件 中的资源在 宿主 中也预置一份---会增加宿主apk大小，且宿主apk也需要更新。
> 3. 不好方案2：将 插件 中的 资源 解压缩，并通过 文件流 去读取资源---不同资源的文件流格式不同，实际操作难度大。

6、资源访问问题的合理解决的思路?

> 1. Activity 的工作主要是通过 ContextImpl(Context的实现类) 完成--- Acitivity中的mBase 就是 ContextImpl 类型

2. Context 中两个抽象方法 getAssets()和getResources() 就是 Context 去获取资源的主要途径---实现这两个方法就可以解决资源问题。

7、资源访问问题的具体解决方式

```
/**============================*
 * 1、加载插件的apk中的资源
 *============================*/
 protected void loadResources(){
    try {
        //1. 通过反射获取到AssetManager的方法addAssetPath
        AssetManager assetManager = AssetManager.class.newInstance();
        Method addAssetPath = assetManager.getClass().getMethod("addAssetPath", String.class);
        //2. addAssetPath能将一个apk中的资源(资源目录orZip)加载到AssetManager
        addAssetPath.invoke(assetManager, mDexPath); //直接将插件apk的路径传入
        mAssetManager = assetManager;
    }catch (Exception e) {
        e.printStackTrace();
    }
    Resources superRes = super.getResources();
    //3. 再通过mAssetManager对象创建新的Resource对象---通过该对象就能访问插件Apk中的资源了
    mResources = new Resources(mAssetManager, superRes.getDisplayMetrics(), superRes.getConfigu
    mTheme = mResources.newTheme();
    mTheme.setTo(super.getTheme());
}
/**=====================================================*
 * 2、在代理Activity中实现`getAssets()和getResources()`
 *=====================================================*/
public AssetManager getAssets(){
    return mAssetManager == null ? super.getAssets() : mAssetManager;
}
public Resources getResources(){
    return mResources == null ? super.getResources() : mResources;
}
```

> 通过两步骤，就可以通过 R 来访问 插件中的资源 了

8、Activity生命周期通过反射管理

> 1. 通过 反射 去获取 Activity 的各种生命周期方法，然后在 代理Activity 中去调用 插件Activity 对应的生命周期即可。
> 2. 缺点是 代码复杂 而且有 性能开销 。
> 3. 代理Activity 中调用 插件Activity 如下：

```
@Override
protected void onResume() {
    super.onResume();
    //1. 获取插件Activity的生命周期方法
    Method onResume = mActivityLifecircleMethods.get("onResume");
    if(onResume != null){
       //2. 调用该方法
        onResume.invoke(mRemoteActivity, new Object[]{});
    }
}
```

9、Activity生命周期通过接口方式管理：

> 1. 接口方式 将 生命周期方法 提取作为一个接口，如 DLPlugin
> 2. 通过 代理Activity 去调用 插件Activity 的生命周期方法。
> 3. 本质上 代理Acitivity 就是接口 DLPlugin 的实现：

```
public interface DLPlugin{
    public void onStart();
    ...
}
//代理Activity中调用了插件Activity中的方法
@Override
protected void onStart() {
    mRemoteActivity.onStart();
    super.onStart();
}
```

10、插件ClassLoader的管理

> 1. 为了对 多插件 进行支持，需要合理管理各个插件的 DexClassLoader .
> 2. 需要避免多个 ClassLoader 加载同一个类时所引发的类型转换错误.
> 3. 通过将不同插件的 ClassLoader 存储在一个 HashMap 中，就可以保证不同插件中的类彼此互不干扰.

```
public class DLClassLoader extends DexClassLoader{
    private static final String TAG = DLClassLoader.class.getName();
    private static final HashMap<String, DLClassLoader> mPluginClassLoaderHashMap = new Has

  /**============================*
   * 能获取到属于不同apk的ClassLoader
   *============================*/
    public static DLClassLoader getClassLoader(String dexPath, Context context, ClassLoader
        //1. 如果已经在HashMap中直接返回
        DLClassLoader dlClassLoader = mPluginClassLoaderHashMap.get(dexPath);
        if(dlClassLoader != null){
            return dlClassLoader;
        }
        //2. 没有加载过，获取dex的输出目录
        File dexOutputDir = context.getDir("dex", Context.MODE_PRIVATE);
        final String dexOutputPath = dexOutputDir.getAbsolutePath();
        //3. 通过dex路径，dex输出目录和父loader创建DLClassLoader对象，并存入HashMap
        dlClassLoader = new DLClassLoader(dexPath, dexOutputPath, null, parentLoader);
        mPluginClassLoaderHashMap.put(dexPath, dlClassLoader);
        return dlClassLoader;
    }

    public DLClassLoader(String dexPath, String optimizedDirectory, String librarySearchPat
        super(dexPath, optimizedDirectory, librarySearchPath, parent);
    }
}
```

参考资料

1. DL : Apk动态加载框架