Android插件化介绍-2-插件化介绍及方案对比

- 动态加载的原理
- Activity、Service的插件化方案
- 资源插件化的方案
- BoardcastReceiver、ContentProvider的插件化方案
- so的插件化方案
- 几种不同的插件化方案对比及应用场景

BroadcastReceiver如何插件化

- 广播分为动态广播接收者和静态广播接收者
- 静态广播是如何注册的？静态广播接收者是PackageManagerService在安装apk或者开机的时候扫描已安装apk并解析配置文件保存在mReceivers中的
- 动态广播是如何注册的？动态广播接收者是通过Binder调用注册在ActivityManagerService中的，保存在mRegisteredReceivers变量中
- 由于插件APK未安装，因此无法通过静态广播的方式，我们可以考虑自己解析配置文件并以动态方式的注册，但这种方式无法满足低版本其他应用发送广播启动进程，高版本去掉了这个特性。

```
Class packageParseClz = Class.forName("android.content.pm.PackageParser");
Object packageParser = packageParseClz.newInstance();
Method parseMethod = packageParseClz.getDeclaredMethod( name: "parsePackage", File.class, int.class);
parseMethod.setAccessible(true);
Object packageObject = parseMethod.invoke(packageParser,  ...args: apkFile, 1 << 2);
Class packageClz = Class.forName("android.content.pm.PackageParser$Package");
Field receiversField = packageClz.getDeclaredField( name: "receivers");
receiversField.setAccessible(true);
ArrayList receives = (ArrayList) receiversField.get(packageObject);

Class componentClz = Class.forName("android.content.pm.PackageParser$Component");
Field intents = componentClz.getDeclaredField( name: "intents");
intents.setAccessible(true);
Field classNameField = componentClz.getDeclaredField( name: "className");
classNameField.setAccessible(true);
for (int i = 0; i < receives.size(); i++) {
    ArrayList<IntentFilter> intentFilters = (ArrayList<IntentFilter>) intents.get(receives.get(i));
    String className = (String) classNameField.get(receives.get(i));
    registerReceiver((BroadcastReceiver) getClassLoader().loadClass(className).newInstance(),intentFilters.get(0));
}
```

ContentProvider如何插件化

- 当我们调用ContentResolver去做数据操作的时候，会先去查询本进程内已经安装了的ContentProvider，找不到的情况下通过PMS查找，需要时启动目标ContentProvider依托的进程

```
13617-13617/com.guolei.plugindemo E/ActivityThread: Failed to find provider info for com.guolei.plugin_1.provider
13617-13617/com.guolei.plugindemo E/ActivityThread: Failed to find provider info for com.guolei.plugin_1.provider
```

```
        at com.google.android.gms.gcm.d.run(Unknown Source)
: Unable to get context for package com.guolei.plugin_1 while loading content provider com.guolei.plugin_1.PluginContentProvider
ice: [9328] e.a: Unable to register for GCM
```

对于本应用的情况

- 我们要解决的问题是将插件中的ContentProvider安装到本进程，已安装的ContentProvider信息存放在ActivityThread的mProviderRefCountMap中，做法是按照ActivityThread启动时安装ContentProvider的方式去做。

```
Class providerClz = Class.forName("android.content.pm.PackageParser$Provider");
Field providerInfoField = providerClz.getDeclaredField( name: "info");
providersField.setAccessible(true);
List<ProviderInfo> providerInfos = new ArrayList<>();
for (int i = 0; i < providers.size(); i++) {
    ProviderInfo providerInfo = (ProviderInfo) providerInfoField.get(providers.get(i));
    providerInfo.applicationInfo = getApplicationInfo();
    providerInfos.add(providerInfo);
}
Class contextImplClz = Class.forName("android.app.ContextImpl");
Field mMainThread = contextImplClz.getDeclaredField( name: "mMainThread");
mMainThread.setAccessible(true);
Object activityThread = mMainThread.get(this.getBaseContext());
Class activityThreadClz = Class.forName("android.app.ActivityThread");
Method installContentProvidersMethod = activityThreadClz.getDeclaredMethod( name: "installContentProviders", Context.class, List.class);
installContentProvidersMethod.setAccessible(true);
installContentProvidersMethod.invoke(activityThread,  ...args: this, providerInfos);
```

1.首先解析APK文件获取ProviderInfo

2.然后修改ProviderInfo的applicationInfo属性,

3.调用ActivityThread#installContentProvider进行安装

对于跨应用的情况

- 由于跨应用情况下，插件APK未安装，因此PMS中没有插件APK的ContentProvider信息。我们必须在我们的宿主APK，注册一个代理ContentProvider，通过代理ContentProvider去解决这个问题

- 我们需要预先定义好协议

```java
@Override
public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String selection, @Nullable String[] s
    //content://com.guolei.delegate.content/com.guolei.plugin_1.provider
    if (uri.getPath() != null) {
        return getContext().getContentResolver().query(Uri.parse("content://" + uri.getPath().substring(1)),
                projection, selection, selectionArgs, sortOrder);
    } else {
        Log.e( tag: "plugin",  msg: "query: " + "delegate");
        return null;
    }
}
```

so文件如何插件化

- so是如何加载的,Runtime用过BaseDexClassLoader#findLibrary去查找，并且通过doLoad去加载，而BaseDexClassLoader中则是通过DexPatchList去做的

- 我们要做的如下操作。
  - 将apk文件中的so文件释放到某个文件夹下
  - 构造一个此so对应的Element，填充到BaseDexClassLoader的nativeLibraryPathElements中
  - 构造一个上面的文件夹对应的File，填充到BaseDexClassLoader的nativeLibraryDirectories中

```
// findNativeLib
Method findLibMethod = elementClz.getDeclaredMethod( name: "findNativeLibrary",String.class);
findLibMethod.setAccessible(true);
  Object soElement = constructor.newInstance(new File("/sdcard/"), true, apkFile, DexFile.loadDex(apkFile.getCa
          file.getAbsolutePath(), 0));
  findLibMethod.invoke(pluginElement,System.mapLibraryName("native-lib"));
ZipFile zipFile = new ZipFile(apkFile);
ZipEntry zipEntry = zipFile.getEntry( name: "lib/armeabi/libnative-lib.so");
InputStream inputStream = zipFile.getInputStream(zipEntry);
File outSoFile = new File(getFilesDir(),  child: "libnative-lib.so");
if (outSoFile.exists()) {
    outSoFile.delete();
}
FileOutputStream outputStream = new FileOutputStream(outSoFile);
byte[] cache = new byte[2048];
int count = 0;
while ((count = inputStream.read(cache)) != -1) {
    outputStream.write(cache,  off: 0, count);
}
outputStream.flush();
outputStream.close();
inputStream.close();
// 构造Element
Object soElement = constructor.newInstance( ...initargs: getFilesDir(), true, null, null);
  findLibMethod.invoke(soElement,System.mapLibraryName("native-lib"));

// 将soElement填充到nativeLibraryPathElements中,
Field soElementField = clz.getDeclaredField( name: "nativeLibraryPathElements");
soElementField.setAccessible(true);
Object[] soElements = (Object[]) dexElementsField.get(pathList);
Object[] newSoElements = (Object[]) Array.newInstance(elementClz,  length: soElements.length + 1);
Object[] toAddSoElementArray = new Object[]{soElement};
System.arraycopy(soElements,  srcPos: 0, newSoElements,  destPos: 0, soElements.length);
// 插件的那个element复制进去
System.arraycopy(toAddSoElementArray,  srcPos: 0, newSoElements, soElements.length, toAddElementArray.length);
soElementField.set(pathList, newSoElements);

//将so的文件夹填充到nativeLibraryDirectories中
Field libDir = clz.getDeclaredField( name: "nativeLibraryDirectories");
libDir.setAccessible(true);
List libDirs = (List) libDir.get(pathList);
libDirs.add(getFilesDir());
libDir.set(pathList,libDirs);
```

插件化方案-代码层次

- 多ClassLoader架构
- 单一ClassLoader架构
- 混合ClassLoader架构，破坏双亲委托机制

插件化方案-资源层次

- 单Resource，存在资源冲突问题
  - 修改aapt
  - 修改arsc和R文件
- 多Resource，不存在资源冲突的问题