HenCoder Plus 第 26 课 讲义

组件化、插件化和热更新

什么是组件化

拆成多个 module 开发就是组件化

什么是插件化

App 的部分功能模块在打包时并不以传统方式打包进 apk 文件中，而是以另一种形式二次封装进 apk 内部，或者放在网络上适时下载，在需要的时候动态对这些功能模块进行加载，称之为插件化。

这些单独二次封装的功能模块 apk ，就称作「插件」，初始安装的 apk 称作「宿主」。

插件化是组件化的更进一步推进。

插件化基础：反射

反射的写法

```
try {
    Class utilClass = Class.forName("com.hencoder.demo.hidden.Util");
    Constructor utilConstructor = utilClass.getDeclaredConstructors()[0];
    utilConstructor.setAccessible(true);
    Object util = utilConstructor.newInstance();
    Method shoutMethod = utilClass.getDeclaredMethod("shout");
    shoutMethod.setAccessible(true);
    shoutMethod.invoke(util);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InstantiationException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
```

反射的目的

Java 既然提供了可见性关键字 `public` `private` 等等，用来限制代码之间的可见性，为什么又要提供反射功能?

- 可见性特性的支持不是为了代码不被坏人使用，而是为了程序开发的简洁性。安全性的话，可见性的支持提供的是 Safety 的安全，而不是 Security 的安全。即，可见性的支持让程序更不容易写出 bug，而不是更不容易被人入侵。
- 反射的支持可以让开发者在可见性的例外场景中，可以突破可见性限制来调用自己需要的 API。这是基于对开发者「在使用反射时已经足够了解和谨慎」的假设的。
- 所以，可见性的支持不是为了防御外来者入侵，因此反射功能的支持并没有什么不合理。

关于 DEX：

- class：java 编译后的文件，每个类对应一个 class 文件
- dex：Dalvik EXecutable 把 class 打包在一起，一个 dex 可以包含多个 class 文件
- odex：Optimized DEX 针对系统的优化，例如某个方法的调用指令，会把虚拟的调用转换为使用具体的 index，这样在执行的时候就不用再查找了
- oat：Optimized Android file Type。使用 AOT 策略对 dex 预先编译（解释）成本地指令，这样再运行阶段就不需再经历一次解释过程，程序的运行可以更快
    - AOT：Ahead-Of-Time compilation 预先编译

插件化原理：动态加载

通过自定义 ClassLoader 来加载新的 dex 文件，从而让程序员原本没有的类可以被使用，这就是插件化的原理。

例如：把 Utils 拆到单独的项目，打包 apk 作为插件引入

```
File f = new File(getCacheDir() + "/demo-debug.apk");
if (!f.exists()) {
    try {
        InputStream is = getAssets().open("apk/demo-debug.apk");
        int size = is.available();
        byte[] buffer = new byte[size];
        is.read(buffer);
        is.close();


        FileOutputStream fos = new FileOutputStream(f);
        fos.write(buffer);
        fos.close();
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

DexClassLoader classLoader = new DexClassLoader(f.getPath(),
getCodeCacheDir().getPath(), null, null);
```

```
try {
    Class oldClass =
classLoader.loadClass("com.hencoder.demo.hidden.Util");
    Constructor utilConstructor = oldClass.getDeclaredConstructors()[0];
    utilConstructor.setAccessible(true);
    Object util = utilConstructor.newInstance();
    Method shoutMethod = oldClass.getDeclaredMethod("shout");
    shoutMethod.setAccessible(true);
    shoutMethod.invoke(util);

    Class activityClass =
classLoader.loadClass("com.hencoder.demo.MainActivity");
    startActivity(new Intent(this, activityClass));
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (IllegalAccessException e) {
    e.printStackTrace();
} catch (InstantiationException e) {
    e.printStackTrace();
} catch (InvocationTargetException e) {
    e.printStackTrace();
}
```

问题一：未注册的组件（例如 Activity）不能打开

- 解决方式一：代理 Activity
- 解决方式二：欺骗系统
- 解决方式三：重写 gradle 打包过程，合并 AndroiManifest.xml

问题二：资源文件无法加载

解决方式：自定义 AssetManager 和 Resources 对象

```
private AssetManager createAssetManager(String dexPath) {
    try {
        AssetManager assetManager = AssetManager.class.newInstance();
        Method addAssetPath =
assetManager.getClass().getMethod("addAssetPath", String.class);
        addAssetPath.invoke(assetManager, dexPath);
        return assetManager;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

```
private Resources createResources(AssetManager assetManager) {
    Resources superRes = mContext.getResources();
    Resources resources = new Resources(assetManager,
superRes.getDisplayMetrics(), superRes.getConfiguration());
    return resources;
}
```

插件化有什么用?

- 早期：解决 dex 65535 问题。谷歌后来也出了 multidex 工具来专门解决
- 一说：懒加载来减少软件启动速度：有可能，实质上未必会快
- 一说：减小安装包大小：可以
- 一说：项目结构拆分，依赖完全隔离，方便多团队开发和测试，解决了组件化耦合度太高的问题：这个使用模块化就够了，况且模块化解耦不够的话，插件化也解决不了这个问题
- 动态部署：可以
    - Android App Bundles：属于「模块化发布」。未来也许会支持动态部署，但肯定会需要结合应用商店（即 Play Store，或者未来被更多的商店所支持）
- bug 热修复：可以