Computer Science and Application 计算机科学与应用, 2017, 7(1), 36-41
Published Online January 2017 in Hans. http://www.hanspub.org/journal/csa
http://dx.doi.org/10.12677/csa.2017.71005

Hans汉斯

Research on Android Dynamic Loading Technology

Zhengzhi Li

College of Computer Science, Sichuan University, Chengdu Sichuan
Email: aaronlzz@qq.com

Received: Dec. 30th, 2016; accepted: Jan. 15th, 2017; published: Jan. 18th, 2017

Abstract

From the beginning of the 21st century, with the rapid development of mobile Internet, the number of global Internet users increased rapidly. In order to meet the needs of different users in different scenarios, mobile applications continue to emerge; while its continuous improvement, people also have a dependence on mobile applications. Mobile applications have been fully integrated into the people's eating, wearing, living, traveling and other aspects. So, in order to adapt to the ever-changing market demand and the use of scenarios, various service providers have to quickly iterative products. Conversely, because the product is too fast iteration, users need to constantly update and download applications, resulting in poor user experience. In order to enable users do not have to reinstall APK to achieve the application upgrade update function, you need to use dynamic loading technology in Android projects. In this paper, the dynamic loading technology of Android is studied.

Keywords

Dynamic Loading, Update App, Update, Mobile Internet

Android动态加载技术的研究

李政志

四川大学计算机学院，四川 成都
Email: aaronlzz@qq.com

收稿日期：2016年12月30日；录用日期：2017年1月15日；发布日期：2017年1月18日

摘 要

21世纪，移动互联网飞跃式的发展，全球互联网用户数量急速上升。为了满足不同用户在不同使用场景

文章引用: 李政志. Android 动态加载技术的研究[J]. 计算机科学与应用, 2017, 7(1): 36-41.
http://dx.doi.org/10.12677/csa.2017.71005

下的需求，移动应用不断推陈出新，在其不断完善的同时，人们对移动应用也产生了依赖性。移动应用已经全面的深入到人们的吃、穿、住、行等方方面面。所以，为了适应市场的不断变化的需求和使用场景，各类服务提供商不得不快速的迭代产品。反之，由于产品的过于快速的迭代，使用户需要不断的更新和下载应用，造成用户体验较差。通过对android虚拟机加载Class的机制研究实现动态加载，让用户不用重新安装APK就实现应用升级更新功能。

关键词

动态加载，应用更新，更新，移动互联网

1. 研究背景

随着智能手机和移动互联网的普及和进一步快速发展，我国的互联网行业正在蒸蒸日上，人们对移动应用和产品在不同的使用场景下的需求越来越旺盛。各类移动互联网应用服务商为了满足用户需求，必须快速的迭代和更新产品[1]。或者像现在的很多 APP 一样，都有一个启动欢迎页，在不同的节日和运营活动的情况下，会有不同的广告页出现。再如，在一个应用发布后，突发一个比较严重的问题，需要紧急修复或者更新。

再者，在早期的安卓市场上发布应用，如果应用中含有广告，可能会审核不通过，那么一些个人开发者会在服务器端配置一个开关，审核时关闭开关，应用就不显示广告了，通过审核后，再通过服务器端把广告开关打开，这样就可以很好的规避应用市场的审核。之后，应用市场通过扫描 APK 内的 manifest 甚至 dex 文件，以此来确定开发者是否在 APK 包里植入广告代码。在服务器端配置开关参数的方法不行了，很多开发者就另辟蹊径。在应用的原生 APK 代码内部写入广告代码，在用户下载安装运行后，再从服务器下载广告代码，运行，实现广告的功能。该方法是可行的，这就是动态加载[2]。

那么接下来我们将具体来说明动态加载。1、应用能够通过在本地加载一些不存在的文件来实现特定的功能；2、这些可执行文件具有可替换性；3、动态加载不包括静态资源(如：启动图，主题，广告在服务器端的控制参数开关等)；4、调用外部的 Dex 文件是 Android 动态加载的核心思想，在极端情况下，Dex 文件可以作为 Android APK 的一个程序入口，都可以通过在服务器下载来实现 Dex 文件完成相应的功能[3]。

2. 传统 PC 软件的动态加载技术

传统 PC 端，动态加载技术被广泛使用，比如有些输入法，在初次安装的时候没有截图功能，在用户第一次使用的时候，会自行从服务器端下载安装，这样就能使用截图功能了。此外，DLL 文件(Dynamic Link Library)在许多软件的安装目录中大量存在，一些特定的功能就是 PC 软件通过调用这些 DLL 里的代码执行的，这些技术就是一种动态加载[4]。在 JAVA 中，JAR 作为其可执行文件，运行于虚拟机 JVM 上，虚拟机则通过 ClassLoader 加载 JAR 文件，并执行其中的代码。所以在 JAVA 中的动态加载，也是通过动态调用 JAR 文件来实现的[5]。

3. Android 应用的动态加载技术定义

与 JAVA 程序类似，Android 应用把虚拟机换成了 Dalvik/ART，把 JAR 换成了 Dex。那么我们可以

考虑，在 Android 中怎样来实现动态加载。Android APP 运行时，是不是也可以通过下载新的应用，或者通过调用 Dex 文件来实现[6]。

可是在 Android 上实现并不那么容易，如果用户下载新的 APK 而不安装，就不可能能够运行。如果用户手动运行这个 APK 进行安装，就是纯粹的安装了一个新的应用。然后启动使用这个应用。这样的方式，就不是上文所说的动态加载了。

在 APK 文件中，往往存在一个或者多个 Dex 文件，所有代码都会被编译到 Dex 中，应用的所有功能，都是通过执行这些 Dex 来实现的。APK 被构建出来后，无法再更换其 Dex 文件的，但是，可以通过储存在外部服务器通过网络下载后，通过加载外部的 Dex 文件来实现动态加载[7]。

4. Android 动态加载的类型

在 Android 项目中，动态加载技术主要有两种技术实现[8]。

其一，为动态加载.so 库。动态加载就是用在了 Android 的 NDK 中，通过 JNI 动态加载.so 库中其封装好的方法。JNI 运行于 Native 层，一般由 C++编译，所以其执行效率比 JAVA 代码高很多，因为 JAVA 代码执行在虚拟机中。所以，Android 中为了完成一些对性能要求很高的功能，经常会使用动态加载.so 库来完成(如：图片高斯模糊处理，Bitmap 的解码等)。另外，.so 库在很多安全领域得到广泛使用，因为其是由 C++编译的，相比 Smail 更难破解，只能被反编译成汇编代码。需要特别注意的是，虽然在一般情况下我们把.so 库一并打包在 APK 中，但是.so 库也是从外部存储文件加载的。

其二，基于 ClassLoader 的动态加载.dex/jar/apk。就是我们在上文提到过的“在 Android 中动态加载由 JAVA 代码编译的 Dex 包并执行其中的代码逻辑”，这个技术在常规的 Android 开发比较少用到，而本文主要写的就是这种动态加载方式。

两种技术的区别：第一种本质上是 java 调用 c++的类库属于 api 调用，一般把复杂运算和高性能要求的代码用 c++实现打包成 so 库供 java 调用。第二种是 java 调用 java，在应用运行时需要把 java 代码编译成 class 文件加载到虚拟机，通过虚拟机加载 class 文件的顺序在每次运行时首先加载我们修改过的 java 文件(同名的 java 文件只加载一次)实现更新覆盖而不用重新安装应用。

5. Android 动态加载的方法和过程

无论是哪种动态加载方式，其基本原理是相通的，都是在程序运行时加载一些外部的可执行文件，然后调用这些文件的某个方法实现业务逻辑[9]。但是，需要注意的是，处于对安全的考虑，由于文件是可执行的，Android 并不允许直接在手机外部储存加载这类可执行文件。对于这些外部的可执行文件，我们为了确保库不会第三方应用而已修改和拦截，可以先把他们拷贝到 data/packagename/内部存储文件路径，然后再将其加载到当前的运行环境中，并调用需要的方法来实现相应的业务逻辑，从而实现动态调用。

与 JVM 不同，Android 的虚拟机不能用 ClassCload 直接加载 dex，而是要用 DexClassLoader 或者 PathClassLoader,他们都是 ClassLoader 的子类，这两者的区别在于，DexClassLoader 可以加载/jar/apk/dex，可以从 SK 卡中加载未安装的 apk，而 PathClassLoader 要传入系统中 apk 的存放的 Path，所以只能加载已经安装了的 APK 文件[10]。

类加载器说明：如图 1，DexClassLoader 和 PathClassLoader 都属于符合双亲委派模型的类加载器,他们并没有重载 loadClass 方法，它们在加载一个类之前，会去检查自己以及自己以上的类加载器(BaseDexClassLoader 和 ClassLoader)是否已经加载了这个类。如果已经加载过了，就会直接将之返回，并不会重复加载。DexClassLoader 和 PathClassLoader 其实都是通过 DexFile 这个类来实现类加载的。Dalvik

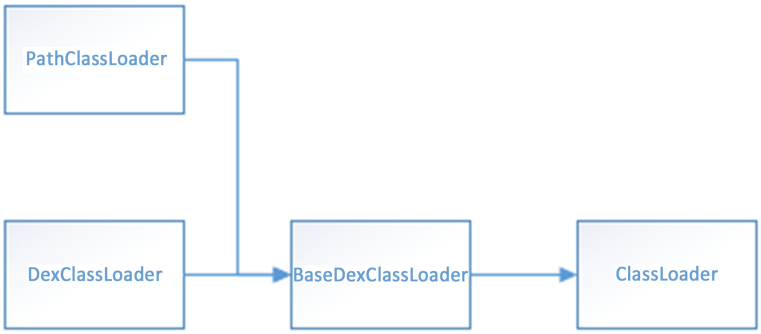

Figure 1. ClassLoader structure
图 1. ClassLoader 结构图

虚拟机识别的是 dex 文件(不是 class 文件)，所以类加载的文件也只能是 dex 文件，或者包含有 dex 文件的 apk、jar 文件。区别在于 DexClassLoader 需要提供一个可写的 outpath 路径，用来释放 apk 或 jar 包中的 dex 文件。PathClassLoader 不能主动从 zip 包中释放出 dex，因此只支持直接操作 dex 格式文件。而 DexClassLoader 可以支持 apk、jar 和 dex 文件，并且会在指定的 outpath 路径释放出 dex 文件[11]。

接着来看看 DexClassLoader 和 PathClassLoader 的构造方法：

```
// DexClassLoader.java
public class DexClassLoader extends BaseDexClassLoader {
    public DexClassLoader(String dexPath, String optimizedDirectory,
            String libraryPath, ClassLoader parent) {
        super(dexPath, new File(optimizedDirectory), libraryPath, parent);
    }
}

// PathClassLoader.java
public class PathClassLoader extends BaseDexClassLoader {
    public PathClassLoader(String dexPath, ClassLoader parent) {
        super(dexPath, null, null, parent);
    }

    public PathClassLoader(String dexPath, String libraryPath,
            ClassLoader parent) {
        super(dexPath, null, libraryPath, parent);
    }
}
```

Dex 加载过程[12]：

1.首先通过 new DexClassLoader 传入四个参数 dexPath (需要被加载的文件地址)、optimizedDirectory:dex (文件被加载优化后的 dex 存放路径)、libraryPath (包含 libraries 的目录列表)、parent (父类构造器)。

2.在 DexClassLoader 构造器中会直接调用父类的构造器，将 dex 文件路径传值给 originalPath 同时构造一个 DexPathList 对象。

3.在 DexPathList 构造器中：把父类构造器赋值给 definingContext、把 dexPath 分割后对应的 file 数组赋值给 dexElements、把 libraryPath 分割成数组加上系统 so 库的目录赋值给 nativeLibraryDirectories。

4.dexElements 是一个简单的实体类，包含 File、DexFile、ZipFile (jar、zip、apk 形式的 file)。在 DexFile 中会调用 native 方法 openDexFile 打开具体的 file 并输出到优化路径。

Android 在运行时使用 ClassLoader 动态加载外部的 Dex 文件非常简单，不用覆盖安装新的 APK，就可以更改 APP 的代码逻辑。但是 Android 却很难使用插件 APK 里的 res 资源，这意味着无法使用新的 XML 布局等资源，同时由于无法更改本地的 Manifest 清单文件，所以无法启动新的 Activity 等组件。不过可以先把要用到的全部 res 资源都放到主 APK 里面，同时把所有需要的 Activity 先全部写进 Manifest 里，只通过动态加载更新代码，不更新 res 资源，如果需要改动 UI 界面，可以通过使用纯 Java 代码创建布局的方式绕开 XML 布局，也可以使用 Fragment 代替 Activity。某种程度上，简单的动态加载功能已经能满足部分业务需求了，特别是一些早期的 Android 项目，那时候 Android 的技术还不是很成熟，而且早期的 Android 设备更是有大量的兼容性问题，只有这种简单的加载方式才能稳定运行。这种模式的框架比较适用一些 UI 变化比较少的项目，比如游戏 SDK，基本就只有登陆、注册界面，而且基本不会变动，更新的往往只有代码逻辑。

我们可以通过动态加载，让现在的 Android 应用启动一些“新”的 Activity，甚至不用安装就启动一个“新”的 APK (原来的 APK 叫“主 APK”，新的 APK 称为“插件 APK”)。主 APK 需要先注册一个空壳的 Activity 用于代理执行插件 APK 的 Activity 的生命周期。主要有一下特点：

1. 主 APK 可以启动未安装的插件 APK；
2. 插件 APK 也可以作为一个普通 APK 安装并且启动；
3. 插件 APK 可以调用主 APK 里的一些功能；
4. 主 APK 和插件 APK 都要接入一套指定的接口才能实现以上功能；

6. 动态加载技术的作用与代价

凡事都有两面性，特别是这种非官方支持的非常规开发方式，在采用前一定要权衡清楚其作用与代价。如果决定了要采用动态加载技术，个人推荐可以现在实际项目的一些比较独立的模块使用这种框架，把遇到的一些问题解决之后，再慢慢引进到项目的核心模块；如果遇到了一些无法跨越的问题，要有能够迅速投入生产的替代方案。

作用：

1、 规避 APK 覆盖安装升级的局限性，提高了用户体验，而且也能规避一些安卓市场的限制；
2、 动态修复一些比较紧急的 bug；
3、 提高启动速度，因为插件模块可以在需要时才初始化，叫做懒加载；
4、 主项目和插件项目可以并行开发，特别是应用体积较大时，可以把一些模块改为动态加载，以插件的形式，这样也可以减少主项目的体积，提高项目的编译速度；
5、 减少主项目的 DEX 方法数量，彻底解决 65535 问题；
6、 从项目管理的角度上来说，分割模块的方式，也做到了代码分离，大大降低了模块之间的耦合度，利于代码管理和 bug 走查；
7、 在 Android 应用的推广其他应用时，可以利用动态加载技术，让用户在不用下载新的 APK 的情况下体验新应用的功能。

代价：

1、 开发方式繁琐，和常规开发不同；
2、 非常规的开发方式，部分 Android ROM 可能已经改动了 Framework 层的代码，而有些框架却使用反射强行改动了这些代码，所以存在兼容性的风险，特别是在一些老旧机型上；
3、 随着动态加载框架复杂程度的加深，项目的构建过程也变得复杂，有可能要主项目和插件项目分别构建，再整；

4、 由于插件项目可能是独立开发，可能遇到主项目和插件项目的运行环境的不同，代码逻辑容易出现bug，而且在主项目中调试插件十分繁琐；

5、 兼容性问题，也是采用动态加载的产检在使用系统资源时经常发生的。

7. 结束语

本文详细阐述了移动互联网时代下 Android 应用的动态加载技术。从 android 的 ClassLoader 类入手，提出了可实行的解决方案，一定程度上的解决了应用更新快用户反复安装的烦恼。也提出了动态加载技术的优势和劣势。系统中用到的技术在应用快速迭代上性能和用户体验的提高上具有重要的价值和广阔的应用前景。

参考文献 (References)

[1] 吴吉义, 李文娟, 黄剑平, 章剑林, 陈德人. 移动互联网研究综述[J]. 中国科学: 信息科学, 2015(1): 45-69.

[2] Feng, Q.-C., Wen, Q.-Y. and Fan, Y.-J. (2011) State Key Laboratory of Networking and Switching Technology Beijing University of Posts and Telecommunications Beijing 100876, China. A Systemic Code-Protecting Methodology for the Dex File on Android platform. IEEE Beijing Section, China, Guangdong University of Technology, China. *Proceedings of* 2011 *IEEE International Conference on Intelligent Computing and Intelligent Systems* (*ICIS* 2011) Vol. 02.IEEE Beijing Section, China, Guangdong University of Technology, China, 4.

[3] 卿斯汉. Android 安全研究进展[J]. 软件学报, 2016(1): 45-71.

[4] 陈璟, 陈平华, 李文亮. Android 内核分析[J]. 现代计算机(专业版), 2009(11): 112-115.

[5] Zhang, X.L., Breitinger, F. and Baggil, I. (2016) Rapid Android Parser for Investigating DEX Files (RAPID). *Digital Investigation*.

[6] 张峰, 李基亮. 校园私有云存储方案的探索[J]. 华东师范大学学报(自然科学版), 2015(S1): 139-145.

[7] Phillips, B., Stewart, C., Hardy, B., *et al*. (2015) Android Programming: The Big Nerd Ranch Guide. Big Nerd Ranch.

[8] 曾健平, 邵艳洁. Android 系统架构及应用程序开发研究[J]. 微计算机信息, 2011(9): 1-3.

[9] 王良, 王伟平, 孟丹. FVS k-匿名: 一种基于 k-匿名的隐私保护方法[J]. 高技术通讯, 2015(3): 228-223.

[10] Weiss, M.A. (2008) Data Structures and Algorithm Analysis in Java: International Edition, 3/E. *Journal of the American Chemical Society*, **130**, 2156-2157.

[11] Gvero, I. (2013) Core Java Volume I: Fundamentals, 9th Edition by Cay S. Horstmann and Gary Cornell. *Acm Sigsoft Software Engineering Notes*, **38**, 33-33.

[12] Peierls, T., Goetz, B., Bloch, J., *et al*. (2006) Java Concurrency in Practice. Java Concurrency in Practice. Addison-Wesley, 1171-1177(7).