Android插件化：从入门到放弃作者包建强发布于 2016年7月14日|首届应用性能管理大会APMCon，聚焦国内外性能优化先进技术。讨论

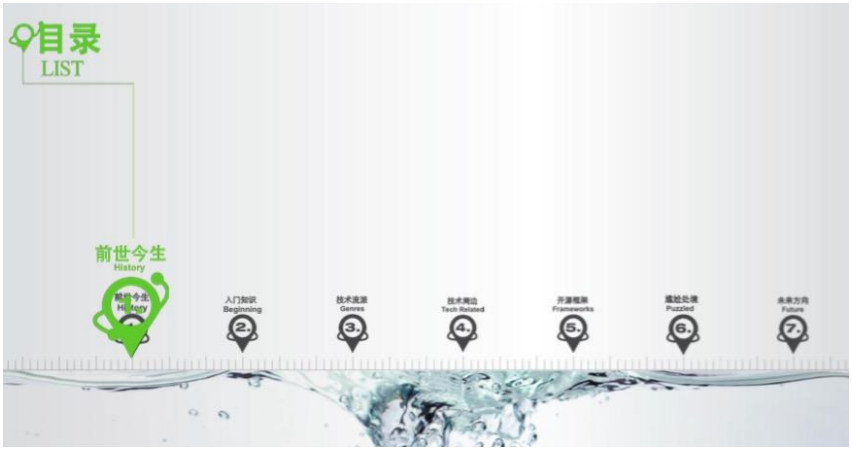
Android插件化：从入门到放弃

作者 包建强 发布于 2016年7月14日 | 首届应用性能管理大会 APMCon，聚焦国内外性能优化先进技术

引言

先简单介绍一下Android插件化。很早之前已经有公司在研究这项技术，淘宝做得比较早，但淘宝的这项技术一直是保密的。直到2015年才陆续出现很多框架，Android插件化分成很多技术流派，实现的方式都不太一样。我今天的主题就是，Android插件化的不同流派、不同思想，以及做插件化需要掌握哪些知识。

今天分享的题目是"从入门到放弃"。插件化技术不是45分钟演讲能说清楚的，我大致算了一下，要真能讲清楚这门技术起码需要六个小时，每个小时讲一个主题，包括Android底层系统与插件化相关的类，四大组件的原理与相应插件化实现方式、增量更新、AAPT等技术 。今天的分享浓缩成45分钟，希望大家可以获益。



分享内容分成七大部分。首先是前世今生，即插件化的历史。第二部分是入门知识，即与插件化有关的Android系统底层的实现原理。第三部分是技术流派，即目前Android插件化多种技术流派及其具体不同的实现方式。第四部分是周边相关的技术实现。第五部分是目前国内流行的开源框架，包括各个公司正在使用的框架，还有流传不是很广但意义也很重大的框架。第六，是针对Android插件化的一些问题的经验分享。最后，是Android插件化的未来—我们是否该放弃这门技术。

前世今生

Android插件化的历史，可能是本次演讲最有价值的内容。我梳理了三年前到现在Android插件化发展的一些规律。

首先，要记住2012年这个时间点。2012年的时候，就有人做插件化技术，是大众点评的屠毅敏，他推出了AndroidDynamicLoader框架，用Fragment来实现。大众点评是国内做App比较早的公司，他们积累了很多的经验，尤其是插件化技术 。通过动态加载不同的Fragement，把想换的页面都换掉。我们也是在这个项目中第一次看到了如何通过addAssetPath来读取插件中的资源。

2013年，出现了23Code。23Code提供了一个壳，在这个壳里可以动态下载插件，然后动态运行。可以在壳外编写各种各样的控件，放在这个框架下去运行。这就是Android插件化技术。这个项目的作者和开源地址，目前不是很清楚。

2014年初，大家也许看过一个视频，阿里一位员工做了一次技术分享，专门讲淘宝的Altas技术，以及这项技术的大方向。但是很多技术细节没有分享。

然后是任玉刚的里程碑式的项目。2014年底，玉刚发布了一个Android插件化项目，起名为dynamic-load-apk，这跟后续介绍的很多插件化项目都不太一样。它没有Hook太多的系统底层方法，而是从上层，即App应用层解决问题，创建一个继承自Activity的ProxyActivity类，然后让插件中的所有Activity都继承自ProxyActivity，并重写Activity所有的方法。之所以说这个项目是里程碑式的，是因为在2015年之前业界没有太多资料可以参考。之后曾和玉刚聊天，他十分感慨当年如何举步维艰地开发这个框架。我当时在途牛工作，使用了这个Android插件化框架。

2015年4月，一个新框架推出来，叫OpenAltas，后来改名为ACDD。这个框架参考了淘宝App的很多经验，主要就是Hook的思想，同时，还首次提出来通过扩展AAPT来解决插件与宿主的资源id冲突的问题。

2015年8月，张勇发布DroidPlugin。这是Android插件化中第二个里程碑式的项目，这个项目太牛了，能把任意的App都加载到宿主里。可以基于这个框架写一个宿主App，然后就可以把别人写的App都当作插件来加载。这个框架的功能的确很强大，但强大的代价就是要改写很多Android系统的底层代码，更别提这哥们还比较懒，没有制订任何说明文档，导致技术人员掌握这个框架不太容易。360的田维术曾编写了一系列文章，专门介绍这个框架，后面我会介绍。

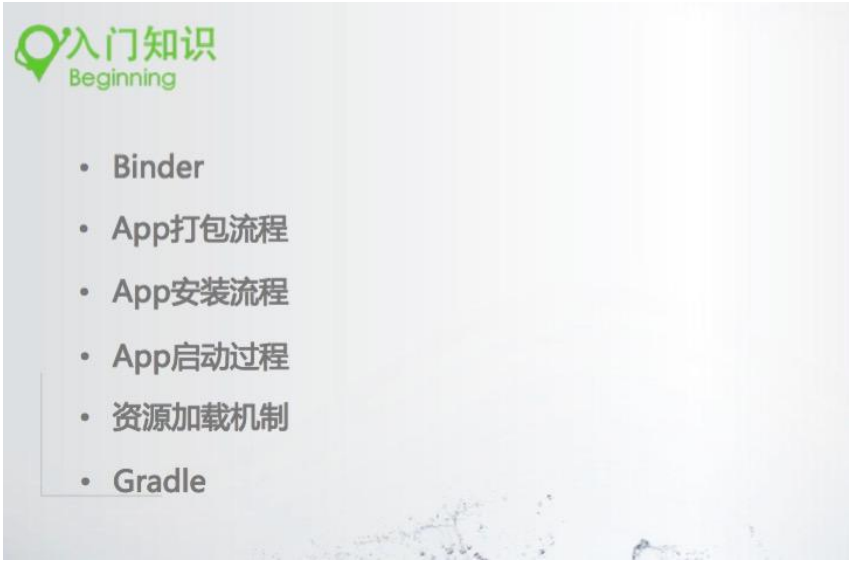
再之后就是百花齐放的时代了，GitHub上有很多插件化框架，但这些框架影响都不大，我们这里就略过了。

接下来登场的是热修复技术。2015年5月，iOS推出了JSPatch，JSPatch通过Runtime的机制，能迅速修复线上App任何一个类的任何一个方法。而当时的Android系统没有能迅速替换的方式。于是，在2015年9月，有人找到了实现迅速替换的途径，就是Andfix，后面会讲它的原理。

2015年10月，大众点评的贾吉鑫做了一个项目，起名为Nuwa（女娲），主要思路跟Andfix差不多，都是解决Android的修复问题，能修复线上的任何一个方法。可惜后来没有继续维护。

最后，2015年底，仍然是Android插件化框架，福建的林广亮提出了一个新机制——Small框架，这个机制不太一样的地方就是，通过脚本的方式来解决资源冲突的问题。

入门知识



介绍完Android插件化的历史，接下来讲一讲Android插件化需要的Android系统底层知识。在座的基本都是做Android开发出身，或许有一半到三分之一是资深的，还有的只做了一两年，希望对插件化有更深的认识。要想完全明白插件化技术，首先需要了解Android系统的底层实现。

首先，做Android系统原代码的人应该非常熟悉Binder，如果没有它真的寸步难行。Binder涉及两层技术。你可以认为它是一个中介者模式，在客户端和服务器端之间，Binder就起到中介的作用，这是我这段时间对Binder的思考。要实现四大组件的插件化，就需要在Binder上做修改。Binder服务端的内容没办法修改，只能改客户端的代码。四大组件每个组件的客户端都不太一样，这个需要大家自己去发现，时间关系，这里就不多说了。

学习Binder的最好方式就是AIDL。你可以读到很多关于AIDL的资料，通过制订一个aidl文件自动生成一个Java类，研究一下这个Java类的每个方法和变量，然后再反过来看四大组件，其实都是跟AIDL差不多的方式。

其次，是App打包的流程。代码写完了，执行一次打包操作，中途经历了资源打包、dex生成、签名等过程。其中最重要的就是资源的打包，即AAPT这一步，如果宿主和插件的资源id冲突，一种解决办法就是在这里做修改。

第三，App在手机上的安装流程也很重要。熟悉安装流程不仅对插件化有帮助，在遇到安装bug的时候也非常重要。手机安装App的时候，经常会有下载异常，提示资源包不能解析，这时需要知道安装App的这段代码在什么地方，这只是第一步。第二步需要知道，App下载到本地后，具体要做哪些事情。手机有些目录不能访问，App下载到本地之后，放到哪个目录下，然后会生成哪些文件。插件化有个增量更新的概念，如何下载一个增量包，从本地具体哪个位置取出一个包，这个包的具体命名规则是什么，等等。这些细节都必须要清楚明白。

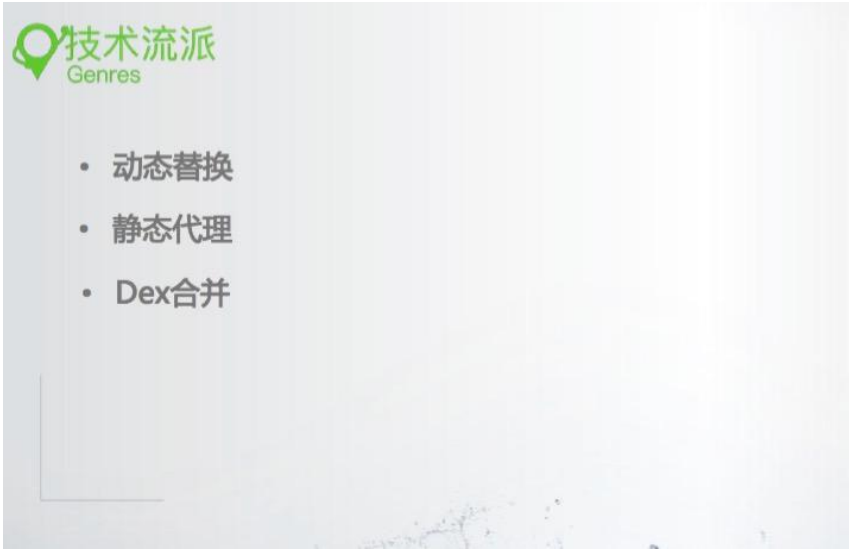
第四，是App的启动流程。Activity启动有几种方式？一种是写一个startActivity，第二种是点击手机App，通过手机系统里的Launcher机制，启动App里默认的Activity。通常，App开发人员喜闻乐见的方式是第二种。那么第一种方式的启动原理是什么呢？另外，启动的时候，main函数在哪里？这个main函数的位置很重要，我们可以对它所在的类做修改，从而实现插件化。

第五点更重要，做Android插件化需要控制两个地方。首先是插件Dex的加载，如何把插件Dex中的类加载到内存？另外是资源加载的问题。插件可能是apk也可能是so格式，不管哪一种，都不会生成R.id，从而没办法使用。这个问题有好几种解决方案。一种是重写Context的getAsset、getResource之类的方法，偷换概念，让插件读取插件里的资源，但缺点就是宿主和插件的资源id会冲突，需要重写AAPT。另一种是重写AMS中保存的插件列表，从而让宿主和插件分别去加载各自的资源而不会冲突。第三种方法，就是打包后，执行一个脚本，修改生成包中资源id。

第六点，在实施插件化后，如何解决不同插件的开发人员的工作区问题。比如，插件1和插件2，需要分别下载哪些代码，如何独立运行？就像机票和火车票，如何只运行自己的插件，而不运行别人的插件？这是协同工作的问题。火车票和机票，这两个Android团队的各自工作区是不一样的，这时候就要用到Gradle脚本了，每个项目分别有各自的仓库，有各自不同的打包脚本，只需要把自己的插件跟宿主项目一起打包运行起来，而不用引入其他插件，还有更厉害的是，也可以把自己的插件当作一个App来打包并运行。

上面介绍了插件化的入门知识，一共六点，每一点都需要花大量时间去理解。否则，在面对插件化项目的时候，很多地方你会一头雾水。而只要理解了这六点核心，一切可迎刃而解。

技术流派



接下来是技术流派。技术流派目前分三种。

第一种是动态替换，也就是Hook。可以在不同层次进行Hook，从而动态替换也细分为若干小流派。可以直接在Activity里做Hook，重写getAsset的几个方法，从而使用自己的ResourceManager和AssetPath；也可以在更抽象的层面，也就是在startActivity方法的位置做Hook，涉及的类包括ActivityThread、Instrumentation等；最高层次则是在AMS上做修改，也就是张勇的解决方案，这里需要修改的类非常多，AMS、PMS等都需要改动。总之，在越抽象的层次上做Hook，需要做的改动就越大，但好处就是更加灵活了。没有哪一个方法更好，一切看你自己的选择。

第二种是静态代理，这是任玉刚的框架采取的思路。写一个PluginActivity继承自Activity基类，把Activity基类里面涉及生命周期的方法全都重写一遍，插件中的Activity是没有生命周期的，所以要让插件中的Activity都继承自PluginActivity，这样就有生命周期了。

第三种是Dex合并，Dex合并就是Android热修复的思想。刚才说到了两个项目——AndFix和Nuwa，它们的思想是相同的。原生Apk自带的Dex是通过PathClassLoader来加载的，而插件Dex则是通过DexClassLoader来加载的。但有一个顺序问题，是由Davlik的机制决定的，如果宿主Dex和插件Dex都有一个相同命名空间的类的方法，那么先加载哪个Dex，哪个Dex中的这个类的方法将会占山为王，后面其他同名方法都替换了。所以，AndFix热修复就是优先加载插件包中的Dex，从而实现热修复。由于热修复的插件包通常只包括一个类的方法，体量很小，和正常的插件不是一个数量级的，所以只称为热修复补丁包，而不是插件。

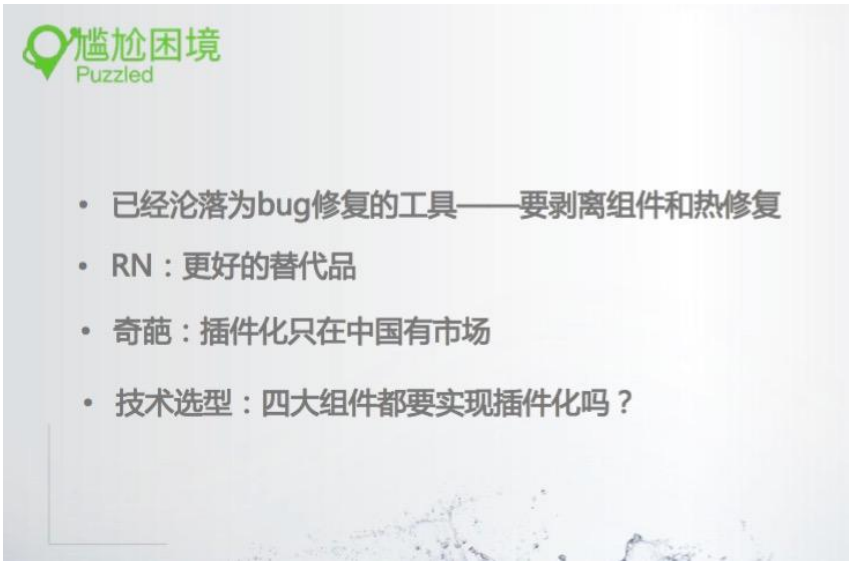
技术周边

关于技术周边的内容有三个部分。首先是AAPT，资源冲突，就是说默认App应用，插件里的资源和数据资源冲突，如果不引入这个资源，相安无事。很多时候就算有冲突也无所谓，问题就出在插件引用资源的时候有冲突了，无法解决，怎么办？那就要立刻改写App。有一个关于打包的App，可以加当前的前缀，改成你想要的。比如，火车票和酒店分别取名，这样就可以指定前缀、打包，插进一个模块，资源的前缀都不一样。小米也承认，会占用0x11这个前缀。这是需要关注的一个点。

第二是增量更新。360目前最牛逼的地方是，把所有数据跟之前一个版本差，产生增量的数据。他们当然也更新了插件化。360的刘存栋做了一个增量更新的框架。可以在后台服务器把两个版本的Android App做拆分，然后把增量包下载到本地，再跟本地进行合并，提供一个STK，再合在一起，这就是增量更新。

第三是插件管理平台，要管理每个版本的差异、每个插件最低数据的版本号。

尴尬困境



接下来是最近两年出现的一些尴尬困境。

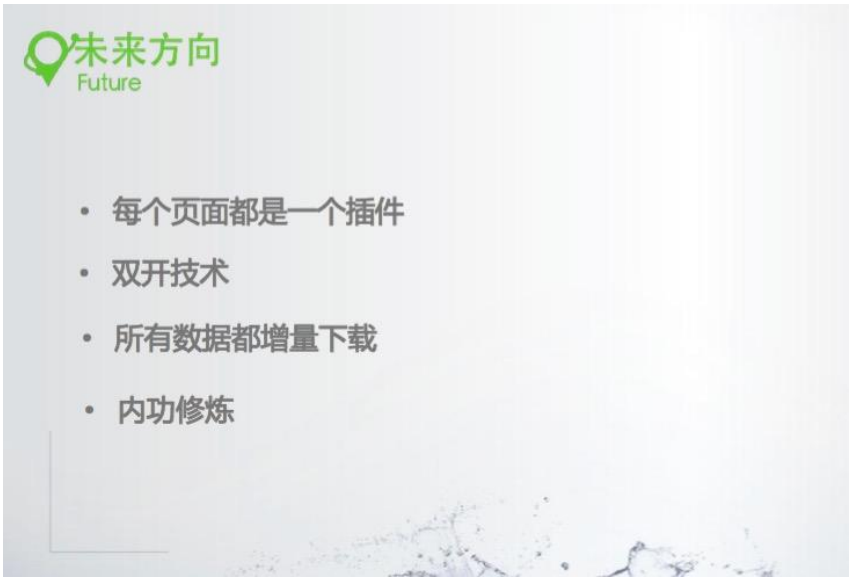
首先，插件化已经沦落为修bug的工具。这跟插件化的初衷不一样，插件化是实现新功能，而不是修复bug。

其次，插件化现在有一个更好的替代品—RN。接下来，RN会是真正实现动态化的最佳方式，至少我是这么认为的。

第三，插件化技术只在中国有市场。国外的公司根本不看好这项技术，这可能是因为他们用GooglePlay，而谷歌官方不建议用插件化这种方式。国外开发者不敢越雷池半步。

第四，四大组件都需要做插件化吗？根据我自己的经验，做一款电商或旅游类的App，有一两百个Activity，Service用得很少，ContentProvider和BroadcastReceiver基本不用。所以，这种App实现Activity和Service的插件化就够了。像手机助手这样的App，非常频繁使用四大组件，所以四大组件都必须实现插件化，这也是张勇当年在360开发出DroidPlugin支持四大组件的原因。

未来方向



最后讲一下Android插件化未来的方向。阿里一位技术专家冯森林曾说，插件化最厉害的发展方向应该是每个Activity都是一个插件。这个观点在插件化技术交流群里一提出来之后，群里所有人都沉默良久。仔细想想，插件化的未来好像的确是这个发展方向，这样就可以将任何一个出问题的Activity迅速替换。但当RN一经提出，这个观点就慢慢消失了，RN比插件化更轻量级，越来越多人选择了RN。

其次，就是双开技术。双开技术是现在非常火的一项技术。如果想实现这种机制，一定要实现上面讲的插件化所涉及的内容。宁波一位高中生Lody，他从初三就开始研究这门技术，将很多不错的双开项目放在GitHub上。

第三，刚才讲的所有数据都增量下载的机制，大家都可以实施一下，虽然做起来很麻烦，但是一旦实现，会让你的App非常快。比如，你每次进入都需要刷新城市列表吗？大约几百KB，即使你开gzip，刷新速度仍然很慢，这时候增量更新就是一个很好的方式。

最后是内功的修炼。通读一遍上面列出来的基础知识，然后再去做App应用，你会清楚知道静态广播、动态广播具体什么时候用，什么情况下可能出bug。AIDL可能会很少用，但真正做设计和实现的时候，这个基本功就非常重要了。所以，插件化只是一门技术，你最应该关注的是其背后的本质，也就是内功修炼。