Android 突破64K方法数的限制

随着安卓平台的不断发展与壮大，市场上大而全的应用比比皆是，产品需求的变更累积和UI交互的极致追求，除了 resources 文件的俱增，在 Android Project 中依赖的 Library 和 自己写的 Java 代码也会越来越多。这些变化，除了会导致打包出的 APK 文件越来越大之外，当项目中java代码包含的方法数（method count）超出一个峰值时，编译过程中就会出现如下错误：

较早版本的编译系统中，错误内容如下：

|  |  |
| --- | --- |
| 1 2 | Conversion to Dalvik format failed: Unable to execute dex: method ID not in [0, 0xffff]: 65536 |

较新版本的编译系统中，错误内容如下：

|  |  |
| --- | --- |
| 1 2 3 | trouble writing output: Too many field references: 131000; max is 65536. You may try using --multi-dex option. |

尽管在不同版本的编译系统中显示的错误内容不尽相同，但都提到了一个具体的数字：65536，也就是本文要讲到的核心内容，Android 64K Method Counts Limit 的峰值。详细信息，参考官网用户指南：[Configure Apps with Over 64K Methods](https://developer.android.com/studio/build/multidex.html)

**Android 64K Method Counts Limit**

Android Project 经过编译打包，其中的Java代码（包括Library）转化为DEX格式的字节码文件，这是Android 5.0之前的Dalvik虚拟机决定的（5.0之后改为ART虚拟机），并且采用short类型引用DEX文件中的method，这也为method数量的峰值大小埋下了隐患。short类型能够表示的最大值是65536，也就说单个DEX文件中最多只有65536个method能够得到引用，如果代码执行了超出部分的method引用，自然会报错，如methodNotFound等。1K等于1024，65536刚好是64K，为了便于称呼和使用，就将这个限制统称为64K方法数的引用限制。

为了解决64K方法数限制的问题，我们可以在项目中使用multidex配置，当项目中的方法数（包括：Android framework，library和我们自己写的代码）超过64K时，编译系统会自动编译出多个DEX文件。

**Multidex Support**

Android 5.0之前，安卓系统采用的是Dalvik虚拟机，采用的是JIT技术（Just-in-time compilation，即时编译，运行时编译DEX字节码文件，这也是以前为什么安卓手机用户总是诟病Android系统比iOS系统运行卡顿的原因），限制每个APK文件只能包含一个DEX文件（即classes.dex）。为了绕开这个限制，Google给我们提供了[multidex support library](https://developer.android.com/tools/support-library/features.html#multidex)兼容包，帮助我们实现应用程序加载多个DEX文件，并且这个兼容包作为程序的主DEX文件，管理者其他DEX文件的访问。

注意：由于[Instant Run](https://developer.android.com/tools/building/building-studio.html#instant-run)机制利用的就是multidex原理，当项目中minSdkVersion参数设置为20或者更小，并且运行在Android 4.4 (API level 20)或更低版本的设备中时，Instant Run将失效。

Android 5.0之后，安卓系统改用了ART虚拟机（Android RunTime），采用的是OAT技术（Ahead-of-time，预编译，在应用安装的时候扫描应用中的所有DEX文件，并编译成一个.oat格式的文件供安卓设备执行，所以相比Dalvik虚拟机下的应用，安装时间较长）。因此可以理解为，使用ART虚拟机下的安卓系统自动支持APK文件中多个DEX的加载。

注意：使用Instant Run时，如果项目中的minSdkVersion参数设为21或更高版本，Android Studio编译运行时会自动使应用支持multidex。但Instant Run仅仅作用于debug版本，我们依然需要给release版本配置multidex来避开64K方法数的限制。

**Config for Multidex With Gradle**

Android Gradle 插件在 Android SDK Build Tools 21.1 及更高版本的编译工具上支持multidex作为编译配置的一部分，所以确保我们的Android SDK Build Tools tools已经更新至21.1或更高版本，然后再来配置应用的multidex部分。

第一步，修改app/build.grale文件，使项目能够使用multidex：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | android {   compileSdkVersion 21  buildToolsVersion "21.1.0"   defaultConfig {  ...   // Enabling multidex support.  multiDexEnabled true  }  ... }  dependencies {  compile 'com.android.support:multidex:1.0.0' } |

第二步，修改AndroidManifest.xml文件，引用MultiDexApplication类：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.yifeng.mdstudysamples">  <application  ...  android:name="android.support.multidex.MultiDexApplication">  ...  </application> </manifest> |

添加这些配置后，编译工具会构建出一个主DEX文件（classes.dex）和其他DEX文件（classes2.dex，classes3.dex等，如果需要的话），编译系统会将他们打包到APK文件中。

注意：一般我们会在项目中自定义一个继承自Application的类，此时就需要重写attachBaseContext()方法，并在该方法里面调用MultiDex.install(this)来支持multidex，可参考：[MultiDexApplication](https://developer.android.com/reference/android/support/multidex/MultiDexApplication.html)

**Optimizing Multidex Development Builds**

multidex配置下的应用，编译系统需要经过复杂的DEX分割运算，导致增加项目的编译时间，从而影响开发人员的开发效率。我们可以使用[productFlavors](http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Product-flavors)构建开发环境和正式环境的不同flavors来优化multidex的长时间编译问题。

对于development flavor，设置minSdkVersion值为21，运行在Android 5.0以上版本的设备中，使用ART-supported格式生成multidex的速度要快得多。对于release flavor，minSdkVersion值则设为应用实际支持的版本，编译系统耗费较长的时间来生成适配多设备的multidex APK文件。如：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 | android {  productFlavors {  // Define separate dev and prod product flavors.  dev {  // dev utilizes minSDKVersion = 21 to allow the Android gradle plugin  // to pre-dex each module and produce an APK that can be tested on  // Android Lollipop without time consuming dex merging processes.  minSdkVersion 21  }  prod {  // The actual minSdkVersion for the application.  minSdkVersion 14  }  }  ...  buildTypes {  release {  runProguard true  proguardFiles getDefaultProguardFile('proguard-android.txt'),  'proguard-rules.pro'  }  } } dependencies {  compile 'com.android.support:multidex:1.0.0' } |

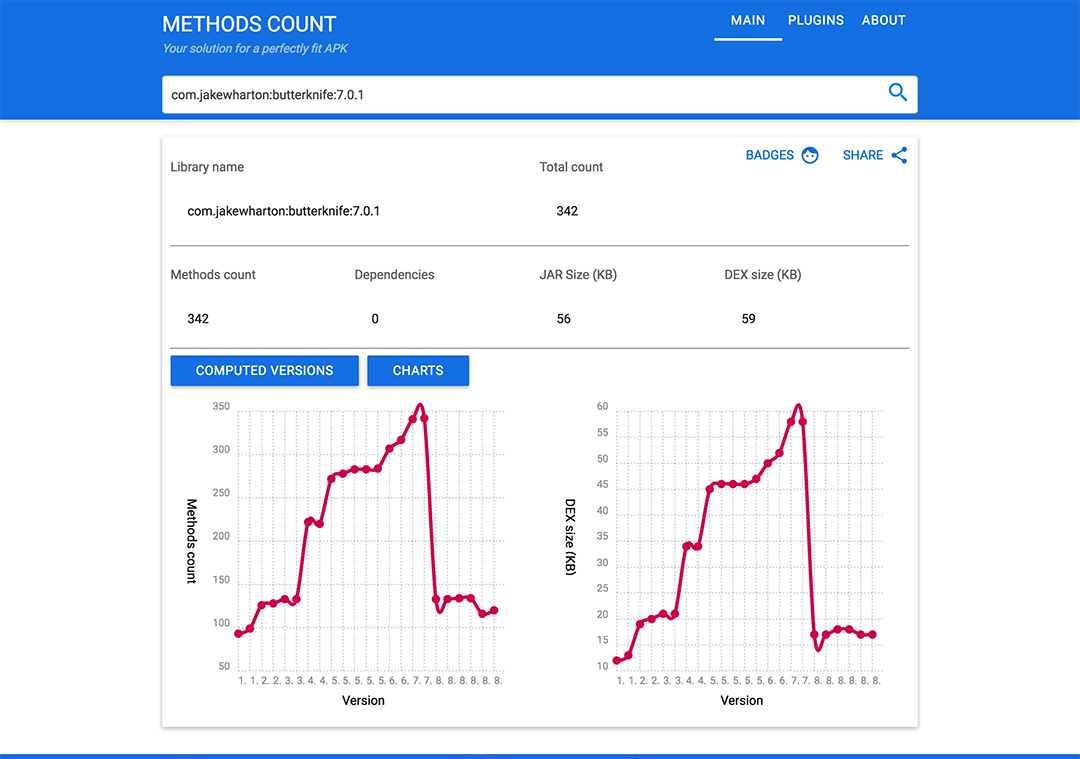
这样，在开发阶段，使用devDebug类型的变种app，取消混淆，支持multidex，并且运行在5.0及以上版本的设备中，能够加快编译过程。有关flavors的信息，以前写过一篇文章：[Android 利用Gradle实现app的环境分离](http://yifeng.studio/2016/09/06/apk-environment-separate/)，更多信息可以参考英文手册：[Gradle Plugin User Guide](http://tools.android.com/tech-docs/new-build-system/user-guide)，对应中文版译文：[Gradle Android插件用户指南翻译](http://avatarqing.github.io/Gradle-Plugin-User-Guide-Chinese-Verision/)。

**Methods Count Statistics**

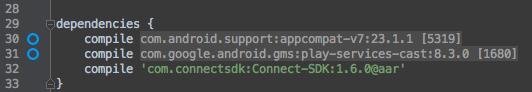
尽管安卓系统支持multidex，我们还是要学会分析我们的应用，查看各个部分的方法数，减少冗余方法。这里推荐几个工具，帮助我们分析。

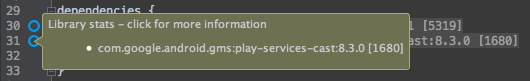
[Library Methods count](http://www.methodscount.com/)

一个在线统计 Android Library 方法数的网站，能够统计出Android领域常见libraries的方法数、JAR文件和DEX文件大小，并且能够选择不同版本，以图表的形式展示出来。

[](http://ocq7gtgqu.bkt.clouddn.com/methodscount-samples-01.png)

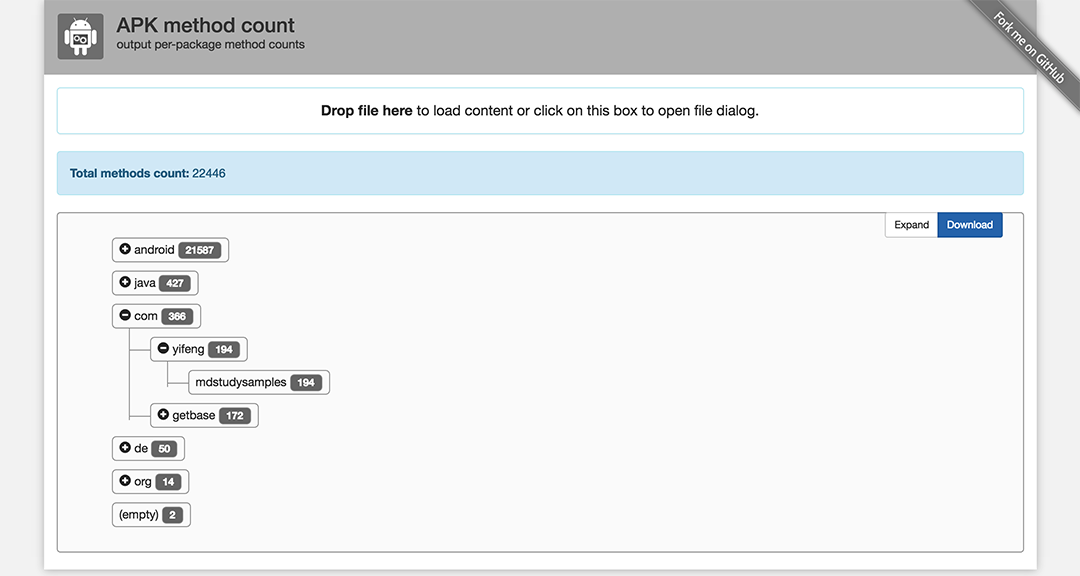
该网站也提供了Android Studio的插件，帮助我们分析项目中所依赖的libraries的方法数，如图所示：

[](http://ocq7gtgqu.bkt.clouddn.com/methodscount-samples-02.png)

[](http://ocq7gtgqu.bkt.clouddn.com/methodscount-samples-03.png)

[Apk Method Count](http://inloop.github.io/apk-method-count/)

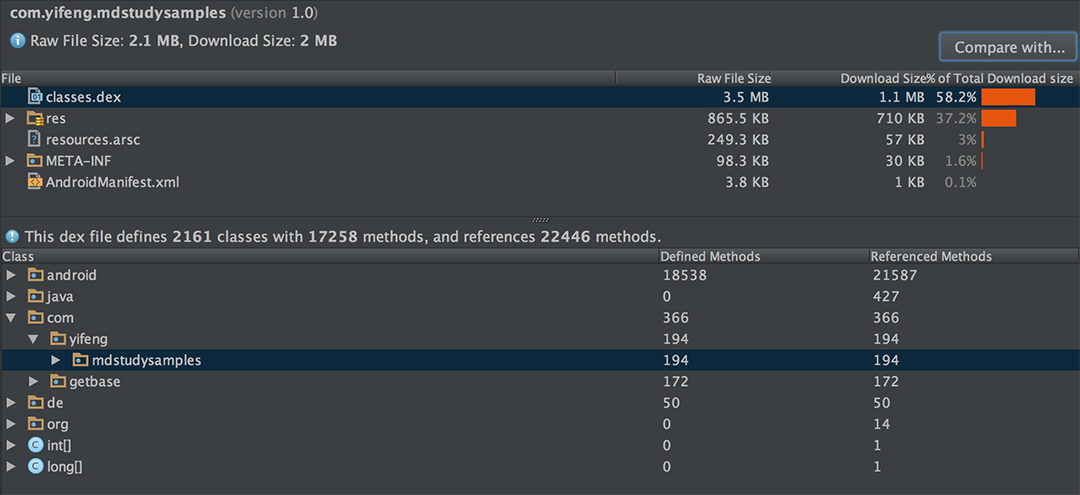
一个在线统计 APK 文件方法数的开源项目，只需要将需要分析的APK文件拖拽上传至此，即可得到分析结果，如图：

[](http://ocq7gtgqu.bkt.clouddn.com/apk-method-count-samples.png)

[Android Studio APK Analyzer](https://developer.android.com/studio/build/apk-analyzer.html)

最后，要重磅推荐Android Studio自带的APK Analyzer，功能齐全，使用方便，绝对是安卓开发人员分析应用的不二选择。使用 Android Studio APK Analyzer ，我们至少能够做到：

* 查看APK压缩文件中各个子文件的大小（如DEX和resource文件）
* 理解DEX文件的结构
* 快速查看APK文件的版本信息（直接查看AndroidManifest.xml内容）
* 直观地比较两个APK文件内容

[](http://ocq7gtgqu.bkt.clouddn.com/Android-Studio-APK-Analyzer-Samples.png)

开发阶段使用Android Studio打开一个项目时，有三种方式使用APK Analyzer工具：

* 直接拖拽APK文件到Android Studio的编辑窗口
* 双击打开项目目录app/build/outputs/apk/下的APK文件
* 点击菜单栏Build->Analyse APK...并选择APK文件