Android 65535的问题

一.概况

项目中由于平时使用了不少的三方库，同时随着App的不断迭代自身代码也在不断增加，这时候方法数超过了65536。带来的后果就是编译和打包失败。

二.原因

当方法数超过65536的时候，会报如下的错误。

The number of method references in a .dex file cannot exceed 64K.

Learn how to resolve this issue at https://developer.android.com/tools/building/multidex.html

com.android.dex.DexIndexOverflowException: method ID not in [0, 0xffff]: 65536

众所周知当Android app代码足够多时， 编译会报方法数超过65535问题。 Google为了解决这个问题，提供了multi-dex方案， 即将代码编译成若干个dex文件，如classes.dex, classes2.dex...classes\*.dex。

我们关心的是怎么解决这个问题，包括插件化、multi-dex等等方案，

随着Android平台持续增长，Android应用程序越来越大。在Android系统中，一个App的所有代码都在一个Dex文件里面。Dex是一个类似Jar的存储了多个Java编译字节码的归档文件。因为Android系统使用Dalvik虚拟机，所以需要把使用Java Compiler编译之后的class文件转换成Dalvik能够执行的class文件。这里需要强调的是，Dex和Jar一样是一个归档文件，里面仍然是Java代码对应的字节码文件。当Android系统启动一个应用的时候，有一步是对Dex进行优化，这个过程有一个专门的工具来处理，叫DexOpt。DexOpt的执行过程是在第一次加载Dex文件的时候执行的。这个过程会生成一个ODEX文件，即Optimised。DexOpt会把每一个类的方法id检索起来，存在一个链表结构里面。但是这个链表的长度是用一个short类型来保存的，导致了方法id的数目不能够超过65536个。而一般方法比属性和类要多，所以一般是方法包65535

/\*

\* Direct-mapped "header\_item" struct.

\*/

struct DexHeader {

u1 magic[8]; /\* includes version number \*/

u4 checksum; /\* adler32 checksum \*/

u1 signature[kSHA1DigestLen]; /\* SHA-1 hash \*/

u4 fileSize; /\* length of entire file \*/

u4 headerSize; /\* offset to start of next section \*/

u4 endianTag;

u4 linkSize;

u4 linkOff;

u4 mapOff;

u4 stringIdsSize;

u4 stringIdsOff;

u4 typeIdsSize;

u4 typeIdsOff;

u4 protoIdsSize;

u4 protoIdsOff;

u4 fieldIdsSize;

u4 fieldIdsOff;

u4 methodIdsSize;

u4 methodIdsOff;

u4 classDefsSize;

u4 classDefsOff;

u4 dataSize;

u4 dataOff; }

2.1关于64 k引用限制

Android应用程序（APK）在Dalvik可执行文件的形式包含可执行的字节码文件（DEX）文件，其中包含已编译的代码（运行你的应用程序）。Dex文件最多能包含65536个方法：包括Android框架方法、library方法的总数、和你自己的代码方法总数。因为65536等于64×1024，这一限制被称为“64k引用限制”。

这个极限就要求我们配置应用程序的构建过程，需要生成多个DEX文件，所以称为multidex 配置。

2.2不同版本下虚拟机

Android 5.0（API leve 21）之前的系统使用Dalvik虚拟机。默认情况下，Dalvik限制一个apk只有一个Dex文件。为了绕过这个限制，我们可以使用com.android.support:multidex也就是Android官方为我们提供的解决这个问题的一个库，它成为我们APK的主要Dex文件的一部分，负责管理我们APK访问其他Dex文件和代码。（注意: 如果咱的项目minSdkVersion是20或更低，运行到Android 4.4(API leve 20)或者更低版本的设备上时需要禁用AndroidStudio的即时运行）

Android 5.0（API leve 21）和更高的系统使用runtime是ART ，原生支持从应用的apk文件加载多个DEX文件。ART在安装应用时预编译应用程序，会扫描多个classes(..N).dex文件编译成一个.oat的文件。

由于这里只是为了阐明“64 k引用限制”的历史背景，所以关于两种虚拟机的知识点这里不做展开。

三.解决方法

尽量避免64k限制

具体建议：

1.选择三方依赖库的时候做好调研工作，选择满足自己功能的并且尽量小的。（建议能用源码的用源码，对于以后自己的定制和维护很有好处）

2.正式打包构建的时候，使用代码混淆器ProGuard混淆移除未使用的代码，也就是不把没有使用的代码打包到我们的apk中。

解决64k问题

app中build.gradle中添加multiDexEnabled true和compile 'com.android.support:multidex:1.0.1';含义：支持多个dex和依赖multidex库。

apply plugin: 'com.android.application'

android {

---省略代码---

compileSdkVersion 23

buildToolsVersion '26.0.2'

defaultConfig {

minSdkVersion 17

targetSdkVersion 22

versionCode 61

versionName "4.2.5"

// Enabling multidex support.

multiDexEnabled true

}

---省略代码---

}

dependencies {

---省略代码---

compile 'com.android.support:multidex:1.0.1'

继承android.support.multidex.MultiDexApplication类

分两种情况：

第一种有自己的Application

重写Applicatio中这个方法如下

@Override

protected void attachBaseContext(Context base) {

super.attachBaseContext(base);

MultiDex.install(this);

}

第二种没有自己的Application

如果我们的APP没有重写过Application类，我们直接继承MultiDexApplication，然后在manifest.xml中注册Application即可。

与此同时可能会报java.lang.OutOfMemoryError: Java heap space错误，那么解决方法是：在app的build.gradle中android中添加如下代码

//加大java堆内存大小

dexOptions {

javaMaxHeapSize "2g"//这里2g或者4g都可以

}

四.multidex库的一些限制因素

第二个Dex文件过大问题

1.Dex文件安装到设备的过程非常复杂，如果第二个Dex文件太大，可能导致应用无响应。此时应该使用ProGuard减小Dex文件的大小。

系统低版本问题

1.由于Dalvik linearAlloc的Bug，应用可能无法在Android 4.0之前的版本启动，如果你的应用要支持这些版本就要多执行测试。

2.同样因为Dalvik linearAlloc的限制，如果请求大量内存可能导致崩溃。Dalvik linearAlloc是一个固定大小的缓冲区。在应用的安装过程中，系统会运行一个名为dexopt的程序为该应用在当前机型中运行做准备。dexopt使用LinearAlloc来存储应用的方法信息。Android 2.2和2.3的缓冲区只有5MB，Android 4.x提高到了8MB或16MB。当方法数量过多导致超出缓冲区大小时，会造成dexopt崩溃。

3.Multidex构建工具还不支持指定哪些类必须包含在首个Dex文件中，因此可能会导致某些类库（例如某个类库需要从原生代码访问Java代码）无法使用。

总之就是在做低版本兼容的时候可能会崩溃，要多测试。（不过现在基本面向的用户使用的都是4.4以上的系统，这些情况出现的几率大大降低）。

其他的补充

随着应用不断迭代更新，业务线的扩展，应用越来越大（比如：集成了各种第三方SDK或者公共开源的Library文件、jar文件）这样一来，项目耦合性就很高，重复作用的类就越来越多了，SO：问题就来了。  
相信大家在做自己公司项目时，都有机会遇到下面的错误：65535

那么让我们看一下为什么会引起这种错误：

在Android系统中，一个APP的所有代码都在一个Dex文件里面。Dex是一个类似Jar的存储了多个Java编译字节码的归档文件。因为Android系统使用Dalvik虚拟机，所以需要把使用Java Compiler编译之后的class文件转换成Dalvik能够执行的class文件。这里需要强调的是，Dex和Jar一样是一个归档文件，里面仍然是Java代码对应的字节码文件。当Android系统启动一个应用的时候，有一步是对Dex进行优化，这个过程有一个专门的工具来处理，叫DexOpt。DexOpt的执行过程是在第一次加载Dex文件的时候执行的。这个过程会生成一个ODEX文件，即Optimised Dex。执行ODex的效率会比直接执行Dex文件的效率要高很多。但是在早期的Android系统中，DexOpt有一个问题，也就是这篇文章想要说明并解决的问题。DexOpt会把每一个类的方法id检索起来，存在一个链表结构里面。但是这个链表的长度是用一个short类型来保存的，导致了方法id的数目不能够超过65536个。当一个项目足够大的时候，显然这个方法数的上限是不够的。尽管在新版本的Android系统中，DexOpt修复了这个问题，但是我们仍然需要对低版本的Android系统做兼容.

目前比较常用的方法：

\*\*(1) 应用插件化,比如使用我正在参与开发的插件化框架 :\*\* [https://github.com/singwhatiwanna/dynamic-load-apk](https://github.com/singwhatiwanna/dynamic-load-apk) ,如果有建议或者相关的问题,欢迎到\*\*Github\*\*上积极参与.

\*\*(2) 分割Dex,多工程: 把所需要的.class文件或者是Jar文件和一些源码一起编译生成一个Jar文件。然后使用Android SDK提供的dx工具把Jar文件转成Dex文件。我们可以提前对它进行ODex操作，让它在被DexClassLoader加载的时候，跳过DexOpt的部分工作，从而加快加载的过程.\*\*(可参考facebook:[https://www.facebook.com/notes/facebook-engineering/under-the-hood-dalvik-patch-for-facebook-for-android/10151345597798920](https://www.facebook.com/notes/facebook-engineering/under-the-hood-dalvik-patch-for-facebook-for-android/10151345597798920),这里边还可以看到在2.3上动态改变LinearAlloc缓冲的解决思路) 这两种方法并不冲突,插件化除了解决应用爆棚，还有很多其他的优点，可以看我之前的文章,不再复述.

\*\*当然,Google看来也意识到了目前应用方法数爆棚的问题, 目前在已经在API 21中提供了通用的解决方案，那就是android-support-multidex.jar. 这个jar包最低可以支持到API 4的版本(Android L及以上版本会默认支持mutidex).

让我们看一下如何应用android-support-multidex.jar(以下都以在Anroid studio中的使用为例,使用eclipse开发需要安装gradle插件，其他基本上相同):\*\*

\*\*首先在app/build.gradle文件中配置，如下：\*\*

android {  
compileSdkVersion 24  
buildToolsVersion "24.0.1"

defaultConfig {

...

minSdkVersion 14

targetSdkVersion 24

...

// 设置支持multidex

multiDexEnabled true

}

...

}

dependencies {  
compile 'com.android.support:multidex:1.0.1'  
}```  
**接下来的配置，有两种配置方式：**  
(1) : 在application的子类中配置（如果你需要有Application子类的情况）

package com.example;

import android.app.Application;

import android.content.Context;

/\*\*

\*

\*/

public class MyApplication extends Application {

@Override

protected void attachBaseContext(Context base) {

super.attachBaseContext(base);

MultiDex.install(this);

}

}```

(2) : 如果你不需要application的子类的情况，那就直接集成MultiDexApplication

package com.example;

import android.support.multidex.MultiDexApplication;  
import android.content.Context;

/\*\*

* Extended MultiDexApplication  
  \*/  
  public class MyApplication extends MultiDexApplication {

// 不需要重写attachBaseContext()

//..........  
}```  
好了，经过上面两步的解决，Android 65535的问题，也得到了解决。

**使用MutiDex的注意事项：**  
一、如果你继承了MutiDexApplication或者覆写了Application中的attachBaseContext()方法。  
Application类中的逻辑的注意事项：  
Application中的静态全局变量会比MutiDex的instal()方法优先加载，所以建议避免在Application类中使用静态变量引用main classes.dex文件以外dex文件中的类，可以根据如下所示的方式进行修改：

@Override

public void onCreate() {

super.onCreate();

final Context mContext = this;

new Runnable() {

@Override

public void run() {

// put your logic here!

// use the mContext instead of this here

}

}.run();

} ```

二. 虽然Google解决了应用总方法数限制的问题，但并不意味着开发者可以任意扩大项目规模。Multidex仍有一些限制：

\* DEX文件安装到设备的过程非常复杂，如果第二个DEX文件太大，可能导致应用无响应。此时应该使用ProGuard减小DEX文件的大小。

\* 由于Dalvik linearAlloc的Bug，应用可能无法在Android 4.0之前的版本启动，如果你的应用要支持这些版本就要多执行测试。

\* 同样因为Dalvik linearAlloc的限制，如果请求大量内存可能导致崩溃。Dalvik linearAlloc是一个固定大小的缓冲区。在应用的安装过程中，系统会运行一个名为dexopt的程序为该应用在当前机型中运行做准备。dexopt使用LinearAlloc来存储应用的方法信息。Android 2.2和2.3的缓冲区只有5MB，Android 4.x提高到了8MB或16MB。当方法数量过多导致超出缓冲区大小时，会造成dexopt崩溃。

\* Multidex构建工具还不支持指定哪些类必须包含在首个DEX文件中，因此可能会导致某些类库（例如某个类库需要从原生代码访问Java代码）无法使用。

避免应用过大、方法过多仍然是Android开发者要注意的问题。Mihai Parparita的开源项目dex-method-counts可以用于统计APK中每个包的方法数量。

通常开发者自己的代码很难达到这样的方法数量限制，但随着第三方类库的加入，方法数就会迅速膨胀。因此选择合适的类库对Android开发者来说尤为重要。

开发者应该避免使用Google Guava这样的类库，它包含了13000多个方法。尽量使用专为移动应用设计的Lite/Android版本类库，或者使用小类库替换大类库，例如用Google-gson替换Jackson JSON。而对于Google Protocol Buffers这样的数据交换格式，其标准实现会自动生成大量的方法。采用Square Wire的实现则可以很好地解决此问题。

\*\*常见问题\*\*

问题1：DexException: Library dex files are not supported in multi-dex mode,你可能会见到如下的错误:

Error:Execution failed for task ':app:dexDebug'.

com.android.ide.common.internal.LoggedErrorException: Failed to run command:  
$ANDROID\_SDK/build-tools/android-4.4W/dx --dex --num-threads=4 --multi-dex  
...  
Error Code:  
2  
Output:  
UNEXPECTED TOP-LEVEL EXCEPTION:  
com.android.dex.DexException: Library dex files are not supported in multi-dex mode  
at com.android.dx.command.dexer.Main.runMultiDex(Main.java:322)  
at com.android.dx.command.dexer.Main.run(Main.java:228)  
at com.android.dx.command.dexer.Main.main(Main.java:199)  
at com.android.dx.command.Main.main(Main.java:103) ```

解决方法：对于dex 的--multi-dex 选项设置与预编译的library工程有冲突,因此如果你的应用中包含引用的lirary工程,需要将预编译设置为false:

android {

// ...

dexOptions {

preDexLibraries = false

}

} ```

问题2：OutOfMemoryError: Java heap space.当运行时如果看到如下错误:

UNEXPECTED TOP-LEVEL ERROR:  
java.lang.OutOfMemoryError: Java heap space ```

解决方法：在dexOptions中有一个字段用来增加java堆内存大小:

android {

// ...

dexOptions {

javaMaxHeapSize "2g"

}

}```

参考相关资料:

\*\*1. MutiDex 官方文档\*\*: [https://developer.android.com/reference/android/support/multidex/MultiDex.html](https://developer.android.com/reference/android/support/multidex/MultiDex.html)

\*\*2.\*\* [http://blog.osom.info/2014/10/multi-dex-to-rescue-from-infamous-65536.html](http://blog.osom.info/2014/10/multi-dex-to-rescue-from-infamous-65536.html)

另附android -support-mutidex.jar下载地址: [http://download.csdn.net/detail/t12x3456/814338

作者：wangling90  
链接：https://www.jianshu.com/p/d714cf9a9b54  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

版权声明：本文为博主原创文章，转载请附上博文链接！