如果App引用的库太多，方法数超过65536后无法编译。这是因为单个dex里面不能有超过65536个方法。为什么有最大的限制呢， 因为android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， short占两个字节（保存-2的15次方到2的15次方-1，即-32768~32767），最大保存的数量就是65536。新版本的Android系统中修复了这个问题， 但是我们仍然需要对低版本的Android系统做兼容.

解决方法有如下几个： 1.精简方法数量,删除没用到的类、方法、第三方库。 2.使用ProGuard去掉一些未使用的代码 3.复杂模块采用jni的方式实现，也可以对边缘模块采用本地插件化的方式。 4.分割Dex

本文只介绍最后一种方法。

multidex方案配置

dex文件拆成两个或多个，为此谷歌官方推出了multidex兼容包，配合AndroidStudio实现了一个APK包含多个dex的功能。 Android 的 Gradle插件在 Android Build Tool 21.1开始就支持使用multidex了。

使用步骤

使用步骤包括： 1.修改Gradle的配置，支持multidex 2.修改你的manifest。让其支持multidexapplication类

注意其中第二步其实还有另外两种替代方法，下面介绍。

修改Gradle的配置，支持multidex:

```
android {
    compileSdkVersion 21
        buildToolsVersion "21.1.0"
        defaultConfig {
        ...
                minSdkVersion 14
                targetSdkVersion 21
                ...
                // Enabling multidex support.
multiDexEnabled true
    }
    ...
}
dependencies {
    compile 'com.android.support:multidex:1.0.0'
}
```

你可以在Gradle配置文件中的defaultConfig、 buildType、productFlavor中设置 multiDexEnabled true。

在manifest文件中，在application标签下添加MultidexApplication Class的引用，如下所示:

```
<?xml version="1.0" encoding="utf-8"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.multidex.myapplication">
    <application
        ...
       android:name="android.support.multidex.MultiDexApplication">
        ...    </application></manifest>
```

当然，如果你重写了Application，可以让自定义Applicationd继承android.support.multidex.MultiDexApplication。

如果之前已经继承了其他Application类，可以重写attachBaseContext()方法，并添加语句MultiDex.install(this);如下：

```
public class MyApplication extends BaseApplication{
    @Override
        protected void attachBaseContext(Context base) {
        super.attachBaseContext(base);
        MultiDex.install(this);
    }
}
```

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:

```
@Overridepublic 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();
}
```

Multidex的局限性

官方文档中提到了Multidex有局限性：

1. 如果第二个（或其他个）dex文件很大的话，安装.dex文件到data分区时可能会导致ANR（应用程序无响应）,此时应该使用ProGuard减小DEX文件的大小。
2. 由于Dalvik linearAlloc的bug的关系，使用了multidex的应用可能无法在Android 4.0 (API level 14)或之前版本的设备上运行。
3. 由于Dalvik linearAlloc的限制，使用了multidex的应用会请求非常大的内存分配，从而导致程序奔溃。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。
4. 在Dalvik运行时中，某些类的方法必须要放在主dex中，Android构建工具可能无法确保所有有此要求的类被编译进主dex中。

这些问题也非常值得我们关注.

一些在二级Dex加载之前,可能会被调用到的类(比如静态变量的类),需要放在主Dex中.否则会ClassNotFoundError. 通过修改Gradle,可以显式的把一些类放在Main Dex中.

```
afterEvaluate {
    tasks.matching {
        it.name.startsWith('dex')
    }
    .each {
        dx ->                    if (dx.additionalParameters == null) {
            dx.additionalParameters = []
                    }
                    dx.additionalParameters += '--multi-dex'
                    dx.additionalParameters += "--main-dex-list=
   $projectDir/<filename>".toString()
        }
    }
```

上面是修改后的Gradle,其中 是一个文本文件的文件名,存放在和这个Gradle脚本同一级的文件目录下. 而这个文本文件的内容如下.实际就是把需要放在Main Dex的类罗列出来.

```
android/support/multidex/BuildConfig/class
android/support/multidex/MultiDex$V14/class
android/support/multidex/MultiDex$V19/class
android/support/multidex/MultiDex$V4/class
android/support/multidex/MultiDex/class
android/support/multidex/MultiDexApplication/class
android/support/multidex/MultiDexExtractor$1/class
android/support/multidex/MultiDexExtractor/class
android/support/multidex/ZipUtil$CentralDirectory/class
android/support/multidex/ZipUtil/class
```

project.afterEvaluate标签在特定的project配置完成后运行，而gradle.projectsEvaluated在所有projects配置完成后运行。

如果用使用其他Lib,要保证这些Lib没有被preDex,否则可能会抛出下面的异常：

```
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:337)
     at com.android.dx.command.dexer.Main.run(Main.java:243)
     at com.android.dx.command.dexer.Main.main(Main.java:214)
     at com.android.dx.command.Main.main(Main.java:106)
```

遇到这个异常,需要在Gradle中修改,让它不要对Lib做preDexing

```
android {
    // ...                      dexOptions {
    preDexLibraries = false
            }
        }
```

如果每次都打开MultiDex编译版本的话,会比平常用更多的时间. Android的官方文档也给了我们一个小小的建议,利用Gradle建立两个Flavor.一个minSdkVersion设置成21,这是用了ART支持的Dex格式, 避免了MultiDex的开销.而另外一个Flavor就是原本支持的最小sdkVersion.平时开发时候调试程序,就用前者的Flavor，发布版本打包就用后者的Flavor.

```
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'
}
```

MultiDex实现原理

下面从DEX自动拆包和动态加载两方面来分析。

1、Dex 拆分

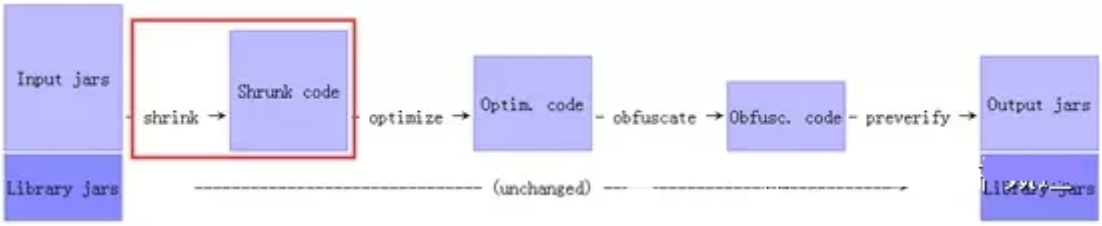
dex拆分步骤分为：

1. 自动扫描整个工程代码得到main-dex-list；
2. 根据main-dex-list对整个工程编译后的所有class进行拆分，将主、从dex的class文件分开；
3. 用dx工具对主、从dex的class文件分别打包成 .dex文件，并放在apk的合适目录。

怎么自动生成 main-dex-list？ Android SDK 从 build tools 21 开始提供了 mainDexClasses 脚本来生成主 dex 的文件列表。查看这个脚本的源码，可以看到它主要做了下面两件事情：

1. 调用 proguard 的 shrink 操作来生成一个临时 jar 包；
2. 将生成的临时 jar 包和输入的文件集合作为参数，然后调用 com.android.multidex.MainDexListBuilder 来生成主 dex 文件列表。

Proguard的官网执行步骤如下：



在 shrink 这一步，proguard 会根据 keep 规则保留需要的类和类成员，并丢弃不需要的类和类成员。也就是说，上面 shrink 步骤生成的临时 jar 包里面保留了符合 keep 规则的类，这些类是需要放在主 dex 中的入口类。

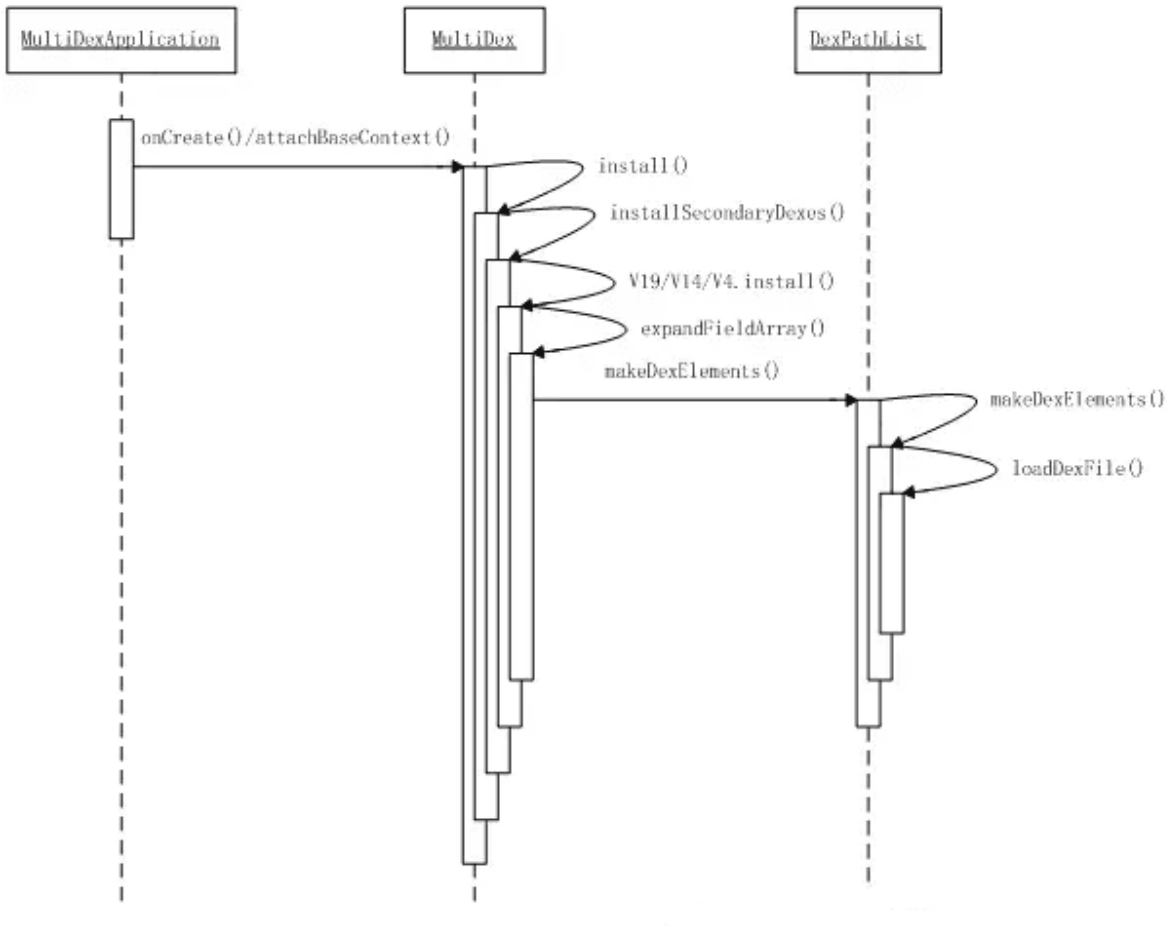
但是仅有这些入口类放在主 dex 还不够，还要找出入口类引用的其他类，不然仍然会在启动时出现 NoClassDefFoundError。而找出这些引用类，就是调用的 com.android.multidex.MainDexListBuilder，它的部分核心代码如下：

```
public MainDexListBuilder(String rootJar, String pathString) throws IOException {
    ZipFile jarOfRoots = null;
    Path path = null;
    try {
        try {
            jarOfRoots = new ZipFile(rootJar);
        } catch (IOException e) {
            throw new IOException("\"" + rootJar + "\" can not be read as a zip archive. ("
                    + e.getMessage() + ")", e);
        }
        path = new Path(pathString);
        ClassReferenceListBuilder mainListBuilder = new ClassReferenceListBuilder(path);
        mainListBuilder.addRoots(jarOfRoots);
        for (String className : mainListBuilder.getClassNames()) {
            filesToKeep.add(className + CLASS_EXTENSION);
        }
        keepAnnotated(path);
    } finally {
        try {
            jarOfRoots.close();
        } catch (IOException e) {
            // ignore
        }
        if (path != null) {
            for (ClassPathElement element : path.elements) {
                try {
                    element.close();
                } catch (IOException e) {
                    // keep going, lets do our best.
                }
            }
        }
    }
}
```

在调用 com.android.multidex.MainDexListBuilder 之后，符合 keep 规则的主 dex 文件列表就生成了。

2、Dex加载

因为Android系统在启动应用时只加载了主dex（Classes.dex），其他的 dex 需要我们在应用启动后进行动态加载安装。 Google 官方方案是如何加载的呢,Google官方支持Multidex 的 jar 包是 android-support-multidex.jar，该 jar 包从 build tools 21.1 开始支持。这个 jar 加载 apk 中的从 dex 流程如下：



此处主要的工作就是从 apk 中提取出所有的从 dex（classes2.dex，classes3.dex，...），然后通过反射依次安装加载从 dex 并合并 DexPathList 的 Element 数组。