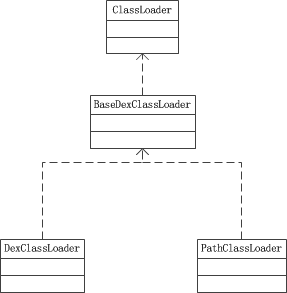
**Android动态热补丁修复技术介绍**

**一、背景**

当一个App发布之后，突然发现了一个严重Bug需要进行紧急修复，通常的做法都是重新打包App、测试、向各个应用市场和渠道换包、提示用户升级、用户下载、覆盖安装。有时候仅仅是为了修改了一行代码，也要付出巨大的成本进行换包和重新发布。这个时候就会想有没有办法以补丁的方式动态修复紧急Bug。答案肯定是有的，以下将介绍用基于Android dex分包方案来实现Android动态修复Bug的技术原理和实现。

**二、原理**

Android的ClassLoader体系，android中加载类一般使用的是PathClassLoader和DexClassLoader，如下图所示：



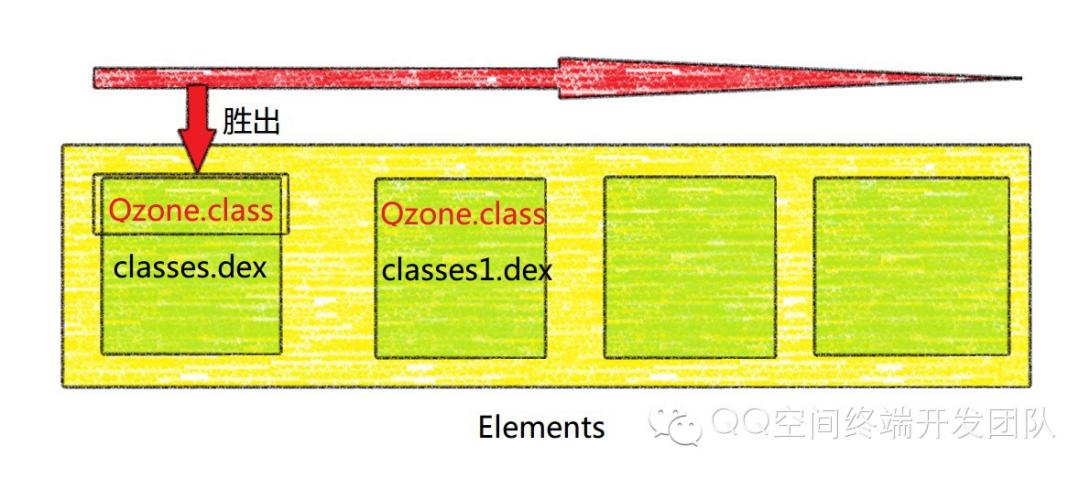
首先看下这两个类的区别：

PathClassLoader：Android是使用这个类作为其系统类和应用类的加载器。并且对于这个类呢，只能去加载已经安装到Android系统中的apk文件。

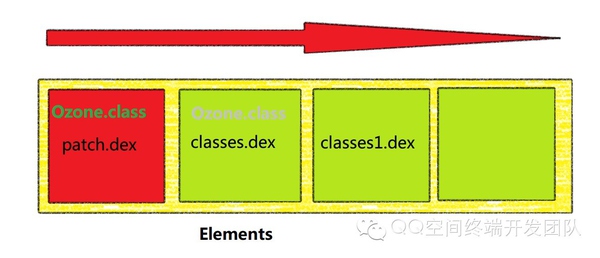
DexClassLoader：该类可以用来从.jar和.apk类型的文件内部加载classes.dex文件。可以用来执行非安装的程序代码。

到这里，只需要明白，Android使用PathClassLoader作为其类加载器，DexClassLoader可以从.jar和.apk类型的文件内部加载classes.dex文件就好了。

一个ClassLoader可以包含多个dex文件，每个dex文件是一个Element，多个dex文件排列成一个有序的数组dexElements，当找类的时候，会按顺序遍历dex文件，然后从当前遍历的dex文件中找类，如果找类则返回，如果找不到从下一个dex文件继续查找。理论上，如果在不同的dex中有相同的类存在，那么会优先选择排在前面的dex文件的类，如下图：



在此基础上，把有问题的把有问题的类打包到一个dex（patch.dex）中去，然后把这个dex插入到Elements的最前面，如下图：



比如，在这个数组的第一个元素放置我们的patch.jar（补丁包），里面包含修复过的类，这样的话，当遍历findClass的时候，我们修复的类就会被查找到，从而替代有bug的类。

至此，貌似热修复原理很简单。事实上，还存在一个CLASS\_ISPREVERIFIED的问题，那就是：**当虚拟机启动时，当verify选项被打开时，如果static方法、private方法、构造函数等，其中的直接引用（第一层关系）到的类都在同一个dex文件中，那么该类会被打上CLASS\_ISPREERIFIED**标志**。**如果一个类被打上了CLASS\_ISPREERIFIED这个标志，那么这个类就会进行dex的校验，如果该类引用的另外一个类在另一个dex文件，就会报错。简单来说，就是在打补丁之前，所修复的类已经被打上标志，通过补丁去修复bug的时候这个时候就不能完成校验，就会报错。

举个例子：

public class BugClass{

public String bug(){

return "bug class";

}

}

public class LoadBugClass{

public String getBugString(){

BugClass bugClass = new BugClass();

return bugClass.bug();

}

}

在app里面有个类叫做LoadBugClass，再其内部引用了BugClass。发布过程中发现BugClass有编写错误，那么想要发布一个新的BugClass类，那么就要阻止LoadBugClass这个类打上CLASS\_ISPREVERIFIED的标志。也就是说，在生成apk之前，就需要阻止相关类打上CLASS\_ISPREVERIFIED的标志了。对于如何阻止，在类的构造函数里面插入了一段代码，代码如下：

if (ClassVerifier.PREVENT\_VERIFY) {

System.out.println(AntilazyLoad.class);

}

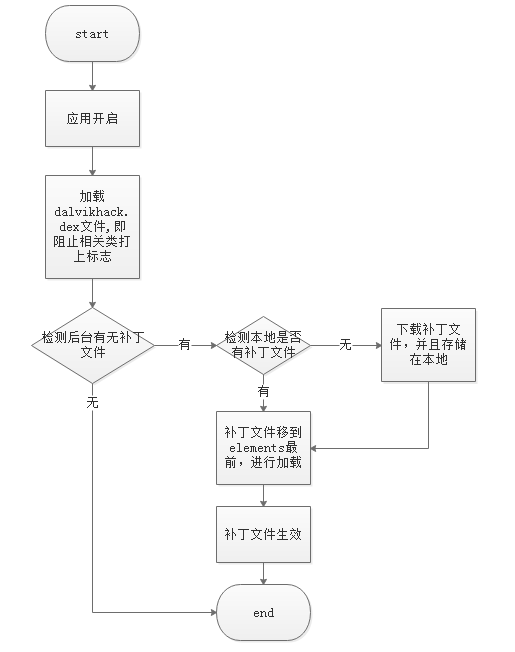
其中AntilazyLoad类会被打包成单独的hack.dex，这样当安装apk的时候，classes.dex内的类都会引用一个在不相同dex中的AntilazyLoad类，这样就防止了类被打上CLASS\_ISPREVERIFIED的标志了，只要没被打上这个标志的类都可以进行打补丁操作。

总的来说，实现动态补丁修复的技术其实就是两件事：

1、动态改变BaseDexClassLoader对象间接引用的dexElements；

2、在app打包的时候，阻止相关类去打上CLASS\_ISPREVERIFIED标志。

整体的流程图如下所示：

****

**三、使用RocooFix热修复框架实现对代码的修复**

1、在Application初始化框架 RocooFix.init(this)；

2、加载补丁的方法；

RocooFix.applyPatch(Context context, String dexPath);

该方法为从指定的目录加载补丁。一般补丁文件都是需要移动后台获取，所以需要网络请求获取补丁文件，存放在本地目录然后加载补丁文件（流程见附录第二点）。

RocooFix.initPathFromAssets(Context context, String assetName);

该方法从assets里取出补丁，一般用于测试。

3、在root的build.gradle增加如下内容:

repositories {

jcenter()

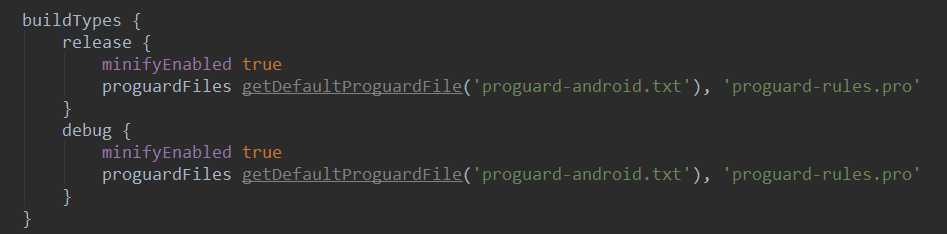
}

dependencies {

classpath 'com.dodola:rocoofix:1.2.6’

}

4、开启混淆



5、在项目的build.gradle文件里添加如下配置

apply plugin: 'com.dodola.rocoofix'

repositories {

jcenter()

}

rocoo\_fix {

includePackage = ['com/dodola/rocoosample']//指定将来可能需要制作补丁的package(就是指定插庄的范围,可以不写)

excludeClass = ['BaseApplication.class']//将不需要加到patch里的类写在这里(不需要插庄的类，可以不写)

preVersionPath = '1'//注意：此项属性只在需要制作补丁的时候才需开启！！如果不需要制作补丁则需要去掉此项

enable = true//注意：关掉此项会无法生成Hash.txt文件

scanref=true//默认为 false，开启这个选项会将与补丁 class 相引用的 class 都打入包中来解决 ART 虚拟机崩溃问题。

}

dependencies {

compile 'com.dodola:rocoo:1.1'

}

在这里需要介绍一下preVersionPath这个属性的作用，当不需要打补丁的时候，这句代码需要注释或删除。假如上一个没有打补丁发布的时候版本是1，那么在打补丁文件时，preVersionPath属性要打开，并且值是1。然后将当前的将build.gradle的versionCode的号码修改，这里修改成2（只要和之前的版本不同就可以，没有具体值的要求），然后运行程序，那么补丁文件就生成了。



**四、生成补丁包的原理**

1、在正式版本编译后，会生成一份缓存文件（hash.txt），里面记录了所有文件的md5，还有一份mapping混淆文件。

2、在打补丁包版本的时候，打开补丁包相应的配置，代码编译，生成的class文件的md5会和对应版本号的正式版本生成的md5版本进行比较，把不同的class文件打包成补丁包（patch.jar）。

**五、注意事项：**

1、任何在补丁加载前的类（比如Application及其子类）都不能插入Hack.class   
 2、必须开混淆   
 3、includePackge 包名是 /分割 不是 .分割   
 4、excludeClass a.class,b.class 不带包名

5、由于补丁包是在Application的onCreate中动态注入的，如果补丁包过大，会使app的启动速度变慢，因此，一定要注意，补丁包不宜过大。

6、为什么选择构造方法插入代码，从而让该类不打上CLASS\_ISPREVERIFIED标志。因为在构造方法插入代码不增加方法数，一个类即使没有显式的构造函数，也会有一个隐式的默认构造函数。

7、混淆。需要把以下的代码添加到混淆文件中。

-keep class com.dodola.rocoofix.\*\* {\*;}

-keep class com.lody.legend.\*\* {\*;}

-keepclassmembers class com.dodola.rocoosample.\*\* {

public <init>();//保留init,和include package保持一致

}

**6、常见问题：**

1、问题：java.lang.IllegalAccessError: Class ref in pre-verified class resolved to unexpected implementation

原因：插桩（就是去掉CLASS\_ISPREVERIFIED）没有成功

解决办法：把需要插桩的类去掉CLASS\_ISPREVERIFIED标志。

2、patch.jar生成不了

原因：操作流程不对

解决办法：执行正确的流程（见附录第一点）

3、rocoofix目录没有生产相应的版本目录

原因：没有开启混淆

解决办法：开启混淆

4、Unsupported major.minor version 52.0

原因：高版本的java project使用了低版本的来运行

解决办法：使用高版本Jdk编译 ，例如1.8

5、如果编译的最新SDK版本小于15，将会报错

解决办法：把最低的编译版本minSdkVersion修改成15或15以上的版本

6、Error:Execution failed for task app rocooJarBeforeDexDebug

原因：gradle 版本太低

解决办法：使用com.android.tools.build:gradle:2.1.2

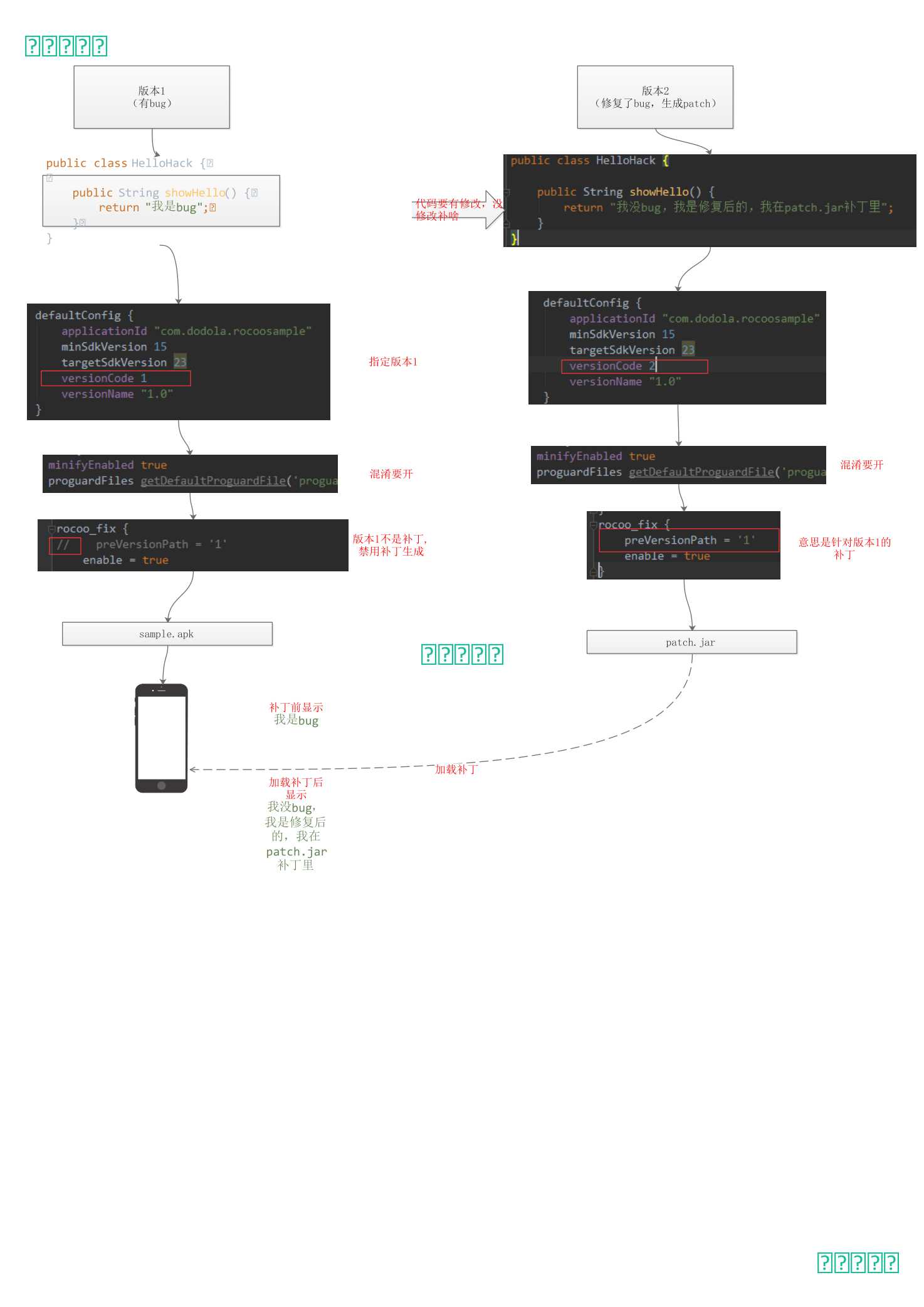
7、>com/android/dx/cf/direct/AttributeFactory

原因：gradle 版本太低

解决办法：使用com.android.tools.build:gradle:2.1.2

附录：

1、补丁生成流程简介：



2、在线补丁包获取流程思路：

