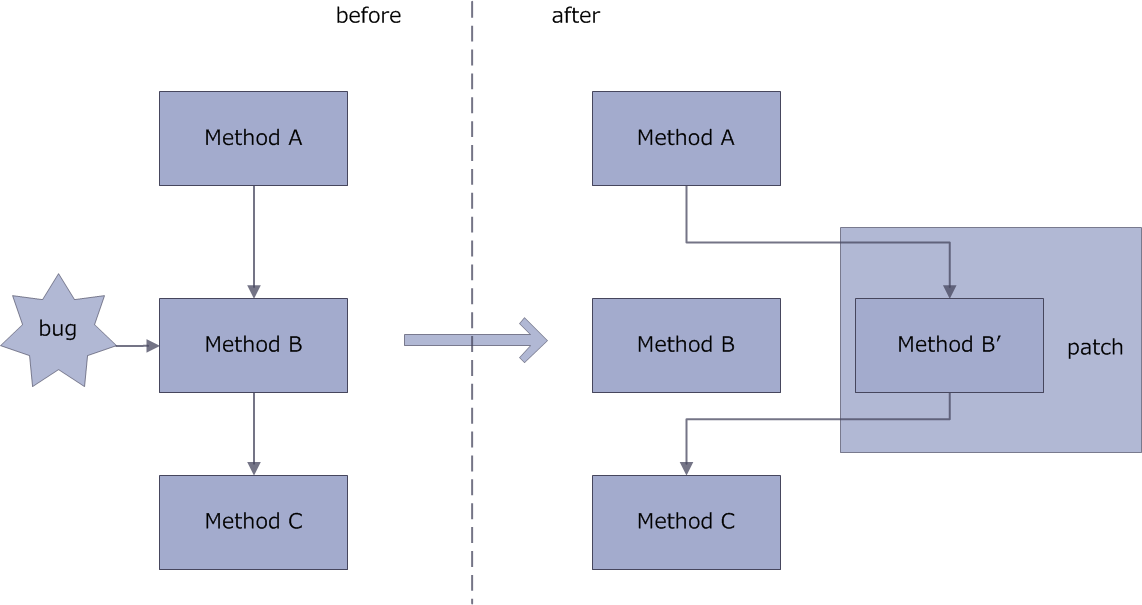
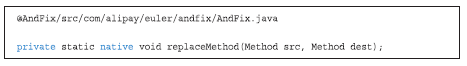
热修复Andfix原理及实践

1. 原理

热修复Andfix的原理简单的说就是方法体的替换。

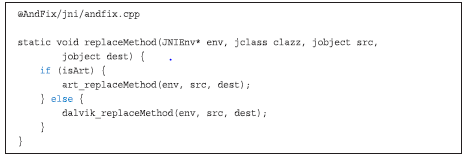


它是在已经加载了的类中直接在native层替换掉原有方法，是在原来类的基础上进行修改。核心在native层的replaceMethod函数。

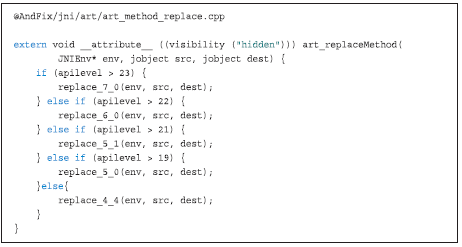


参数是在java层通过反射机制得到的Method对象所对应的jobject，src对应是需要被替换的方法，dest对应新方法，新方法在补丁包中。

Android的java运行环境，在4.4以下用的是dalvik虚拟机，而在4.4以上用的是art虚拟机。因此替换有两种方法：



因为相同的虚拟机一样存在不同的Android版本，比如相同的art虚拟机存在5.0，6.0等，而且不同版本的底层java对象是不同的，因此需要进一步的区分不同的替换函数。

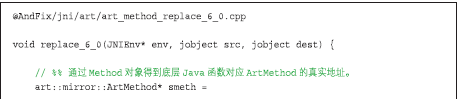


下面以6.0为例。

每个java方法在art虚拟机中都对应着一个ArtMethod。ArtMethod记录了这个java方法所有的信息，包括所属类，访问权限，代码执行地址等。

所以上面所说的方法的替换，实际上就是将新方法中的ArtMethod所有成员替换旧方法ArtMethod中的所有成员。

以replace\_6\_0()为例：



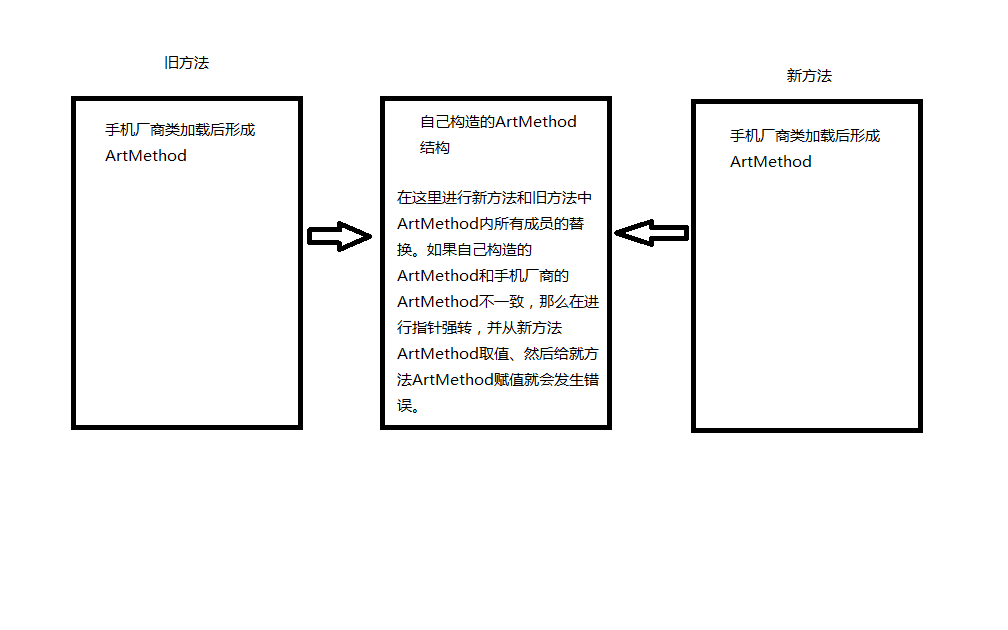


我们通过env->FromReflectedMethod，可以有Method对象得到这个方法对应的ArtMethod的真正起始地址，然后就可以把把强转为ArtMethod指针，从而对其所有成员进行修改。

但是！！！但是！！！但是！！！这里的ArtMethod结构是alibaba根据Android开源代码中的ArtMethod结构就行重写的，和实际运行设备中的ArtMethod结构可能会不一样，这种情况发生在手机厂商没有完全使用Android开源代码中的ArtMethod，而是对其进行了修改，那么这种情况下用自己构造的ArtMethod结构替换成手机厂商修改过的ArtMethod，那么里面的成员就会发生错乱，导致热修复失败。

可能大家会有疑问ArtMethod什么时候存在的？

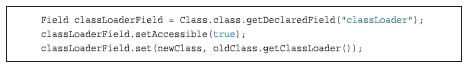
类加载完成后，得到的是一个Class对象。这个Class对象关联有一系列的ArtField对象和ArtMethod对象。其中，ArtField对象描述的是成员变量，而ArtMethod对象描述的是成员函数。



因此这就导致了很大的局限性，手机厂商修改了ArtMethod就无法实现热修复了。这也是Andfix不支持很多机型的原因。

使用Andfix进行热修复还有其他的局限性：

1. 发生bug的类，修bug后形成的新类不能改变方法的数量，也不能改变成员的数量，原因是这样的，一旦补丁类中出现了方法的增加和减少，就会导致这个类以及整个Dex的方法数的变化，方法数的变化伴随着方法索引的变化，这样在访问方法时就无法正常地索引到正确的方法了。字段的增加和减少同理。
2. 修复了的非静态方法会被反射调用。这种情况下是什么原因呢？通过反射可以得到Method，而这个Method是新类中的，包含的类信息也是新类的，当invoke的时候，传递进去的却是旧类的实例，这时候由于两个类的ClassLoader不一致，但是无法正确的invoke，进而抛出异常，这个的解决方法是将新类的ClassLoader修改为旧类的ClassLoader。



**Demo实践**

**Andfix Github：https://github.com/alibaba/AndFix**

1.新建一个Android Studio Project Andfix

2.在app的Gradle文件中添加必要的依赖，点击同步

compile 'com.alipay.euler:andfix:0.5.0@aar'



其中compile 'com.android.support:appcompat-v7:25.0.1'只是为了实现兼容，动态申请权限使用。

3.新建Application子类AndFixApplication，并在Manifest中添加

4.在onCreate方法中实例化PatchManager并初始化，并load加载。

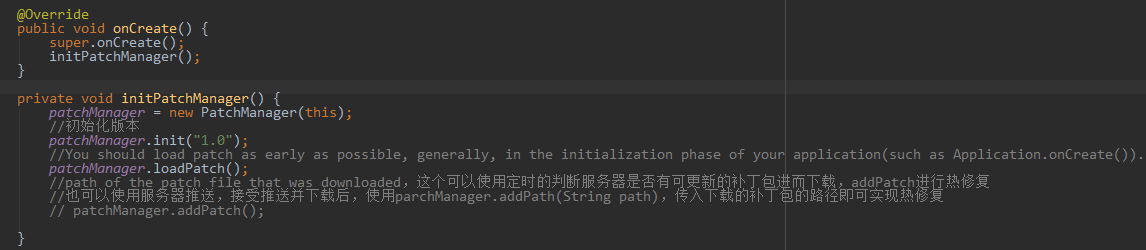
patchManager = new PatchManager(this);

//初始化版本

patchManager.init("1.0");

//You should load patch as early as possible, generally, in the initialization phase of your //application(such as Application.onCreate()).

patchManager.loadPatch();



5.接着通过parchManager.addPath(String path)方法传入补丁包的路径即可，便可完成热修复。

补丁包的获取：

1.可以使用定时的判断服务器是否有可更新的补丁包进而下载，parchManager.addPath(String path)进行热修复，传入下载的补丁包的路径。

2.也可以使用服务器推送，接受推送并下载后，使用parchManager.addPath(String path)，传入下载的补丁包的路径。

6.实际应用中，可以通过以上方法进行修复，而在本demo中，是直接将补丁包放到--/Android/date/本应用包名(com.example.andfix)/file中，

因此在AndFixApplication定义了补丁包的路径为String final

//默认的补丁路径  
public final static String *PATCH\_PATH* = "/storage/emulated/0/Android/data/com.example.andfix/files/";

另外为了让这个文件夹存在，在initPatchManager方法中添加了this.getExternalFilesDir(null).getAbsolutePath();

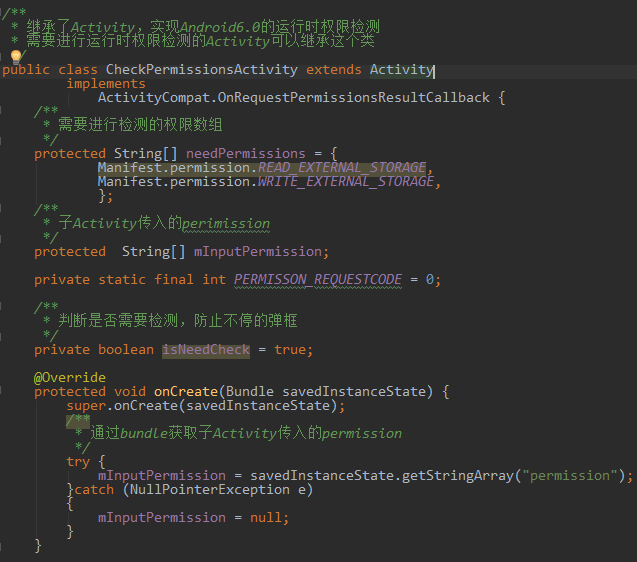
因此AndFixApplication的所有内容是：

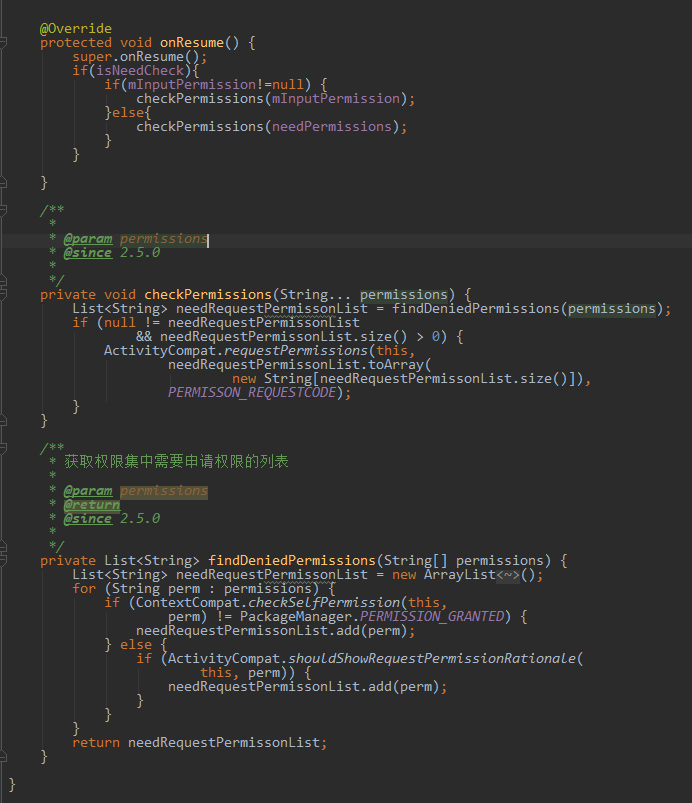


7.因为需要访问本地存储。所以需要添加访问权限。

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>  
<uses-permission android:name="为android.permission.WRITE\_EXTERNAL\_STORAGE"/>

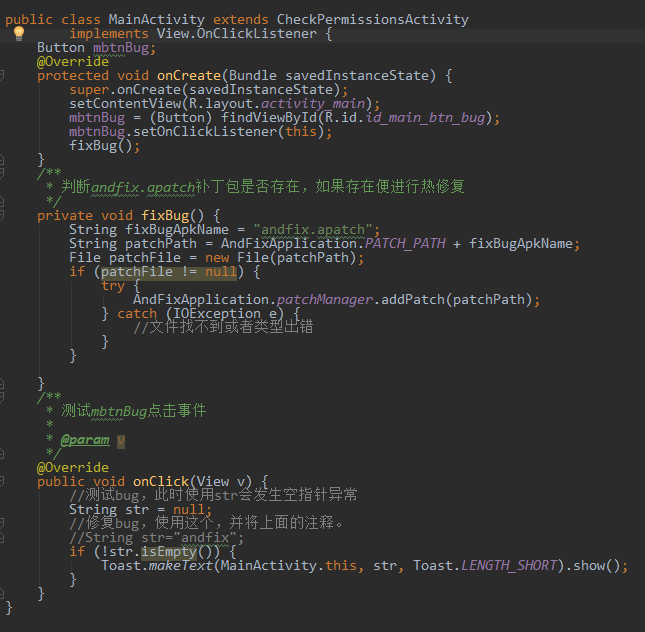
由于Android6.0需要动态权限的申请，所以添加了CheckPermissionsActivity类，是之后添加的Activity的父类，CheckPermissionsActivity类的主要作用就是动态判断当前Activity需要的权限是否已经获取，不获取并动态申请。CheckPermissionsActivity的内容





8.接着修改MainActivity继承自CheckPermissionsActivity，并且让Button控件mbtnBug的点击事件中触发bug—NPE。

内容如下：



9.接着build一个带签名的apk并命名为andfix-old,接着装在手机上。

点击按钮后，应用会Crash掉。

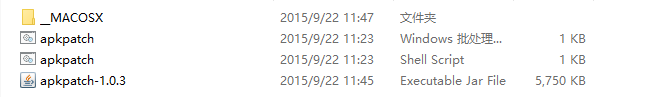
10.接在将MainActivity中的onClick方法里面的//String str="andfix";注释去掉并将String str = null;注释，也就是修复了bug。

11.此时再用相同的签名证书build带签名的apk，并明明为andfix-new.

12.此时下载apkpatch 工具

<https://github.com/alibaba/AndFix/raw/master/tools/apkpatch-1.0.3.zip>

解压后：



13进入cmd，并cd到解压的目录

通过以下命令生成.patch文件。

usage: apkpatch -f <new> -t <old> -o <output> -k <keystore> -p <\*\*\*> -a <alias> -e <\*\*\*>

-a,--alias <alias> keystore entry alias.

-e,--epassword <\*\*\*> keystore entry password.

-f,--from <loc> new Apk file path.

-k,--keystore <loc> keystore path.

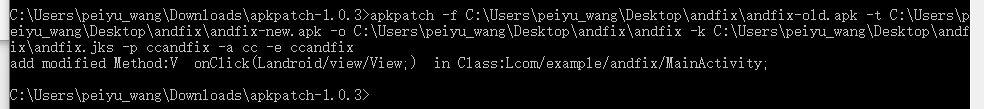
-n,--name <name> patch name.

-o,--out <dir> output dir.

-p,--kpassword <\*\*\*> keystore password.

-t,--to <loc> old Apk file path.

生成.patch文件



补充：

如果有多个bug，多个.patch，可以使用一下命令将.patch整合

usage: apkpatch -m <apatch\_path...> -o <output> -k <keystore> -p <\*\*\*> -a <alias> -e <\*\*\*>

-a,--alias <alias> keystore entry alias.

-e,--epassword <\*\*\*> keystore entry password.

-k,--keystore <loc> keystore path.

-m,--merge <loc...> path of .apatch files.

-n,--name <name> patch name.

-o,--out <dir> output dir.

-p,--kpassword <\*\*\*> keystore password.

14因为在程序中判断.patch文件是否文件，的文件名是andfix.apatch，所以将生成的patch文件重新命名为andfix.apatch，然后将该文件拷贝到…/Android/data/com.example.andfix/files/文件夹中，此时重新打开应用

点击按钮，不会Crash，已经可以正常弹出andfix。

15demo展示结束。