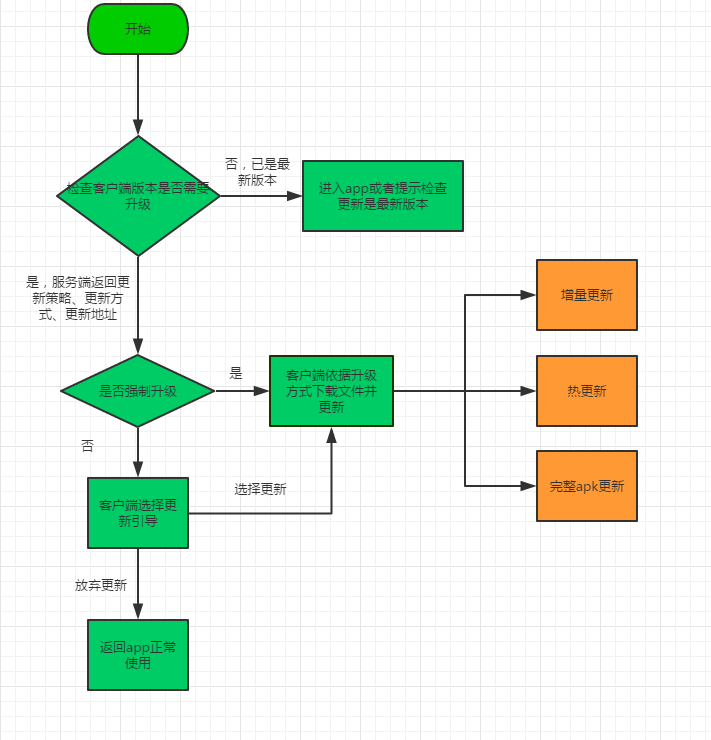
### 升级流程：



**接口请求字段:**

1.**versionName** 版本名称 String

2. **versionCode** 版本号 String 通过版本号,判断是否有最新版本和 差分升级版本

3.**clientType**  系统类型 int 如andorid 手机 android 平板 ios 等

4.**channelType**  渠道号 int 如360市场 百度市场 等

5.**packageName**  包名 String

**接口返回字段设计：**

*/\*\**

*\* 版本号*

*\*/*

**private String versionCode;**

*/\*\**

*\* 版本名称*

*\*/*

**private String versionName;**

*/\*\**

*\* 平台类型*

*\*/*

**private int client;**

*/\*\**

*\* 包名*

*\*/*

**private String packageName;**

*/\*\**

*\* 渠道号*

*\*/*

**private int channelName;**

*/\*\**

*\* 旧的版本apk的MD5值*

*\*/*

**private String oldApkMD5;**

*/\*\*  
 \* 0:非强制更新  
 \* 1：强制更新*

*\* 2. 不需要更新  
 \*/***private int updatePolicy**;  
*/\*\*  
 \* 0：完整apk更新  
 \* 1：增量更新  
 \* 2：热更新  
 \*/***private int updateMethod**;  
*/\*\*  
 \* 增量更新(或热更新)文件的地址  
 \*/***private** String **updatePatchUrl**;  
*/\*\*  
 \* 升级后的版本  
 \*/***private** String **updateVersion**;  
*/\*\*  
 \* 更新说明  
 \* 复文本  
 \*/***private** String **updateMessage**;  
*/\*\*  
 \* 更新版本发布的时间  
 \*/***private long releaseTime**;

*/\*\**

*\* 下载文件的MD5*

*\*/*

**private String fileMD5;**

*/\*\**

*\* 下载文件的长度*

*\*/*

**private String fileLength;**

*/\*\**

*\* patchCode 为补丁版本号, 如果patch 有 多个, 则需要判断手机端apk补丁升级版本,增量升级需要*

*\*/*

**private String patchCode;**

*/\*\**

*\* updateUrl 全apk下载url*

*\*/*

**private String updateUrl;**

### 一、增量更新：

步骤：1、生成差分包patch

2、客户端下载patch

3、客户端旧安装包和patch合成新安装包

4、通过安全校验，安装apk

增量更新的原理，就是将手机上已安装版本apk与服务器端最新apk进行二进制对比，得到差分包，用户更新程序时，只需要下载差分包，并在本地使用差分包与已安装apk，合成新版apk。

例如，当前手机中已安装微博V1，大小为12.8MB，现在微博发布了最新版V2，大小为15.4MB，我们对两个版本的apk文件差分比对之后，发现差异只有3M，那么用户就只需要要下载一个3M的差分包，使用旧版apk与这个差分包，合成得到一个新版本apk，提醒用户安装即可，不需要整包下载15.4M的微博V2版apk。

apk文件的差分、合成，可以通过 [开源的二进制比较工具 bsdiff](http://www.daemonology.net/bsdiff/) 来实现，又因为bsdiff依赖bzip2，所以我们还需要用到 [bzip2](http://www.bzip.org/downloads.html). bsdiff中，bsdiff.c 用于生成差分包，bspatch.c 用于合成文件。

弄清楚原理之后，我们想实现增量更新，共需要做3件事：

在服务器端，生成两个版本apk的差分包；

在手机客户端，使用已安装的apk与这个差分包进行合成，得到新版的微博apk；

校验新合成的apk文件是否完整，MD5或SHA1是否正确，如正确，则引导用户安装；

通过bsdiff 和 bspatch 差分包工具分别生成差异文件和重新打包apk

目前已经封装好 bspatch库并通过demo工程测试成功。

命令：bsdiff oldfile(旧apk包) newfile(最新apk包) patchfile(生成的差分文件存储位置)  
例如: bsdiff xx\_v1.0.apk xx\_v2.0.apk xx.patch

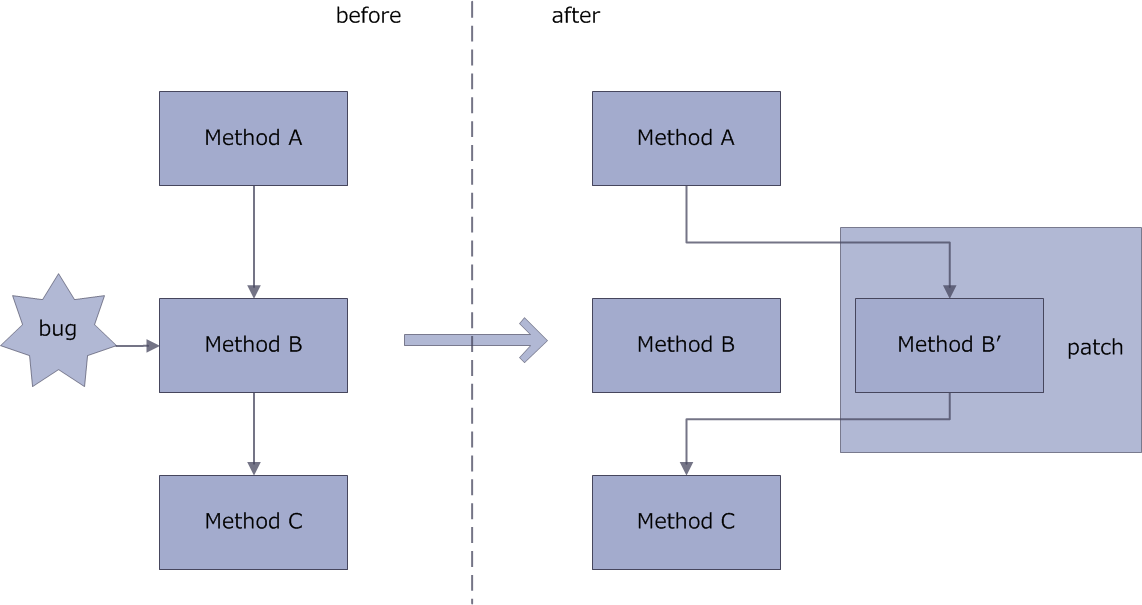
bspatch的命令格式为：  
bspatch oldfile(旧apk包) newfile(重新打包apk) patchfile(差分文件)

**ps:差分包生成策略**

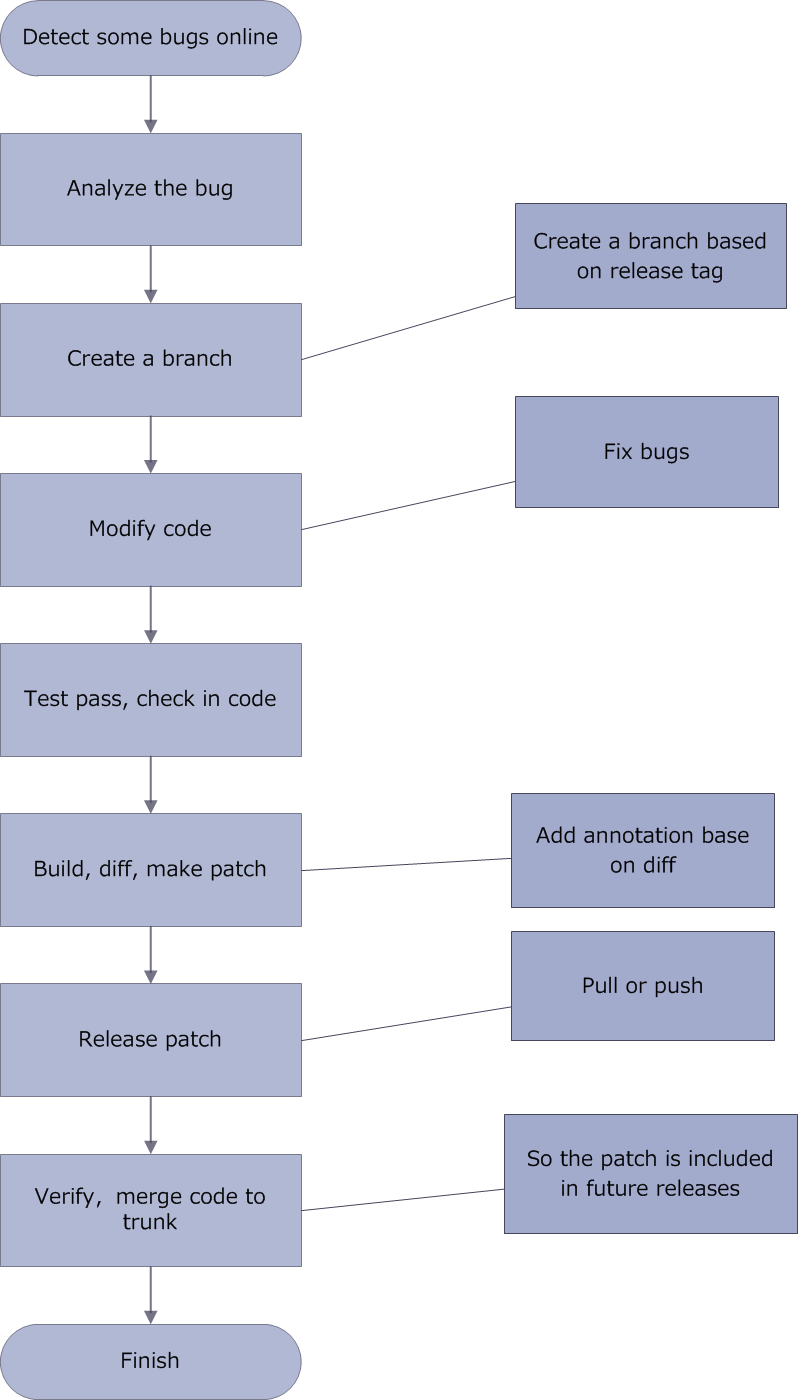
app端 打包差分升级包, 并告知服务器差分升级向下兼容的版本号

服务器 记录最新版本支持的向下支持的差分升级的版本号,通过请求apk版本号字段查询是否支持差分升级,比如当前最新版本为3.0 ,支持差分升级的版本为2.5 和 2.8, 那么当版本号为2.5的数据请求的时候, 则支持升级到3.0的差分升级

### 二、热更新：



类似git版本控制的patch过程



热更新目前有几家做得比较好的，分别是:

1、百度的实现：RocooFix\HotFix

2、大众点评的实现：NuWa

3、阿里巴巴的实现：AndFix (c库成功编好，已经写了测试工程测试通过，未全面测试)

步骤：

1. 对比安装包生成apatch文件
2. 客户端下载apatch文件
3. 客户端通过AndFix运行时加载apatch文件

**AndFix原理分析：**

apkpatch将两个apk做一次对比，然后找出不同的部分。可以看到生成的apatch了文件，后缀改成zip再解压开，里面有一个dex文件。通过jadx或者jd-gui查看一下源码，里面就是被修复的代码所在的类文件,这些更改过的类都加上了一个\_CF的后缀，并且变动的方法都被加上了一个叫@MethodReplace的annotation，通过clazz和method指定了需要替换的方法。  
 AndFix库会根据annotation来寻找需要替换的方法。最后由JNI层完成方法的替换。

AndFix缺陷：

1、无法添加新类和新的字段

2、需要使用加固前的apk制作补丁，但是补丁文件很容易被反编译，也就是修改过的类源码容易泄露。

**HotFix库实现：**

**HotFix 原理:**

参考文章

1. http://blog.csdn.net/lmj623565791/article/details/49883661

2.https://mp.weixin.qq.com/s?\_\_biz=MzI1MTA1MzM2Nw==&mid=400118620&idx=1&sn=b4fdd5055731290eef12ad0d17f39d4a&scene=1&srcid=1106Imu9ZgwybID13e7y2nEi#wechat\_redirect QQ空间团队APP热补丁

3.https://m.oschina.net/blog/308583 设备的分包技术

**HotFix 升级步骤**

**1.**生成替换出现bug类的jar文件

App中有如下几个类

BugClass 出现bug的类

BugLoad 引用了bug的类

那么我们需要将BugClass该类替换掉

重写写这个类, 其包名,类需要与原来的类保持一致

需要将该BugClass 转成jar, 并将该jar使用dx工具进行转化

**bin>jar cvf patch.jar com/danale/main/\* //将class文件转为jar包**

**bin>dx --dex --output patch\_dex.jar patch.jar //将jar包进行dx转化**

然后将patch\_dex.jar, 放置到assets 文件夹下.( 当然该文件同样可以放置到网上,这里为了方便处理)

ps: dx 工具在..\sdk\build-tools\android-4.4W,文件夹中,可以配置其环境变量,便于使用

**2.**防止应该该bug类的类被打上CLASS\_ISPREVERIFIED标签

**( 1 )**也就是说需要防止BugLoad类被打上CLASS\_ISPREVERIFIED标签, 那么就需要让BugLoad类引用其他dex中的类,才能防止被打上标签. 所以需要定义一个

帮助类. 如 AntilazyLoad

将该类同样进行1的操作

**bin>jar cvf hack.jar dodola/hackdex/AntilazyLoad //将class文件转为jar包**

**bin>dx --dex --output hackdex\_dex.jar hack.jar //将jar包进行dx转化**

将 hackdex\_dex.jar, 放置到assets文件夹下,(该jar需要在app发布的时候已经定义好了)

**( 2 )** 需要将BugLoad中注入AntilazyLoad引用. 由于BugLoad 和 AntilazyLoad 不在同一个dex中, 所以在代码编写的时候是不可能引用到AntilazyLoad. 所以需要将AnilazyLoad 注入到BugLoad的class字节码文件中, 这里就需要使用到javassist库了. 当我们不再更改BugLoad,那么BugLoad的字节码文件也就不会重新生成, 那么我们可以拿到该字节码文件将 AntilazyLoad注入到BugLoad字节码中,如下

String path = "D:/git-work-montum/MainApp/bin/";

ClassPool classes = ClassPool.getDefault();

classes.appendClassPath(path + "classes");

CtClass c = classes.get("com.danale.main.BugLoad");

CtConstructor ctConstructor = c.getConstructors()[0];

**ctConstructor.insertAfter("System.out.println(dodola.hackdex.AntilazyLoad.class);");**

c.writeFile(path+"/output");

然后将 classes文件下的BugLoad.class 替换为 bin/ output/... 目录下的BugLoad.class, 可以使用反编译工具查看BugLoad.class 是否被注入成功,如下

**3.**导入HotFix资源文件实现热修复

**( 1 )**此时assets文件加下已经有两个jar了, hackdex\_dex.jar 和 patch\_dex.jar , 由于AntilazyLoad是被注入到BugLoad中的, 为了防止AntilazyLoad找不到class, 需要在app的application的onCreate 方法中将hackdex\_dex.jar加载进来, 如下:

File dexPath = new File(getDir("dex", Context.MODE\_PRIVATE), "hackdex\_dex.jar");

Utils.prepareDex(this.getApplicationContext(), dexPath, "hackdex\_dex.jar");

HotFix.patch(this, dexPath.getAbsolutePath(), "dodola.hackdex.AntilazyLoad");

Class<?> loadClass = this.getClassLoader().loadClass("dodola.hackdex.AntilazyLoad");

ps: Utils工具类就是将jar文件拷贝到dexPath处

**( 2 )** 前面的准备工作做完了之后,就剩下最后一步了, 将patch\_dex.jar中的BugClass 加载进来从而替换掉出现问题的类,如下

//准备补丁,从assert里拷贝到dex里

File dexPath = new File(getDir("dex", Context.MODE\_PRIVATE), "patch\_dex.jar");

Utils.prepareDex(this.getApplicationContext(), dexPath, "patch\_dex.jar");

HotFix.patch(this, dexPath.getAbsolutePath(), "com.danale.main.\*");

至此, android 热补丁完成了

### 三、完整apk更新：

一般更新流程

* 版本检查 --- 下载apk --- 引导安装apk
* 升级界面 和 数据迁移（延后）。

更新流程代码实现地址：<https://github.com/DroidKOF/UpdateEngine>