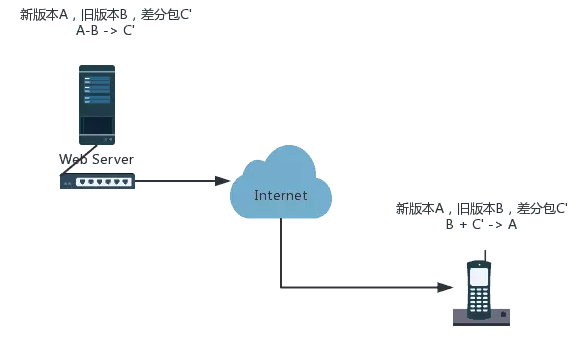
# 概述

热修复的东西，偶尔聊天谈到了增量更新，当然了两个完全不是一个东西

# 增量更新原理



## 1、增量更新主要分为两步：

    1）服务端拿新版本A和旧版本B做差分，生成差分包C‘

    2）客户端检测到可增量更新的差分包，下载差分包C‘之后，和本地旧版本B做合成，生成新版本A。

## 2、步骤详细展开：

**服务器端：**服务端的同学拿到客户端同学开发的新版本A，跟已发布的旧版本B1，B2，B3...做了差分生成相应的差分包C1，C2，C3...，并生成相应差分包的MD5值，当然全量包的签名、MD5值也是需要的，这样客户端需要的所有数据就OK了。

**客户端：**用户手动更新或程序主动请求检测更新：

        1）客户端用MD5值和版本号作为参数向服务端请求更新数据，若服务端没有差分包则返回全量包下载URL、MD5值、签名值。

        2）若服务端存在相应的差分包则返回差分包下载URL，全量包签名值、全量包和差分包MD5值，全量包签名值和MD5值。把差分包下载到本地之后（C1），先做MD5值校验，确保下载的差分包数据的完整性，校验失败则走全量更新逻辑，校验无误和本地现有安装的旧版本（B1）进行差分合并生成新版本（A），之后进行合成版本的MD5值校验和签名校验，确保合成文件的完整性和签名信息的正确性。校验无误进行安装。

## 3、需要考虑的一些问题：

    1）服务端生成的差分包大小接近新包大小，或者直接超过新包大小，就没必要进行差分更新；

    2）下载到本地之后是否需要进行签名校验依赖各自情况，若有和系统方进行合作的，系统方一般会拿APK进行二次签名之后作为系统内置应用。

    3）下载文件当然也需要支持断点续传，考虑再细点，下载APK的过程中有可能被劫持或者被运营商重定向，如果是全量更新下载，可以和服务端约定每段下载数据的校验逻辑规则，在HTTP头中附加校验字段数据，确保万无一失；

    4）服务端是否根据客户端的更新请求实时生成差分数据？从目前生成差分包的测试数据来看，这个实现是不靠谱的。最好就是有新版本之后，在服务端先把差分包数据准备好，而不是等到请求更新的时候再生成差分包。

# 二、现有增量更新实现方案

#### 1、BSDiff/Patch

这个实现方案是最多的，网上一大堆都是这个方案的实现，Android系统也整合了这个实现。更多资料可参考[Binary diff/patch](http://www.daemonology.net/bsdiff/)

external/bsdiff/

#### 2、HPatch

    博客资料参考[开源我的基于字节的数据补丁算法库HDiffPatch](https://blog.csdn.net/housisong/article/details/9003013" \t "_blank)，GitHub代码资源：[HDiffPatch](https://github.com/sisong/HDiffPatch" \t "_blank)，这份代码资源也提供了三个实现方案的对比测试，但是不是基于Android APK文件的差分，所以测试结果数据跟下面的测试结果有差异。

    另附上：HDiffPatch和BSDiff/Patch两个方案Android Demo实现[GitHub代码资源](https://github.com/snowdream/android-diffpatch" \t "_blank)，解决了不熟悉C开发的环境编译配置问题。

#### 3、XDelta

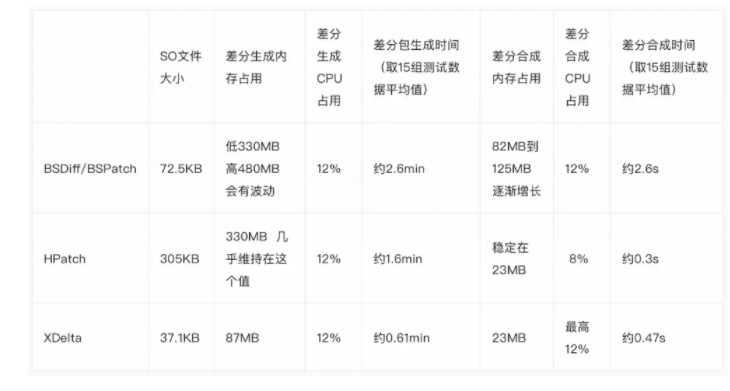
    参考[XDelta官网](http://xdelta.org/" \t "_blank)，这边需要注意的是**必须基于3.0.4版本**，最新的版本编译生成的SO得到的测试结果有问题，生成的差分包很大，差分包的合成也有问题，对比了两个版本的代码，只在几个小地方的处理逻辑有差异，那些逻辑看着也不像是导致问题的原因，如果有熟悉XDelta和C开发的大神知道原因烦告知下原因。

#### 4、Courgette

    用在Chrome 浏览器的更新上，在BSDiff/BSPatch基础上改进的，性能更优，But...不适用于Android APK更新，详细可参考[Courgette测试报告](https://blog.csdn.net/echo000001/article/details/38081229" \t "_blank)。

    想要了解更多Courgette的内容可参考[Courgette官方文档](http://dev.chromium.org/developers/design-documents/software-updates-courgette" \t "_blank)。

# BSDiff/BSPatch、HPatch、XDelta测试数据对比



#### 1、获取当前APK文件路径

    当一个APK安装之后，系统中保留有一份APK备份文件，通过以下代码方式获取APK文件的绝对路径。

public String getInstallPath(){

        try{

            String packageName = getPackageName();

            ApplicationInfo info = getPackageManager().getApplicationInfo(packageName, 0);

            return info.sourceDir;

        }catch (Exception e){

            return null;

        }

}

#### 2、MD5值校验

    参考博客：[Android数据加密之MD5加密](https://www.cnblogs.com/whoislcj/p/5885006.html" \t "_blank)

#### 3、签名校验

    分别获取当前APK签名值，合成APK的签名值，对比无误就可以进行安装了

    /\*\*

    \* 获取当前APK签名

    \*

    \* @param context

    \* @return

    \*/

    private Signature[] getSignatureInfo(Context context) {

        PackageManager pm = context.getPackageManager();

        List apps = pm.getInstalledPackages(PackageManager.GET\_SIGNATURES);

        Iterator iter = apps.iterator();

        while (iter.hasNext()) {

            PackageInfo packageinfo = iter.next();

            String packageName = packageinfo.packageName;

            String currentPackage = getPackageName();

            if (packageName.equals(currentPackage)) {

                return packageinfo.signatures;

            }

        }

        return null;

    }

    /\*\*

    \* 获取外部APK文件签名

    \*

    \* @param apkFile APK文件路径

    \* @return

    \*/

    public static Signature[] getSignatureInfo(String apkFile){

        DisplayMetrics metrics = new DisplayMetrics();

        metrics.setToDefaults();

        Object pkgParserPkg = null;

        Class[] typeArgs = null;

        Object[] valueArgs = null;

        try {

            Class packageParserClass = Class.forName("android.content.pm.PackageParser");

            Constructor packageParserConstructor = null;

            Object packageParser = null;

            //由于SDK版本问题，这里需做适配，来生成不同的构造函数

            if (Build.VERSION.SDK\_INT > 20) {

                //无参数 constructor

                packageParserConstructor = packageParserClass.getDeclaredConstructor();

                packageParser = packageParserConstructor.newInstance();

                packageParserConstructor.setAccessible(true);//允许访问

                typeArgs = new Class[2];

                typeArgs[0] = File.class;

                typeArgs[1] = int.class;

                Method pkgParser\_parsePackageMtd = packageParserClass.getDeclaredMethod("parsePackage", typeArgs);

                pkgParser\_parsePackageMtd.setAccessible(true);

                valueArgs = new Object[2];

                valueArgs[0] = new File(apkFile);

                valueArgs[1] = PackageManager.GET\_SIGNATURES;

                pkgParserPkg = pkgParser\_parsePackageMtd.invoke(packageParser, valueArgs);

            } else {

                //低版本有参数 constructor

                packageParserConstructor = packageParserClass.getDeclaredConstructor(String.class);

                Object[] fileArgs = { apkFile };

                packageParser = packageParserConstructor.newInstance(fileArgs);

                packageParserConstructor.setAccessible(true);//允许访问

                typeArgs = new Class[4];

                typeArgs[0] = File.class;

                typeArgs[1] = String.class;

                typeArgs[2] = DisplayMetrics.class;

                typeArgs[3] = int.class;

                Method pkgParser\_parsePackageMtd = packageParserClass.getDeclaredMethod("parsePackage", typeArgs);

                pkgParser\_parsePackageMtd.setAccessible(true);

                valueArgs = new Object[4];

                valueArgs[0] = new File(apkFile);

                valueArgs[1] = apkFile;

                valueArgs[2] = metrics;

                valueArgs[3] = PackageManager.GET\_SIGNATURES;

                pkgParserPkg = pkgParser\_parsePackageMtd.invoke(packageParser, valueArgs);

            }

            typeArgs = new Class[2];

            typeArgs[0] = pkgParserPkg.getClass();

            typeArgs[1] = int.class;

            Method pkgParser\_collectCertificatesMtd = packageParserClass.getDeclaredMethod("collectCertificates", typeArgs);

            valueArgs = new Object[2];

            valueArgs[0] = pkgParserPkg;

            valueArgs[1] = PackageManager.GET\_SIGNATURES;

            pkgParser\_collectCertificatesMtd.invoke(packageParser, valueArgs);

            // 应用程序信息包, 这个公开的, 不过有些函数变量没公开

            Field packageInfoFld = pkgParserPkg.getClass().getDeclaredField("mSignatures");

            Signature[] info = (Signature[]) packageInfoFld.get(pkgParserPkg);

            return info;

        } catch (ClassNotFoundException e) {

            e.printStackTrace();

        } catch (NoSuchMethodException e) {

            e.printStackTrace();

        } catch (Exception e) {

            e.printStackTrace();

        }

        return null;

    }

    /\*\*

    \* 签名值对比

    \*

    \* @param s1

    \* @param s2

    \* @return

    \*/

    private boolean compareSignatures(Signature[] s1, Signature[] s2) {

        if (s1 == null) {

            return false;

        }

        if (s2 == null) {

            return false;

        }

        HashSet set1 = new HashSet();

        for (Signature sig : s1) {

            set1.add(sig);

        }

        HashSet set2 = new HashSet();

        for (Signature sig : s2) {

            set2.add(sig);

        }

        // Make sure s2 contains all signatures in s1.

        if (set1.equals(set2)) {

            return true;

        }

        return false;

    }

# 集成到app上面

https://github.com/cundong/SmartAppUpdates

Android 增量更新全解

https://www.jianshu.com/p/6cda8fd3464a