#### 腾讯--增量升级为什么减少升级代价，增量升级原理

在前几年，整体移动网络环境相比现在差很多，加之流量费用又相对较高，因此每当我们发布新版本的时候，一些用户升级并不是很积极，这就造成了新版本的升级率并不高。而google为了解决了这个问题，提出了Smart App Update，即增量更新（也叫做差分升级）。

尽管现在网络环境有了很大的提升，但一个不争的事实就是应用越做越大，因此，增量更新在目前的仍然是一种解决APP更新包过大的有效方案。今天，我们就来聊聊增量更新。

##### 什么是增量更新？

增量更新的关键在于如何理解增量一词。来想想平时我们的开发过程，往往都是今天在昨天的基础上修改一些代码，app的更新也是类似的：往往都是在旧版本的app上进行修改。这样看来，增量更新就是原有app的基础上只更新发生变化的地方，其余保持原样。

与原来每次更新都要下载完整apk包的做法相比，这样做的好处显而易见：每次变化的地方总是比较少，因此更新包的体积就会小很多。比如“师父说”安装包的体积在6m左右，如果不采用增量更新，用户每次更新都需要下载大约6m左右的安装包，而采用增量更新这种方案之后每次只需要下载2m左右的更新包即可，相比原来做法大大减少了用户下载等待的时间。

##### 增量升级优势在哪里？

​ 3.差分的优势

* 大小非常小
* 安全-必须是特定的节点才能进行升级
* 相对于整包来说更容易控制

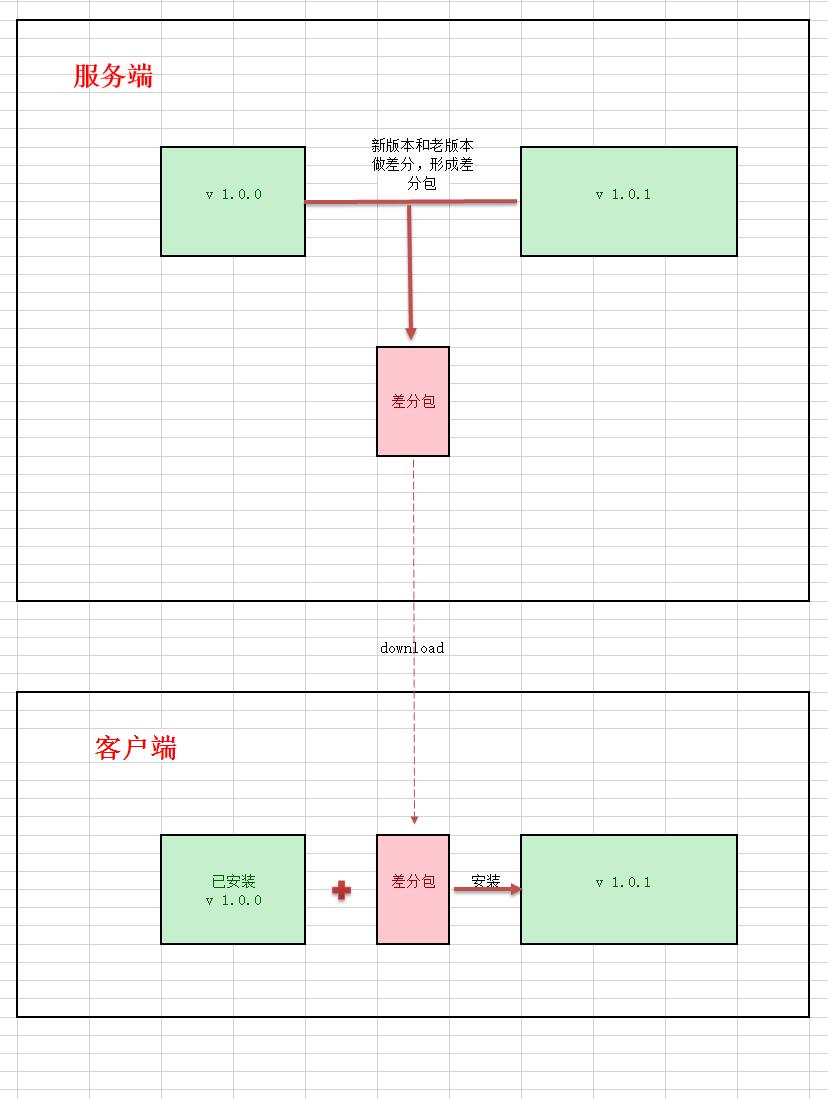
##### 为什么会减少升级代价

例如旧版本的APK有5M，新版的有8M，更新的部分则可能只有3M左右(这里需要说明的是，得到的差分包大小并不是简单的相减，因为其实需要包含一些上下文相关的东西)，使用差分升级的好处显而易见，

那么你不需要下载完整的8M文件，只需要下载更新部分就可以，而更新部分可能只有3、4M，可以很大程度上减少流量的损失。

##### 增量更新的原理

增量更新的原理非常简单，简单的说就是通过某种算法找出新版本和旧版本不一样的地方（这个过程也叫做差分），然后将不一样的地方抽取出来形成所谓的更新补丁（patch），也称之为差分包。客户端在检测到更新的时候，只需要下载差分包到本地，然后将差分包合并至本地的安装包，形成新版本的安装包，文件校验通过后再执行安装即可。本地的安装包通过提取当前已安装应用的apk得到。

演示：差分包的生成与合并 ##### 如下图所示： 

现在的问题在于如何生成差分包以及合并差分包。这里，我们借助开源库[bsdiff](http://www.pokorra.de/coding/bsdiff.html)来解决以上两个问题。首先我们先演示一下差分包的形成与合并。

下载bsdiff\_win\_exe.zip，解压到本地。如下图:

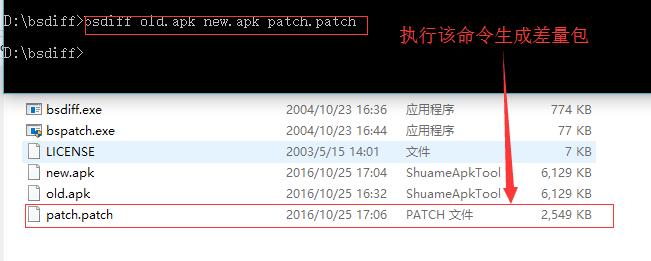


è¿éåå¾çæè¿°

然后，我们先打出一个安装包，假设为old.apk。对源码做修改后，再打出一个新的安装包new.apk。此处old.apk相当于老版本的应用，而new.apk相当于新版本的应用。接下来，我们利用bsdiff来生成差分包patch.patch。

##### 生成差分包

将上面的old.apk和new.apk放入bsdiff解压后的目录，然后在控制台中执行命令bsdiff old.apk new.apk patch.patch,稍等一会便可以生成差分包patch.patch，如下

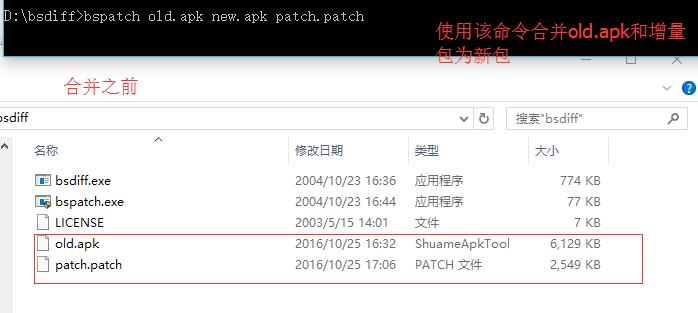


åè¡¨åå®¹

##### 合并差分包

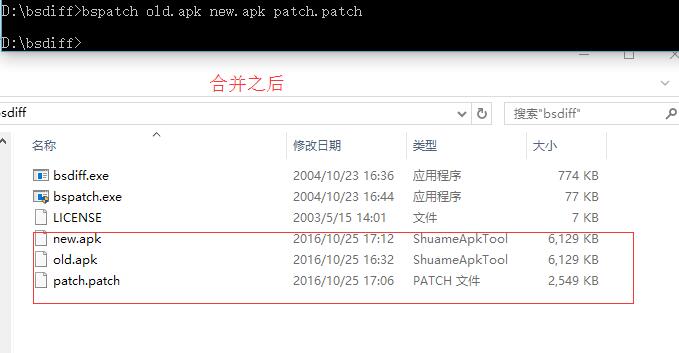
合并old.apk和patch.patch，生成新的安装包new.apk。只要此处合并出来的new.apk和上面我们自己打出来的new.apk一样，那么就可以认为它就是我们需要的新版本安装包。

我们来看看如何合并。将old.apk和patch.patch放入bsdiff文件夹，合并之前为



è¿éåå¾çæè¿°

然后执行命令bspatch old.apk new.apk patch.patch，稍等一会之后便可以看到合并出的new.apk.如下：



è¿éåå¾çæè¿°

不出意外，合并而来的new.apk应该和我们自己打出来的new.apk是一模一样的，这可以通过验证两者的md5来认定。

我们已经弄明白增量更行是怎么一回事。下面，我们就以“师父说”为对象进实践一把。

##### 实践：让师父说支持增量更新

客户端支持增量更新总体和上面的演示差不多，唯一的区别在于客户端要自行编译bspatch.c来实现合并差分包，也就是所谓的ndk开发，这里我们首先要下载bsdiff的源码以及bszip的源码，以便后面使用。在as中如何进行ndk开发不是本文的重点。

1.编写BsPatchUtil类 BsPatchUtil中只有一个natvie方法patch(String oldApkPath,String newApkPath,String patchPath)用于实现增量包的合并：

public class BsPatchUtil {  
 static {  
 System.loadLibrary("apkpatch");  
}  
  
   
public static native int patch(String oldApkPath, String newApkPath, String patchPath);

##### 2.编写C代码

在实现BsPatchUtil之前，我们需要将bspatch.c以及bzip的相关代码拷贝到jni目录下（bzip只保留.h头文件和.c文件）。并将bspatch.c中的main()方法名修改为executePatch()，并且修改其中bzip的引入头为#include "bzip2/bzlib.h".目录结构如下：

##### è¿éåå¾çæè¿°

注意:上图当中的em.c是一个空文件，用来避免在window下编译产生的未知错误。

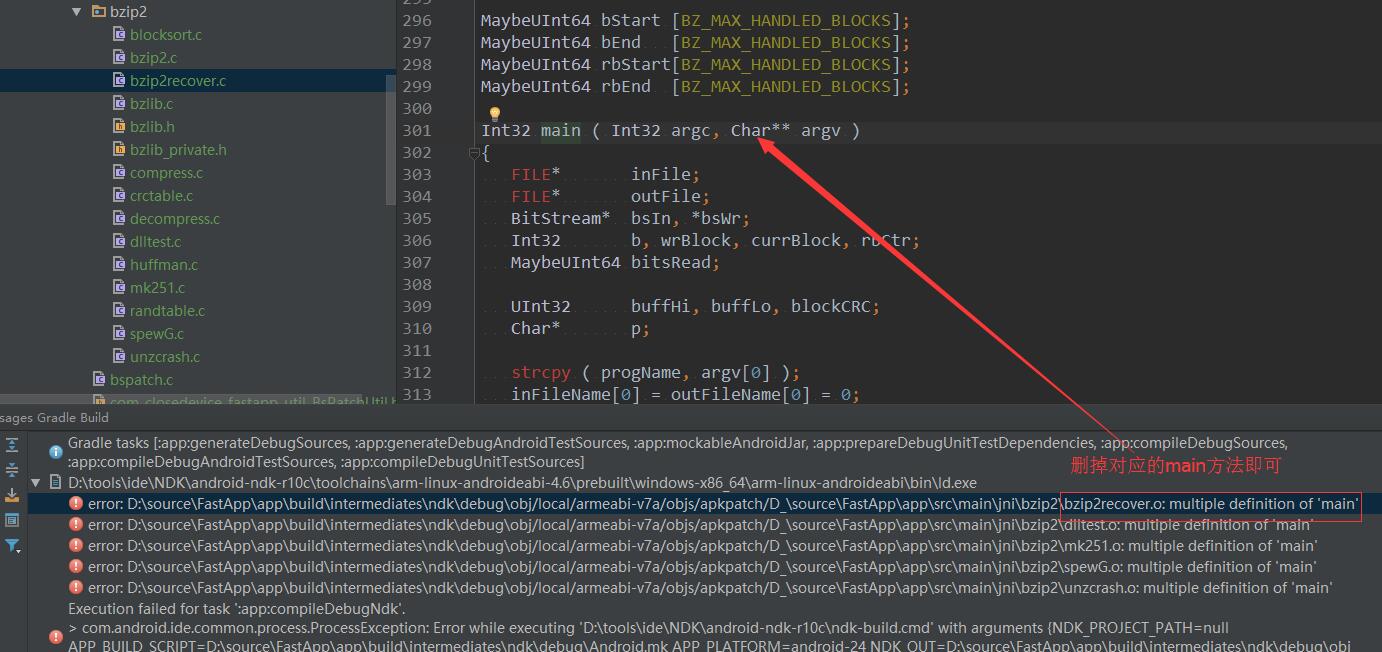
接下来我们就可以在bspatch\_util.c中实现相关的代码了：

#include "com\_closedevice\_fastapp\_util\_BsPatchUtil.h"  
  
JNIEXPORT jint JNICALL Java\_com\_closedevice\_fastapp\_util\_BsPatchUtil\_patch  
 (JNIEnv \*env, jclass clazz, jstring old, jstring new, jstring patch){  
 int args=4;  
 char \*argv[args];  
 argv[0] = "bspatch";  
 argv[1] = (char\*)((\*env)->GetStringUTFChars(env, old, 0));  
 argv[2] = (char\*)((\*env)->GetStringUTFChars(env, new, 0));  
 argv[3] = (char\*)((\*env)->GetStringUTFChars(env, patch, 0));  
  
 //此处executePathch()就是上面我们修改出的  
 int result = executePatch(args, argv);  
  
 (\*env)->ReleaseStringUTFChars(env, old, argv[1]);  
 (\*env)->ReleaseStringUTFChars(env, new, argv[2]);  
 (\*env)->ReleaseStringUTFChars(env, patch, argv[3]);  
  
 return result;  
}

至此，大部分工作已经完成了。配置app moudle中的build.gradle中添加ndk配置

defaultConfig {  
 applicationId "com.closedevice.fastapp"  
 minSdkVersion 14  
 targetSdkVersion 24  
 versionCode 1  
 versionName "1.0.0"  
 testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
  
 //ndk配置  
 ndk{  
 moduleName "apkpatch"  
 abiFilters "armeabi", "armeabi-v7a","x86"  
 }  
 }

接下来，我们编译试试（ndk环境的配置这里不做说明，自行配置即可），不出意外会遇到以下错误：



è¿éåå¾çæè¿°

该问题的解决方法也非常简单，注释掉对应文件的main()方法即可。重新编译，不出意外没什么问题了。接下来，我们就需要在合适的地方合并差分包了。

##### 3.合并差分包

上面的过程做完之后，就可以通过BsPatchUtil.patch()来合并当前安装包和差分包了。

这里，我们假设差分包已经从服务器下载到本地了。

首先来看如何获取当前安装包。我们安装的应用通常在、data/app下，可以通过一下代码获取其路径：

public static String getApkInstalledSrc(){  
 return BaseApplication.context().getApplicationInfo().sourceDir;  
 }

下面就可以通过BsPatchUtil.patch(String oldApkPath,String newApkPath,String pathPath)来进行合并了。此处需要注意两点：

1. 合并的地方建议放在外置存储（SDcard）当中
2. 合并的过程比较耗时，需要放到子线程中进行。

### 4.安装

任何更新包在下载完成后首先要做的就是进行ＭＤ５校验，以便确认该更新包是正规途径下载而来的。同样，对于合并之后的更新包，首先要做的事情也是进行ＭＤ5校验，校验通过之后，再进行安装：

public static void installAPK(Context context, File file) {  
 Intent intent = new Intent();  
 intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);  
 intent.setAction(Intent.ACTION\_VIEW);  
 intent.setDataAndType(Uri.fromFile(file),  
 "application/vnd.android.package-archive");  
 context.startActivity(intent);  
 }

到现在，增量更新已经完成。现在可以把增量包以及合并之后的安装包进行删除了。

大体代码如下：

private void smartupdate() {  
 Observable.create(new Observable.OnSubscribe<File>() {  
 @Override  
 public void call(Subscriber<? super File> subscriber) {  
 //定义生成的新包  
 File newApk = new File(Environment.getExternalStorageDirectory(), "newApk.apk");  
  
 //假设patch.patch文件已经下载到sdcard上，切已经校验通过  
 File patch = new File(Environment.getExternalStorageDirectory(), "patch.patch");  
  
 if(!patch.exists()){  
 subscriber.onError(new IOException("patch file not exist!"));  
 return;  
  
 //合并差分包  
 BsPatchUtil.patch(OSUtil.getApkInstalledSrc(), newApk.getAbsolutePath(), patch.getAbsolutePath());  
 if (newApk.exists()) {  
 subscriber.onNext(newApk);  
 subscriber.onCompleted();  
 patch.delete();  
 }else{  
 subscriber.onError(new IOException("bspatch failed,file not exist!"));  
 }  
  
  
 }  
 }).subscribeOn(Schedulers.newThread())  
 .observeOn(AndroidSchedulers.mainThread())  
 .doOnSubscribe(new Action0() {  
 @Override  
 public void call() {  
 showDialog("正在应用差分包");  
 }  
 })  
 .subscribe(new Subscriber<File>() {  
 @Override  
 public void onCompleted() {  
 hideDialog();  
 }  
  
 @Override  
 public void onError(Throwable e) {  
 hideDialog();  
 LogUtils.d(e.getMessage());  
 }  
  
 @Override  
 public void onNext(File file) {  
 OSUtil.installAPK(getActivity(),file);  
 }  
 });  
  
  
 }

##### 增量更新的缺点

增量更新虽让有效的解决了更新包过大的问题，但是存在以下几点问题：

1. 客户端和服务端需要加入相应的支持。每次发布新版本，服务端都需要为以前所有的老版本生成对应的差分包，并根据客户端端请求返回对应的更新包，维护过程将会变得相对复杂。客户端需要对差分包做更为详细的验证，防止出错，除此之外，客户端应该可以根据服务端更新开关来确定当前是使用完整更新还是增量更新。
2. apk包之间的差异过小时，比如2m以下，此时生成的差分包仍然有几百k，此时使用增量更新得不偿失，毕竟形成差分包和合并的过程都非常耗时。另外，但版本之间变化非常大的时候，通常是是大版本好变化的时候，比如从v 1.0.0到2.0.0,此时使用完整更新也不错。

在师父说中已经添加主要代码，可自行练习。效果如下：



1.gif