笔墨Android
2019年04月24日 阅读 731

Android增量更新

首先增量更新应该了解个概念：增量更新：在版本较近的apk升级的时候，根据两个apk之间的差异（生成差异包），合成新的安装包，在应用内部进行升级的一种操作（需要重新安装apk文件）。热更新：在发布的版本有BUG的时候，动态加载dex文件，在不影响apk的情况下进行修复BUG（不需要重新安装apk文件）。

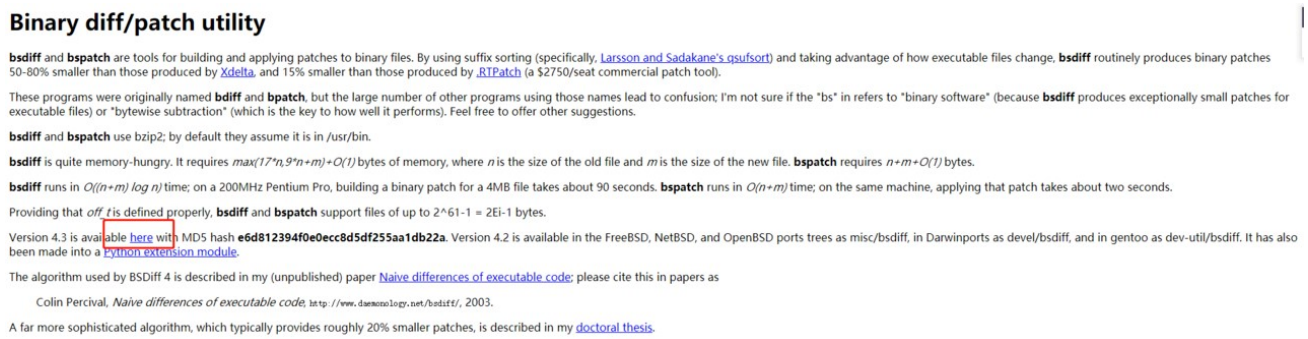
所以从本质上增量更新和热更新是不同的！这个概念要明确。。。

本文知识点：

- android 增量更新的一些工具
- android 增量更新的配置（JNI的配置）
- android 增量更新的开发
- android 遇到的一些问题
- demo地址的分享

所谓工欲善其事，必先利其器。所以工具是很重要的！这里先和大家介绍一下相应的工具！！！

- 1.1 bsdiff-4.3下载地址(用于生成新的安装包)



Binary diff/patch utility

bsdiff and **bspatch** are tools for building and applying patches to binary files. By using suffix sorting (specifically, Larsson and Sadakane's qsufsort) and taking advantage of how executable files change, **bsdiff** routinely produces binary patches 50-80% smaller than those produced by Xdelta, and 15% smaller than those produced by .RTPatch (a $2750/seat commercial patch tool).

These programs were originally named **bdiff** and **bpatch**, but the large number of other programs using those names lead to confusion; I'm not sure if the "bs" in refers to "binary software" (because **bsdiff** produces exceptionally small patches for executable files) or "bytewise subtraction" (which is the key to how well it performs). Feel free to offer other suggestions.

bsdiff and **bspatch** use bzip2; by default they assume it is in /usr/bin.

bsdiff is quite memory-hungry. It requires *max(17*n,9*n+m)+O(1)* bytes of memory, where *n* is the size of the old file and *m* is the size of the new file. **bspatch** requires *n+m+O(1)* bytes.

bsdiff runs in *O((n+m) log n)* time; on a 200MHz Pentium Pro, building a binary patch for a 4MB file takes about 90 seconds. **bspatch** runs in *O(n+m)* time; on the same machine, applying that patch takes about two seconds.

Providing that *off_t* is defined properly, **bsdiff** and **bspatch** support files of up to 2^61-1 = 2Ei-1 bytes.

Version 4.3 is available here with MD5 hash **e6d812394f0e0ecc8d5df255aa1db22a**. Version 4.2 is available in the FreeBSD, NetBSD, and OpenBSD ports trees as misc/bsdiff, in Darwinports as devel/bsdiff, and in gentoo as dev-util/bsdiff. It has also been made into a Python extension module.

The algorithm used by BSDiff 4 is described in my (unpublished) paper Naive differences of executable code; please cite this in papers as

Colin Percival, *Naive differences of executable code*, http://www.daemonology.net/bsdiff/, 2003.

A far more sophisticated algorithm, which typically provides roughly 20% smaller patches, is described in my doctoral thesis.

- 1.2 bzip2-1.0.6 用于补充bsdiff中c的缺失代码
- 1.3 bsdiff-v4.3-win-x64 用于生成相应的差异包

鉴于大家的找着费劲，所以呢~所以呢~我也没准备！哈哈！！！开玩笑，这里是我百度网盘的地址(增量更新工具 提取码为r1ws)，如果失效的话！及时联系我哦！

2. 增量更新的配置（JNI的配置）

其实每次配置jni的环境我都头疼，但是为了工作也是没办法。后来我发现一个事情，假如之前项目没有ndk那么怎么能快速集成呢？这里我每次都是新建一个ndk项目然后各种copy就好了！！！

2.1 导入bspatch.c文件(这个是在bsdiff-4.3中的，够体贴吧！！！)

其实在当你导入这个文件的时候，你会发现。妈的一堆红。。。这是在逗我吗？别急，之后就好了！！！

2.2 导入bspatch.c文件所需要的文件

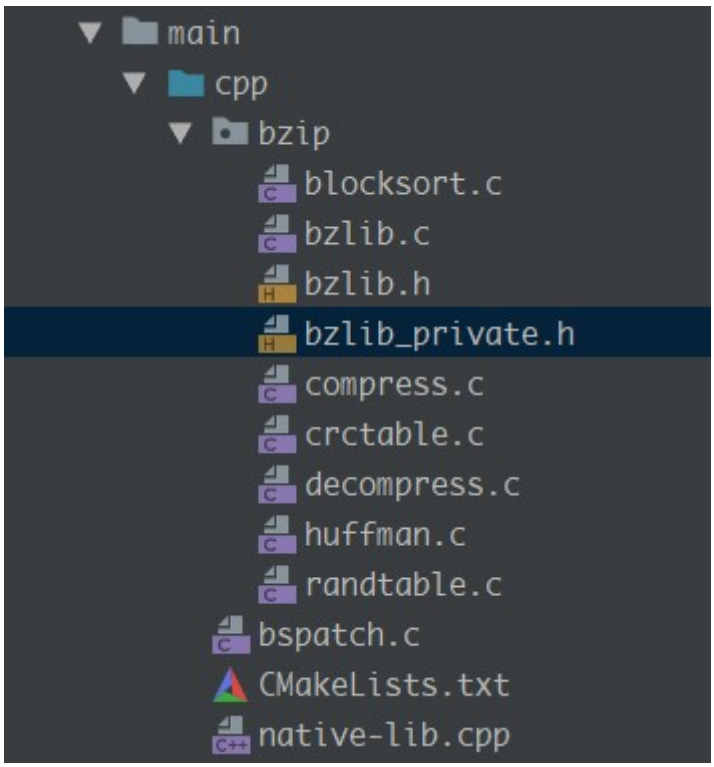
记得当初让你们下的bzip2吧？其实却得就是这个里面得一些东西！！！其实你把俩面的东西全都搞过来是可以的，但是作为一个有逼格的程序员，这怎么可能呢？那么需要导入那些文件呢？看看下面这段代码(在bzip中有一个Makefile文件中的代码):

c 复制代码

```
OBJS= blocksort.o  \
      huffman.o    \
```

```
        decompress.o \
        bzlib.o
```

对需要的就是这些文件！我把我导入的文件的示意图放上了，按照名称直接导入就好了！



这里面有一些坑需要踩：

1. 在导入bzlib.h文件的时候，android studio3.+和之前的有区别(看那个不报错用哪个吧)：

c 复制代码

```
//#include <bzlib.h>
//新版本只有这样能导入
#include "bzip/bzlib.h"
```

2. 在bspatch.c文件中有一个main方法，最好把这个名称改成别的，防止其它c代码也使用这个main方法，造成出错！

2.3 配置jni环境

这个我感觉是最繁琐的，其实配置这些真的好烦，，，行了不发牢骚了！！！

2.3.1 创建生成bsPath的方法！

cpp 复制代码

```cpp
#include <jni.h>
#include <string>

//这个主要是为了导入其中的方法
extern "C" {
extern int p_main(int argc, char *argv[]);
}
extern "C"
JNIEXPORT void JNICALL
Java_com_angle_netupdatademo_MainActivity_bsPath(JNIEnv *env, jobject instance, jstring oldApk_,

    //将java字符串转换成char指针
    const char *oldApk = env->GetStringUTFChars(oldApk_, 0);
    const char *patch = env->GetStringUTFChars(patch_, 0);
    const char *output = env->GetStringUTFChars(output_, 0);

    // 释放相应的指针gc
    env->ReleaseStringUTFChars(oldApk_, oldApk);
    env->ReleaseStringUTFChars(patch_, patch);
    env->ReleaseStringUTFChars(output_, output);
}
```

这里面的内容我们一会在盘他！！！

2.3.2 配置CMakeLists.txt文件

其实关于这块我觉得有必要系统的学习一下，所以这里我们就不做太多介绍了，我注解写的很详细的！！！

复制代码

```
cmake_minimum_required(VERSION 3.4.1)

# 查找文件系统中指定模式的路径，如/* 匹配根目录的文件（注意路径）
file(GLOB bzip_source ${CMAKE_SOURCE_DIR}/bzip/*.c)

# 设置本地动态库 编译生成动态库
add_library(
        #模块名
        native-lib
        # 动态库/分享可以
        SHARED
```

```
        #配置相应的文件引用
        bspatch.c
        #这个相当与别名引用，上面设置了一个别名
        ${bzip_source}
)

#系统库，日志输出log
find_library(
        log-lib

        log)

#需要链接或者编译的库
target_link_libraries(
        native-lib

        ${log-lib})
```

基本上就是这些配置了！！！

2.3.3 app的build.gradle配置相应的内容

这里其实就是在c++的demo中添加了相应的cpu架构的兼容

复制代码

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.angle.netupdatademo"
        minSdkVersion 14
        targetSdkVersion 28
        versionCode 1
        versionName "2.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
        externalNativeBuild {
            cmake {
                cppFlags ""
                //兼容cpi架构
                abiFilters 'armeabi-v7a'
            }
        }
```

```
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rul
        }
    }
    externalNativeBuild {
        cmake {
            path "src/main/cpp/CMakeLists.txt"
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
}
```

这里也没有什么太多好说的，关于这些建议看看jni配置的一些文章！

以上所有的配置就结束了！！！ 是不是很烦，没事，马上就看到日出了！！！

3. 增量更新的开发

其实一般的增量更新都是在splash页面中动态从服务器下载patch文件，放在某个文件夹下，然后通过之前安装apk的路径进行生成新包，最后安装。基本上就是这个流程！

所以这里需要解决的一些内容：

- pathc文件的生成
- old.apk安装的路径的获取
- 怎么生成新的安装包

分解处任务就很好做了！

3.1 patch文件的生成

解压bsdiff-v4.3-win之后会有两个文件，直接把两个android安装包放进去之后，通过命令 **bsdiff**

3.2 old.apk安装路径的获取

其实这个很简单，一行代码就能搞定

java 复制代码

```
String oldApk = getApplicationInfo().sourceDir;
```

3.3 怎么生成新的apk文件

这里就要用到我们之前没有盘的那个jni方法了！

c 复制代码

```
#include <jni.h>
#include <string>

//这个主要是为了导入其中的方法
extern "C" {
extern int p_main(int argc, char *argv[]);
}
extern "C"
JNIEXPORT void JNICALL
Java_com_angle_netupdatademo_MainActivity_bsPath(JNIEnv *env, jobject instance, jstring oldApk_,

    //将java字符串转换成char指针
    const char *oldApk = env->GetStringUTFChars(oldApk_, 0);
    const char *patch = env->GetStringUTFChars(patch_, 0);
    const char *output = env->GetStringUTFChars(output_, 0);

    //bspatch ,oldfile ,newfile ,patchfile
    char *argv[] = {"", const_cast<char *>(oldApk), const_cast<char *>(output), const_cast<char
    p_main(4, argv);

    // 释放相应的指针gc
    env->ReleaseStringUTFChars(oldApk_, oldApk);
    env->ReleaseStringUTFChars(patch_, patch);
    env->ReleaseStringUTFChars(output_, output);
}
```

看中间增加的那段代码，这个p_main就是之前让你改的那个main方法，需要四个参数，然后...转。这里面其实是c的一些知识，我也不是很明白。但是代码很好理解！

那么接下来，就直接调用这个发方法生成一个相应的apk就可以了！！！

跨度过大的情况，差分包和新包没有什么区别，不需要做差分了！

5. Demo地址

Demo地址

使用方法，生成一个apk（老的），然后增加内容，在生成一个apk（新的）。生成拆分包！然后把这两个都放在sd卡的根目录安装老的apk，然后点击更新！！！

关注下面的标签，发现更多相似文章

Android

评论

输入评论...

王为民 开发工程师
这个很赞,感谢分享
9小时前 回复

1天前 回复

笔墨Android (作者) Android工程师

回复 Star(星): 是的，所以说每个版本都有自己的patch包

23小时前 回复

相关推荐

专栏 · 秉心说 · 13小时前 · Android

方舟编译器最新技术细节

6 3

专栏 · KunMinX · 17小时前 · Android

这样理解，你也能在 30 秒内讲明白 TCP 三次握手

24 5

专栏 · Android心路历程 · 15小时前 · Android

Android进程保活方案

16 2

专栏 · blankj · 1天前 · Android

Android 侧划，如斯优雅

47 12

专栏 · hzw1490152780934 · 20小时前 · Android

瓦片地图加载框架——TiledMapView

7 1

专栏 · 胡七筒 · 2天前 · JavaScript / Android

程序猿生存指南-63 贪心姑娘

61 88

25 9

专栏 · wingjay · 1天前 · Android

Android 架构之高可用移动网络连接

53

专栏 · 胖宅老鼠 · 1天前 · Android

Android进阶知识：事件分发与滑动冲突

38

专栏 · 恋猫de小郭 · 2天前 · Android / 掘金技术征文

Android 集成 Agora SDK 快速体验 RTC 版多人视频聊天|掘金技术征文

59 12