**MediaScanner总结**

**一、时序图**

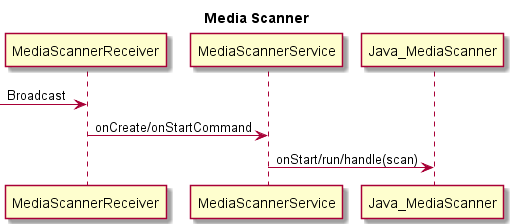


图1

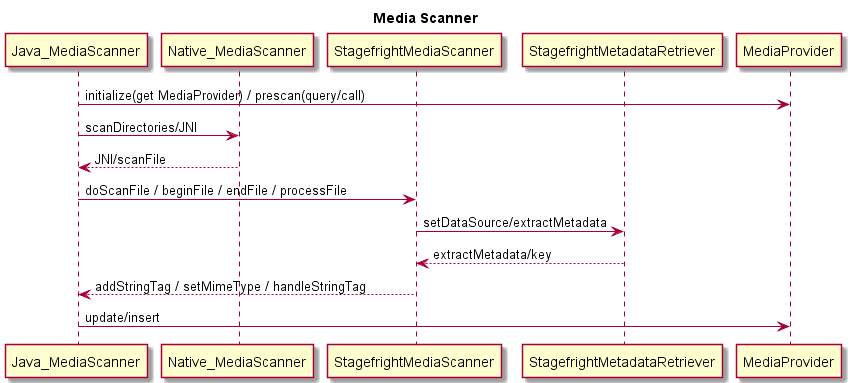


图2

**二、MediaScanner实现**

**1、MediaScanner Java部分**

进程名：android.process.media

Pakage名：com.android.providers.media

Code路径：packages/providers/MediaProvider

编译结果：system/priv-app/MediaProvider/MediaProvider.apk

**2、MediaScannerReceiver启动MediaScannerService**

**A、MediaScannerReceiver继承自BroadcastReceiver, 静态注册，监听的广播事件如下：**

android.intent.action.BOOT\_COMPLETED

android.intent.action.MEDIA\_MOUNTED

android.intent.action.MEDIA\_UNMOUNTED

android.intent.action.MEDIA\_SCANNER\_SCAN\_FILE

action\_media\_scanner\_stop

action\_media\_scanner\_start

**B、收到广播后onReceive中action：**

1. BOOT\_COMPLETED，scan INTERNAL\_VOLUME and EXTERNAL\_VOLUME
2. action\_media\_scanner\_stop，stopMediaScanner
3. action\_media\_scanner\_start，startMediaScanner。Scan EXTERNAL\_VOLUME
4. ACTION\_MEDIA\_MOUNTED，Scan EXTERNAL\_VOLUME
5. ACTION\_MEDIA\_SCANNER\_SCAN\_FILE，scanFile指定路径

**C、Scan和scanFile**

1. 通过startService启动MediaScannerService，MediaScannerService的onCreate和onStartCommand会被执行。

**3、MediaScannerService启动java MediaScanner**

MediaScannerService extends Service implements Runnable。

**A、onCreate**

第一次运行service，会调用onCreate，初始化变量mExternalStoragePaths，另启一个工作线程去扫描文件，执行耗时操作。因MediaScannerService运行在主线程中，另起线程防止block主线程。

**B、run**

工作线程会执行Runnable中的run方法。

1、设置线程优先级为11，android线程中最低优先级为19

2、Looper.prepare()给工作线程创建一个looper

3、new ServiceHandler，创建一个handler，默认绑定上面创建的looper

4、Looper.loop()，循环读取Message，Message由handler发送。Message处理，looper获取message之后，交由callback函数处理，如果没有callback函数，会走handler的派生接口handleMessage处理消息。

Loop不会立即返回，线程在一直循环等待并读取Message,退出时，需要在MediaScannerService中的onDestroy接口中调用mServiceLooper.quit()使loop退出，线程运行才会结束。

**C、onStartCommand**

每次启动service均会调用onStartCommand。

1、synchronized，同步，等待工作线程把handler创建完成。

2、根据intent创建Message，并mServiceHandler.sendMessage(msg)

3、返回值START\_REDELIVER\_INTENT。在运行onStartCommand后，如果service进程被kill，系统会自动重启service，并传入最后一个intent给onStartCommand。直到调用stopSelf()才停止传递intent。如果在被kill后还有未处理好的intent，那被kill后服务还是会自动启动。因此onStartCommand不会接收到任何null的intent。

START\_STICKY和START\_STICKY：当进程被杀死后onDestroy()是不会被执行的

START\_REDELIVER\_INTENT：当进程被杀死后onDestroy()会被执行

**D、handleMessage**

1、filepath，scanFile

2、volume，scan

3、stopSelf（startId），扫描完成之后，根据ID停止service。

4、onDestroy。mServiceLooper.quit()，退出loop。结束工作线程。

**E、scan/scanFile**

1、getContentResolver().insert。数据库操作，在MediaProvider中实现

2、sendBroadcast(new Intent(Intent.ACTION\_MEDIA\_SCANNER\_STARTED, uri))

3、openDatabase。数据库操作，MediaProvider中insert、insertInternal。

4、createMediaScanner，创建扫描器，设置扫描器中的语言和地区，和后面的字符编码有关系。

5、scanner.scanDirectories，使用扫描器开始真正的扫描

6、getContentResolver().delete扫描结束后删除scanUri

7、sendBroadcast(new Intent(Intent.ACTION\_MEDIA\_SCANNER\_FINISHED, uri))，所有存储文件扫描结束。

**4、java MediaScanner**

**a、加载libmedia\_jni.so**

Jni路径：frameworks/base/media/jni

1、调用native\_init、native\_setup，new StagefrightMediaScanner把这个native的scanner实例存放在java对象MediaScanner的私有数据mNativeContext中。

**b、scanDirectories**

取MediaProvider、audiouri、videouri等

prescan创建一个FileCache，这个FileCache是从MediaProvider中已有信息构建出来的，也就是历史信息。后面根据扫描得到的新信息来对应更新历史信息。没看懂，待分析。

Postscan，做一些清除工作，与前面记录下来的历史信息FileCache做比较，从数据库中删除掉已经被移除文件的相关信息。

**c、processDirectory，逐一扫描volume**

传给native的StagefrightMediaScanner两个参数，1、u盘路径，2、一个MyMediaScannerClient对象（存储着java中的MyMediaScannerClient对象）。

Jni code : MyMediaScannerClient myClient(env, client);

**5、native MediaScanner**

**A、processDirectory doProcessDirectory**

1、文件夹名字是.或..开头将被忽略，文件夹名字太长也将被忽略。

2、如果是文件夹，递归调用doProcessDirectory，深度优先

**B、scanFile**

1、调用jni层MyMediaScannerClient::scanFile

2、通过之前jni中存入的java MyMediaScannerClient对象，回掉java scanFile方法。

**6、java MyMediaScannerClient**

**A、scanFile**

beginFile。预处理，查看缓存中有无该文件，有则修改访问时间、长度、是否有变化等，没有则创建。已有的文件被修改过或新建文件，执行scan。

doScanFile：只扫描音视频和图片文件，音视频processFile(path, mimeType, this)，图片processImageFile

endFile：audio、video、image数据填充，例如：图片的经纬度宽高等信息，音频的作者封面年份时长等，视频的作者封面时长分辨率等。

新文件直接插入数据库（insert），数据库中已有的文件若有修改则更新（update）。

**7、native StagefrightMediaScanner**

**A、processFile/processFileInternal**

**1、new MediaMetadataRetriever**

Code路径：frameworks/av/media/libmedia

getService(String16("media.player"))得到mediaplayerservice

mRetriever = MediaPlayerService::createMetadataRetriever()

mRetriever = new MetadataRetrieverClient(pid)

code路径：frameworks/av/media/libmediaplayerservice

**2、setDataSource**

MetadataRetrieverClient::setDataSource

Player打分得到getPlayerType，createRetriever(playerType)，一共有三个：

StagefrightMetadataRetriever

MstMetadataRetriever

WhaleyMetadataRetriever

本文仅分析StagefrightMetadataRetriever

**3、extractMetadata**

获得mime\_type后调用client.setMimeType(value)，设置mimetype：

client.addStringTag(kKeyMap[i].tag, value){ handleStringTag（）}

设置数据的keymap：

static const KeyMap kKeyMap[] = {

{ "tracknumber", METADATA\_KEY\_CD\_TRACK\_NUMBER },

{ "discnumber", METADATA\_KEY\_DISC\_NUMBER },

{ "album", METADATA\_KEY\_ALBUM },

{ "artist", METADATA\_KEY\_ARTIST },

{ "albumartist", METADATA\_KEY\_ALBUMARTIST },

{ "composer", METADATA\_KEY\_COMPOSER },

{ "genre", METADATA\_KEY\_GENRE },

{ "title", METADATA\_KEY\_TITLE },

{ "year", METADATA\_KEY\_YEAR },

{ "duration", METADATA\_KEY\_DURATION },

{ "writer", METADATA\_KEY\_WRITER },

{ "compilation", METADATA\_KEY\_COMPILATION },

{ "isdrm", METADATA\_KEY\_IS\_DRM },

{ "width", METADATA\_KEY\_VIDEO\_WIDTH },

{ "height", METADATA\_KEY\_VIDEO\_HEIGHT },

};

Client存储的是java MediaScanne中的MyMediaScannerClient对象，最终会调用到java MyMediaScannerClient中的方法：setMimeType和handleStringTag，handleStringTag把数据写入ContentValues变量values中。

endFile方法调用MediaProvider类把ContentValues values写入数据库，通过MediaProvider insert/update。

**三、结束**

Scanner的架构流程到这里结束，接下来还需要分析细节和具体功能的实现，分两部分：a、metadata数据的提取Retriever的实现。b、数据库操作MediaProvider的实现。整个scanner模块涉及到ContentProvider、Service、BroadcastReceiver、Retriever。

1. **StagefrightMetadataRetriever\MstMetadataRetriever**

在new MediaMetadataRetriever和setdatasource之后会得到Retriever实例，828的code中有三个：

A、StagefrightMetadataRetriever

B、MstMetadataRetriever

C、WhaleyMetadataRetriever

数据提取extractMetadata，由个Retriever实现，去获取。

例如StagefrightMetadataRetriever：

mCtx->mExtractor = MediaExtractor::Create(mSource);

根据source创建Extractor，比如MP3Extractor,通过ID3提取音频的metadata信息。Code路径frameworks/av/media/libstagefright

**2、MediaProvider：数据库操作**

mContext是MediaScannerService创建的时候new ContextImpl被创造，Service继承ContextWrapper继承Context，getContentResolver来自ContextWrapper。最终调用到：

[frameworks](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/)/[base](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/)/[core](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/core/)/[java](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/core/java/)/[android](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/core/java/android/)/[app](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/core/java/android/app/)/[ContextImpl.java](http://androidxref.moretv.cn/source/xref/titan-m-938/frameworks/base/core/java/android/app/ContextImpl.java)中getContentResolver

在mediascanner中initialize时获得一个MediaProvider对象：

mMediaProvider = mContext.getContentResolver().acquireProvider("media")

scanner中的数据库操作均是通过MediaProvider类的接口：

query/call/insert/update

**3、scanner扫描结束之后，串口的如下目录会生成两个db文件。**

/data/data/com.android.providers.media/databases/

internal.db和external.db，分别存放内部存储设备和外部存储设备的扫描结果。