thumbnail调查

Last edited by **caoquanli** 1 month ago

Thumbnail生成调查

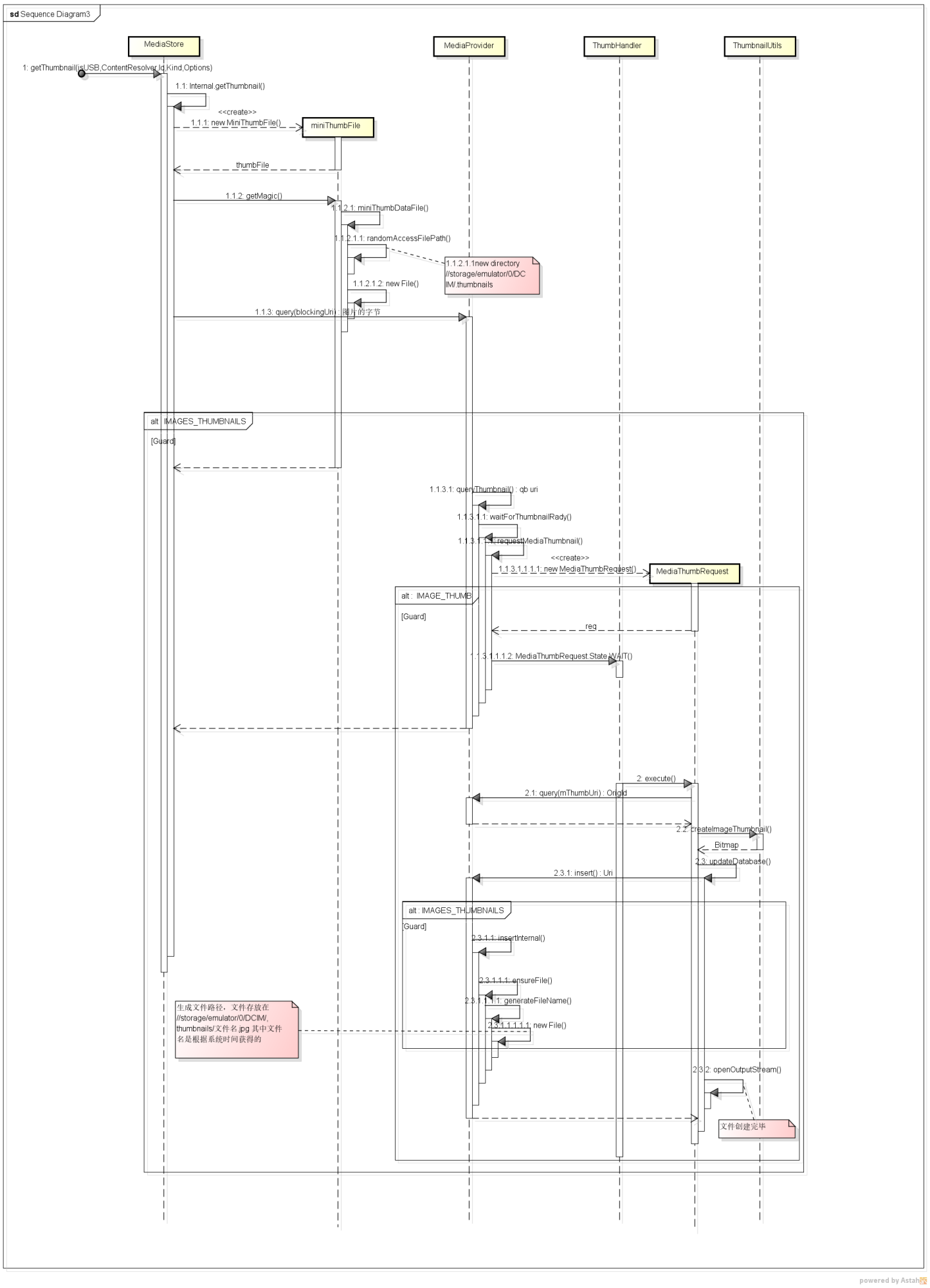
前面一到三部分都是介绍图片thumbnail的生成过程，在第四部分开始介绍歌曲artwork。

一 简介

Android提供了专门的获取图片Thumbnail的方法——getThumbnail()，这个方法有四个参数：ContentResolver cr，origId，kind，BitmapFactory.Options，第一个参数是android提供的一种用来实现数据共享的方法，第二个参数是原始数据（image）的在数据库中的唯一id，第三个参数是想要获取的Thumbnail的类型，有两种类型，第一种是MINI_KIND，这种类行的Thumbnail大小一般为512×384，为矩形，还有一种是MICRO_KIND类型，这种类型大小为96×96，是一个正方形图片，在实际应用中所获得的缩略图一般都是MICRO_KIND，第四个参数一般设置为null。通过这个方法，以及后面的一系列操作，就能获取到想要的Thumbnail。

主要涉及文件：MediaStore.java，MiniThumbFile.java，MediaProvider.java，MediaThumbRequest.java，ThumbUtils.java

二 生成缩略的时序图



三 生成流程详解

1 调用getThumbnail方法

在数据库中获取到原始图片的信息后，调用这个方法，传入参数，参数上面有解释，自行设定，就会走到生成图片thumbnail的流程中去。这个方法的实现是在InternalThumbnail中。下面去这个方法中去看。

2 调用InternalThumbnail的getThumbnail方法

这部分的主要功能就是操作文件，首先new一个MiniThumbFile，然后直接去获取源文件的magicId，这个id是在MiniThumbFile中创建文件的时候生成的文件中每张图片都会有一个id，就是这个玩意，这个id其实就是为了定位文件的，通过这个id可以判断这个文件是否已经生成。下面就去这个对象中看看。

3 MiniThumbFile

这里会创建一个文件，存放源文件的数据，存放在storage/emulated/0/DCIM/.thumbnails目录里（这个目录在MiniThumbFile中直接指定创建），此目录android专门用来存放thumbnail文件，这个文件中他给每张图片都回给一个固定大小的空间存放数据，图片越多，这个文件也就越大。这个文件在kind为MINI_KIND时只是一个过渡文件，但是kind为MICRO_KIND时第一次创建完之后后面需要thumbnail都是从这个文件中读出来。这个文件创建完之后如果不存在，返回0，如果之前已经创建好了返回一个id值。这个值就是用来判断文件是否已经生成。

4 对于magic不等于0

值不等于0的情况，两种KIND走两个路径，MINI_KIND通过查询数据库得出，MICRO_KIND通过解析上面过程生成的文件得到。如果中间没有出现错误，会返回一个bitmap。 这里源文件是怎么和缩略图对应的呢？这里会有一个origId，就是通过这个id去和源文件去对应，以至于不会造成乱取。

5 对于magic等于0

这种情况就会生成thumbnail文件，创建一个blockingUri，然后去query，通过这个uri在query的时候做了很多事，下面一一分析。

6 query

query的最终实现是在MediaProvider中，通过解析传进去的Uri，当查询的table=IMAGES_THUMBNAIS时，调用queryThumbnail()方法。

7 queryThumbnail

这里首次按会用到上面的blockingUri中发的blocking值，如果传进来的是1，才会继续下面的动作，发出处理thumbnail的请求。执行waitForThumbnailReady()方法，这个方法也是传进来一个Uri，这个Uri把上面的Uri中的thumbnail改为media，并且加了一个origId ，然后又会查询一次，这次获得的是文件路径，这里就会发出一个请求，请求处理thumbnail，并且初始状态为WAIT，等待调用的requestMediaThumbnail()方法执行完。进入这个方法，这个方法回new一个MediaThumbRequest()，初始传进去的参数有Uri，文件path，magic，还有priority，因为这里涉及到new的一个工作线程“thumb thread”来处理这个消息，所以会设置这个线程的优先级。然后会给mThumbHandler中send一个IMAGE_THUMB消息。下面看怎么处理这个消息。

8 处理消息

看MediaProvider的onCreat中会new一个线程来处理消息，然后new一个Handler来处理，这个handler就是mThumbHandler，当从队列中拿出来的消息是IMAGE_THUMB时，调用MediaThumbRequest的execute()方法。

9 execute()

这个方法首先调用MiniT humbFile，拿到第三步中创建的文件，这里会判断magic的值如果不是0，和上面一样。如果是0，调用ThumbnailUtils中的方法来创建缩略图，这里简单提一下，不细说（不过里面在获取bitmap的时候，因为图片编码格式的问题，所以获取方式也有些不同，对于格式为GPEG格式的，会从它的EXIF中获得，但是对于别的格式，走bitmapFactory中的方法）根据传进去的path值，创建出bitmap，然后调用updateDatabase()，这个方法显而易见，就是更新数据库，这个方法里会调用insert方法，这里只是会插进去一些基本信息，但是不会插路经，下面就是创建文件路径。

10 insert

显而易见，这个方法的最终实现肯定是在MediaProvider中，insert方法的实现都是在insertInternal中，看这个方法，根据传进来的uri获得match，通过判断，走case为IMAGE_THUMBNAILS，ensureFile(),这里面有个generateFileName()这个里面就会创建出这个文件的路径。然后插入到数据库中的DATA中。然后回代MediaThumbRequest，回拿到这个路径，将创建的bitmap存入里面。

11 回到getThumbnail

在查询一系列操作之后会回到这里，因为有两中kind，所以在取Thmbnail时也有两种方法，当kind为MINI_KIND时，从数据库中查询获得。当kind为MICRO_KIND时，从创建的MiniThumbFile中获得，都是调用getMiniThumbFromFile()这个方法，但是实现大的方法是不同的。

到这里Thumbnail就创建完了。

这里面只是简单的讲了一下image的缩略图的获取方法，对于video和audio在下面部分，其实走的路线都大同小异只是入口不一样。西面就详细分析一下歌曲的artwork的生成流程。

四 歌曲AlbumArt（Artwork）生成流程

现在的MP3文件在播放的时候都会显示一张图片，有很多人会问这张图片是哪里来的，其实现在的mp3中都会寸一张Attached Picture，简称AlbumArt，这个也是一个tag信息，会编在编码中，然后经过解析之后能够提取出来，在播放的时候能够展示出来。也就是通常我们看到的专辑图片。下面就来简单介绍一下他的提取流程。这部分内容只是简单的走一下流程，细节部分不做过多讲解，需要的同学自己阅读源码去分析。

具体生成的流程时序图

他的解析过程是和歌曲的解析流程紧密相连的，这部分的调用是在MediaProvider.insert()开始，在MediaScannner的endFile的会插入数据库，调用insert，然后在MediaProvider中去实现。

MediaProvider.java部分

1 insert and insertInternal

当uri匹配到是AUDIO_MEDIA的时候，会获取它的rowId，调用insertFile()

2 insertFile

根据前面传进来的mediatype查看走哪一步，如果是audio

```
case FileColumns.MEDIA_TYPE_AUDIO: {
    // SQLite Views are read-only, so we need to deconstruct this
    // insert and do inserts into the underlying tables.
    // If doing this here turns out to be a performance bottleneck,
    // consider moving this to native code and using triggers on
    // the view.
    values = new ContentValues(initialValues);

    String albumartist = values.getAsString(MediaStore.Audio.Media.ALBUM_ARTIST
    String compilation = values.getAsString(MediaStore.Audio.Media.COMPILATION)
    values.remove(MediaStore.Audio.Media.COMPILATION);

    // Insert the artist into the artist table and remove it from
    // the input values
    Object so = values.get("artist");
    String s = (so == null ? "" : so.toString());
    values.remove("artist");
    long artistRowId;
    HashMap<String, Long> artistCache = helper.mArtistCache;
    String path = values.getAsString(MediaStore.MediaColumns.DATA);
    synchronized(artistCache) {
        Long temp = artistCache.get(s);
        if (temp == null) {
            artistRowId = getKeyIdForName(helper, db,
                    "artists", "artist_key", "artist",
                    s, s, path, 0, null, artistCache, uri);
        } else {
            artistRowId = temp.longValue();
            Log.i(TAG, "get a album artist id in cache artistRowId = " + artist
        }
    }
    String artist = s;

    // Do the same for the album field
    so = values.get("album");
    s = (so == null ? "" : so.toString());
    values.remove("album");
    long albumRowId;
    HashMap<String, Long> albumCache = helper.mAlbumCache;
    synchronized(albumCache) {
        int albumhash = 0;
        if (albumartist != null) {
            albumhash = albumartist.hashCode();
        } else if (compilation != null && compilation.equals("1")) {
            // nothing to do, hash already set
        } else {
            albumhash = path.substring(0, path.lastIndexOf('/')).hashCode();
        }
        String cacheName = s + albumhash;
        Long temp = albumCache.get(cacheName);
```

```
                    if (temp == null) {
                        albumRowId = getKeyIdForName(helper, db,
                                "albums", "album_key", "album",
                                s, cacheName, path, albumhash, artist, albumCache, uri);
                    } else {
                        albumRowId = temp;
                        Log.i(TAG, "get a album row id in cache albumRowId = " + albumRowId
                    }
                }

                values.put("artist_id", Integer.toString((int)artistRowId));
                values.put("album_id", Integer.toString((int)albumRowId));
                so = values.getAsString("title");
                s = (so == null ? "" : so.toString());
                values.put("title_key", MediaStore.Audio.keyFor(s));
                // do a final trim of the title, in case it started with the special
                // "sort first" character (ascii \001)
                values.remove("title");
                values.put("title", s.trim());

                computeDisplayName(values.getAsString(MediaStore.MediaColumns.DATA), values
                break;
            }
```

做了很多事，一步步分析，根据initialValues创建ContentValues对象，从该对象获取到albumartist，compilation，然后获取artist，如果为null则赋值为空，然后获取一个artistCache的hashmap对象，第一次从map中获取artist则肯定为null，然后就会调用到getKeyIdForName，之后就是对album做处理，同样的也是会调到getKeyIdForName，只是传进去的参数是不一样。下面去看看getKeyIdForName这个方法。

3 getKeyIdForName

首先通过query去查询keyFileId，计算查到的数量，第一次肯定是0，如果是0，则执行下面的步骤

```
String [] selargs = { k };
       helper.mNumQueries++;
       Cursor c = db.query(table, null, keyField + "=?", selargs, null, null, null);

       try {
           switch (c.getCount()) {
               case 0: {
                       // insert new entry into table
                       ContentValues otherValues = new ContentValues();
                       otherValues.put(keyField, k);
                       otherValues.put(nameField, rawName);
                       helper.mNumInserts++;
                       rowId = db.insert(table, "duration", otherValues);
                       if (path != null && isAlbum && ! isUnknown) {
                           // We just inserted a new album. Now create an album art thumbn
                           makeThumbAsync(helper, db, path, rowId);
                       }
                       if (rowId > 0) {
                           String volume = srcuri.toString().substring(16, 24); // extract
                           Uri uri = Uri.parse("content://media/" + volume + "/audio/" + t
                           getContext().getContentResolver().notifyChange(uri, null);
                       }
                   }
                   break;
```

如果是传进来的table是album，并且path不空，album不是know,执行makeThumbAsync。

4 makeThumbAsync

这个方法的开始会给new一个ThumbData对象，然后给各项参数赋值，这个对象很重要，里面参数在后面会一直用，这点需要注意一下， 这个方法就是专门为歌曲创建artwork的，不难，就是给主线程的quene发送了一个ALBUM_THUMB的消息，当然处理去handleMessage中去看。

```
else if (msg.what == ALBUM_THUMB) {
                    ThumbData d;
                    synchronized (mThumbRequestStack) {
                        d = (ThumbData)mThumbRequestStack.pop();
                    }
```

```
                    IoUtils.closeQuietly(makeThumbInternal(d));
                    synchronized (mPendingThumbs) {
                        mPendingThumbs.remove(d.path);
                    }
```

在处理消息的时候，当消息是ALBUM_THUMB,从栈中拿出一个ThumbData对象，然后去调用makeThumbInternal。

5 makeThumbInternal

因为这部分代码有点多，只放关键部分代码

`byte[] compressed = getCompressedAlbumArt(getContext(), mExternalStoragePaths, d.path);` 后面还有一个writeAlbumArt函数，这部份之后就到了核心部分了，这个函数我们还是得回过头在详细得看看，先调用getCompressedAlbumArt方法去获取图片的字节数组，这部分在后面继续介绍他是怎么就获取到的，然后就是根据这个数组去获得bitmap对象，进行加工处理，然后有个获取uri的方法，getAlbumArtOutputUri，这个里面也做了很多事，之后再看，之后会走到writeAlbumArt这个方法中。现在到这里我们能看出就是先获取在处理，先看怎么获取的， 这行就调用到 getCompressedAlbumArt中去了，之后的部分都是对拿到的这个文件字节数组做处理，这里就不在说了，下面看 getCompressedAlbumArt这个方法。

5 getCompressedAlbumArt

```
byte[] compressed = null;

        try {
            File f = new File(path);
            ParcelFileDescriptor pfd = ParcelFileDescriptor.open(f,
                    ParcelFileDescriptor.MODE_READ_ONLY);

            try (MediaScanner scanner = new MediaScanner(context, INTERNAL_VOLUME)) {
                compressed = scanner.extractAlbumArt(pfd.getFileDescriptor());
            }
            pfd.close()
```

这部分就很关键了，首先根据文件路径获取文件描述符，然后可以看出他最终是回到MediaScanner中去了，new了一个MediaScanner对象，然后调用它的exeractAlbumArt方法，后面部分也是对获取到的compressed字节数组文件做简单处理，这里也不细说了。

6 MediaScanner.java

这部分很简单就一个exeractAlbumArt方法，而且这个方法在这里只有个定义，是一个native函数。

7 libStagefright框架部分

这部分内容流程请大家参考另一篇文章：libStagefright框架分析，走的流程都是一样的，只是方法换成了exeractAlbumArt，最终返回的是一串字节数组，这里简单介绍一下它从歌曲中去提取文件的方法。

8 MP3Extractor.cpp

在getMetaData的时候会获取它的AlbumArt，

```
const void *data = id3.getAlbumArt(&dataSize, &mime);
if (data) {
        meta->setData(kKeyAlbumArt, MetaData::TYPE_NONE, data, dataSize);
        meta->setCString(kKeyAlbumArtMIME, mime.string());
    }
```

很清楚就是掉到了id3的getAlbumArt方法，然后把取到的信息set到kKeyAlbumArt这一列中。

9 ID3.cpp

```
ID3::getAlbumArt(size_t *length, String8 *mime) const {
    *length = 0;
    mime->setTo("");

    Iterator it(
            *this,
            (mVersion == ID3_V2_3 || mVersion == ID3_V2_4) ? "APIC" : "PIC");

    while (!it.done()) {
        size_t size;
        const uint8_t *data = it.getData(&size);
        if (!data) {
            return NULL;
```

```
        }

        if (mVersion == ID3_V2_3 || mVersion == ID3_V2_4) {
            uint8_t encoding = data[0];
            size_t consumed = 1;

            // *always* in an 8-bit encoding
            size_t mimeLen = StringSize(&data[consumed], size - consumed, 0x00);
            if (mimeLen > size - consumed) {
                ALOGW("bogus album art size: mime");
                return NULL;
            }
            mime->setTo((const char *)&data[consumed]);
            consumed += mimeLen
```

发现了没，这个AlbumArt的存放位置就是在PIC或者APIC这一帧中，这具体怎么存放的我也不知道，这个和它的编码方式相关，有兴趣的同学自己去了解一下。

10 getAlbumArtOutputUri

以上就是获取完了，现在就到了做相应处理了，首先会调用getAlbumArtOutputUri，这个方法需要说明一下参数，这些参数都是在makeThumbAsync时new的对象的时侯赋的，现在这地方在从ThumbData的实参中拿出来，关键是那个Uri是什么内容很重要，再赋值的时候是直接指定的，用的是sAlbumArtBaseUri（"content:\media\external\audio\albumart"），大家应该每时每刻都注意参数uri的内容，MediaProvider部分始终是围绕Uri来做处理的，所以这部分很重要，一定要理解uri的作用以及内容。下面就是函数具体内容：

```
    Uri getAlbumArtOutputUri(DatabaseHelper helper, SQLiteDatabase db, long album_id, Uri a
        Uri out = null;
        // TODO: this could be done more efficiently with a call to db.replace(), which
        // replaces or inserts as needed, making it unnecessary to query() first.
        if (albumart_uri != null) {
            Cursor c = query(albumart_uri, new String [] { MediaStore.MediaColumns.DATA },
                    null, null, null);
            try {
                if (c != null && c.moveToFirst()) {
                    String albumart_path = c.getString(0);
                    if (ensureFileExists(albumart_uri, albumart_path)) {
                        out = albumart_uri;
                    }
                } else {
                    albumart_uri = null;
                }
            } finally {
                IoUtils.closeQuietly(c);
            }
        }
        if (albumart_uri == null){
            ContentValues initialValues = new ContentValues();
            initialValues.put("album_id", album_id);
            try {
                ContentValues values = ensureFile(false, initialValues, "", ALBUM_THUMB_FOL
                helper.mNumInserts++;
                long rowId = db.insert("album_art", MediaStore.MediaColumns.DATA, values);
                if (rowId > 0) {
                    out = ContentUris.withAppendedId(ALBUMART_URI, rowId);
                    // ensure the parent directory exists
                    String albumart_path = values.getAsString(MediaStore.MediaColumns.DATA)
                    ensureFileExists(out, albumart_path);
                }
            } catch (IllegalStateException ex) {
                Log.e(TAG, "error creating album thumb file");
            }
        }
        return out;
```

不知道大家发现没有这地方有一次对Uri做了处理，先慢慢分析再说，这部分做了很多事，查询，然后生成文件，在给数据库插数据，干的事挺多的，有个ensureFile方法和db.insert方法，ensureFile方法之前在获取image的Thumbnail的时候就展示过，这里就不看了，只是传进去的数据不一样，存储的文件夹是指定好的，ALBUM_THUMB_FOLDER（Android/data/com.android.providers.media/albumthumbs）；这个目录建立在mExternalStoragePaths[0]之后的，所以正常就在/storage/emulated/0/Android/data/com.android.providers.media/albumthumbs这个目录下。现在文件是造好了，数据也插了，后面当然就是给文件填充数据了，这部分在下面。然后回过头在看看这个函数，这个函数的返回值

是Uri，返回的是out，这个Uri不在插入数据库完成之后，他会对这个uri在一次做处理，赋值为ALBUMART_URI（"content:\media\external\audio\albumart"），所以呢，在处理完之后后面方法用到的这个函数的返回值的就是ALBUMART_URI。理解一个函数的功能最开始就是要了解它的参数以及返回值，通过看它的参数以及返回值我们能对这个方法了解个大概。下面继续分析到了下面的方法。

11 writeAlbumArt

上面就是获取的流程，现在看获取到这个字节数组之后干的事，这个跑到ContentResolver中去了。先看writeAlbumArt这个方法，

```
private void writeAlbumArt(
            boolean need_to_recompress, Uri out, byte[] compressed, Bitmap bm) throws IOExc
        OutputStream outstream = null;
        // Clear calling identity as we may be handling an IPC.
        final long identity = Binder.clearCallingIdentity();
        try {
            outstream = getContext().getContentResolver().openOutputStream(out);

            if (!need_to_recompress) {
                // No need to recompress here, just write out the original
                // compressed data here.
                outstream.write(compressed);
            } else {
                if (!bm.compress(Bitmap.CompressFormat.JPEG, 85, outstream)) {
                    throw new IOException("failed to compress bitmap");
                }
            }
        } finally {
            Binder.restoreCallingIdentity(identity);
            IoUtils.closeQuietly(outstream);
        }
```

很明显，获取了一个ContentResolver对象，然后调用openOutputStream方法，这地方不详细说了，就是ContentResolver中的方法去怎么怎么实现的，中间过程不说了，最后后调到ContentProvider的openFile中，实现当然又到MediaProvider中去了。

12 openFile

这个方法的具体实现是在MediaProvider中，但是中间的调用过程还是很复杂的，是ContentProvier和ContentResolver的交互，在MediaProvider中最后去最终实现。 这里就只看MediaProvider中得openFile（）

```
public ParcelFileDescriptor openFile(Uri uri, String mode)
            throws FileNotFoundException {

        uri = safeUncanonicalize(uri);
        ParcelFileDescriptor pfd = null;

        if (URI_MATCHER.match(uri) == AUDIO_ALBUMART_FILE_ID) {
            // get album art for the specified media file
            DatabaseHelper database = getDatabaseForUri(uri);
            if (database == null) {
                throw new IllegalStateException("Couldn't open database for " + uri);
            }
            SQLiteDatabase db = database.getReadableDatabase();
            if (db == null) {
                throw new IllegalStateException("Couldn't open database for " + uri);
            }
            SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
            int songid = Integer.parseInt(uri.getPathSegments().get(3));
            qb.setTables("audio_meta");
            qb.appendWhere("_id=" + songid);
            Cursor c = qb.query(db,
                    new String [] {
                        MediaStore.Audio.Media.DATA,
                        MediaStore.Audio.Media.ALBUM_ID },
                    null, null, null, null, null);
            try {
                if (c.moveToFirst()) {
                    String audiopath = c.getString(0);
                    int albumid = c.getInt(1);
                    // Try to get existing album art for this album first, which
                    // could possibly have been obtained from a different file.
                    // If that fails, try to get it from this specific file.
```

```
                    Uri newUri = ContentUris.withAppendedId(ALBUMART_URI, albumid);
                    try {
                        pfd = openFileAndEnforcePathPermissionsHelper(newUri, mode);
                    } catch (FileNotFoundException ex) {
                        // That didn't work, now try to get it from the specific file
                        pfd = getThumb(database, db, audiopath, albumid, null);
                    }
                }
            } finally {
                IoUtils.closeQuietly(c);
            }
            return pfd;
        }
```

根据上面得分析这里最后传进来得Uri是以albumart结尾得Audio_Uri,因此走上面得case，代码不多，但是挺复杂得，一步步分析一下，还有上面传进来得mode是r，好了言归正传分析代码，根据uri获取数据库，后面又new一个SQLiteQueryBuilder对象这个对象是为了方便生成sql语句而创建得，然后query歌曲得路径以及album_id，然后通过查到得album_id以及用ALBUMART_URI(content:\media\external\audio\albumart)去调用openFileAndEnforcePathPermissionsHelper（）方法，参数就是uri和album_id,去这个方法中去看看。

13 openFileAndEnforcePathPermissionsHelper()

```
    private ParcelFileDescriptor openFileAndEnforcePathPermissionsHelper(Uri uri, String mo
            throws FileNotFoundException {
        final int modeBits = ParcelFileDescriptor.parseMode(mode);

        File file = queryForDataFile(uri);

        checkAccess(uri, file, modeBits);

        // Bypass emulation layer when file is opened for reading, but only
        // when opening read-only and we have an exact match.
        if (modeBits == MODE_READ_ONLY) {
            file = Environment.maybeTranslateEmulatedPathToInternal(file);
        }

        return ParcelFileDescriptor.open(file, modeBits);
```

代码不复杂，跑到ParcelFileDescriptor中去了，首先通过调用queryForDataFile方法去获取一个file，这个方法代码就不放了，很简单，就是根据uri去query到一个路径，然后在根据这个路径去获取到一个file。后面还有一个checkAccess，这个其实我也没看懂在干啥，大概看看，其实也就是在检查Uri权限。继续，返回了一个ParcelFileDescriptor的open方法，获取一个ParcelFileDescriptor对象。

到这里就得回去writeAlbumArt中，在获取到一个outputstream，在这里判断一下是否需要压缩，不需要就放到bitmap对象里面。

14 总结

以上部分就是具体的歌曲artwork的获取方式以及大致流程，大部分都是在MediaProvider中完成，这里面知识挺复杂的，想要了解的同学仔细看代码，能学到不少东西。

获取albumart的特殊方法

虽然在MediaScanner中会对artwork做处理，但是android也提供了一个临时的方法去获取它的artwork只是这个方式只能获取到，并不会对数据库又影响。 `MediametadataRetriever.java` 这个对象是android提供的一个可以获取单个媒体文件的一些经常用到的信息的方法，不仅仅是audio文件，video文件的一些信息这里也能获取到，简单介绍一下，感兴趣同学可以去看看，对与artwork的获取，这里也提供了一个接口getEmbeddedPicture（）

```
  public byte[] getEmbeddedPicture() {
         return getEmbeddedPicture(EMBEDDED_PICTURE_TYPE_ANY);
     }

  private native byte[] getEmbeddedPicture(int pictureType);
```

很明显最终会回到native层中去，找这个文件对应的jni函数，根据MediametadataRetriever的包可以定位到android_media_mediaMetadataRetriever.cpp文件，这个就是MediametadataRetriever的jni交互类，找到getEmbeddedPicture的实现类去它注册的地方看看就知道，就是android_media_MediaMetadataRetriever_getEmbeddedPicture

```
static jbyteArray android_media_MediaMetadataRetriever_getEmbeddedPicture(
        JNIEnv *env, jobject thiz, jint pictureType)
{
```

```
    ALOGV("getEmbeddedPicture: %d", pictureType);
    MediaMetadataRetriever* retriever = getRetriever(env, thiz);
    if (retriever == 0) {
        jniThrowException(env, "java/lang/IllegalStateException", "No retriever available")
        return NULL;
    }
    MediaAlbumArt* mediaAlbumArt = NULL;

    // FIXME:
    // Use pictureType to retrieve the intended embedded picture and also change
    // the method name to getEmbeddedPicture().
    sp<IMemory> albumArtMemory = retriever->extractAlbumArt();
```

发现了没，到这里大家应该也就明白了，这里又是调到libstagefright框架中去了，实现方法还是extractAlbumArt，因此如果歌曲文件存在albumArt这两个方法获取到的文件是一样的。在不向下追踪了。还有人可能不明白为什么上面在java层调用getEmbeddedPicture的时候没有传进参数是怎么获取像因为件的artwork的这里呢其实在new一个MediaMetadataRetriever对象的时候是需要在之后调用它的setDataSource这个方法去传进来资源，然后才能调用别的方法去获取相应的数据。因此在用的时候一定要注意。

全章总结

对与MediaScanner来说，大体流程其实也不太复杂，就是里面处理琐碎的事比较复杂，就如上面得thumbnail获取，涉及到很多知识，想要彻底搞懂得好好阅读源码。

以上就是我对于缩略图这块的理解，大家如果有兴趣可以读一下，对于理解大致流程是有帮助的，有错，请提出，大家一起学习