mediascanner & mediaprovider

Last edited by **caoquanli** 1 month ago

在每次开机以及一些特定活动下，系统都会对内存进行扫描。扫描是对内存中的一些数据进行分类，为一些应用提供便利。这里主要讲是对媒体文件扫描（主要是歌曲）。

Android扫描流程是分为很多步的，其中相关代码在源码目录下的\packages\providers\MediaProvider下面的几个源码。 扫描流程比较复杂涉及的源码比较多主要有MediaProvider.java（MP），MediaScannerReceiver.java（MSR），MediaScannerService.java（MSS），MediaScanner.java（MS）。

Block图

MediaScannerBlock图2

Java层时序图

MedaiScannerjava层

C++层时序图

mediascannerC++层

首先我们分析一下源码目录下的/packages/providers/MediaProviders目录，从这里我们可以发现它实际就是一个apk，先看看.xml文件，看看注册ingrain，发现他会创建一个android.process.media进程，后面还有一个receiver，接收intent，当然也有intentfilter，从这里我们就能发现它接收广播的种类。这个模块在android四大组建中充分运用了三个，broadcastReceiver，contentProvider，service，分别对应MediaScannerReceiver，MediaProvider，MediaScannerService，再就是这部分充分运用jni知识，调用native层的对象，以及用libmedia/libstagefright库。

1 接收活动

MediaScanner主线程一直在系统中等待执行，MSR是一个入口类，有一个onReceive函数接收Intent，在接受到几个特定Intent（ACTION_BOOT_COMPLETED，ACTION_SCAN_SPECIFY_PATH，EXTERNAL_UDISK_VOLUME）后会执行scan函数，这个函数有两类，一类是scan，对文件夹进行扫描，还有一个scanFile，对文件进行扫描，然后会开启一个service。

```
    new Intent(context, MediaScannerService.class).putExtras(args));
```

下面进入这个服务。

2 调用MSS

MSS，是MediaScanner中的具体服务部分，启动服务之后，都是会首先执行OnCreat，在这个函数中会首先申请一个电源锁，因为在扫描过程中用的时间会比较久所以申请电源锁防止cpu睡眠。然后会启动一个工作线程线程，这个线程就是跑在主线程中的工作线程，执行部分都在这里了，执行run函数，线程跑起来，当中会new一个ServiceHandler（这个也就是一个handler），这个handler用的looper，就是主线程的looper，等待消息。执行onStartcommand函数，这个函数中首先有一个处理框，是否执行完ServiceHandler，如果没有等待0.1秒，然后就是创建消息以及给消息队列中放消息。消息发出去了，看看handleMassage，这个首先就是对当前的消息在binder进行处理，然后获得这个文件的路径信息，或者是把上面扫描的媒体文件进行解析，这个函数主要实现的是获取文件路径判断路径是否与数据库中的文件路径一致，然后会传进来两个参数。后面是对MP的一些调用对传进来的数据进行修改，主要是判断文件路径和卷名是否与是数据库中的一样，如果一样则用数据库中的路径，如果不一样进行修改。在消息处理过程中会调用scan函数。

3 执行scan函数

这个scan函数，首先发送一个扫描开始广播,然后根据volume查找数据库，如果数据库存在，打开数据库，获得这个目录，要是不一致，就是新插进来的这地方会去去MediaProvider中去进行insert，这个insert对应于查找以及新建数据库。 数据就是在这个地方建立的，之后在扫描结束的时候，就会给数据擦汗如数据，或者是更新数据，要操作数据苦必须现家里数据库，下面看看建立数据库的流程。

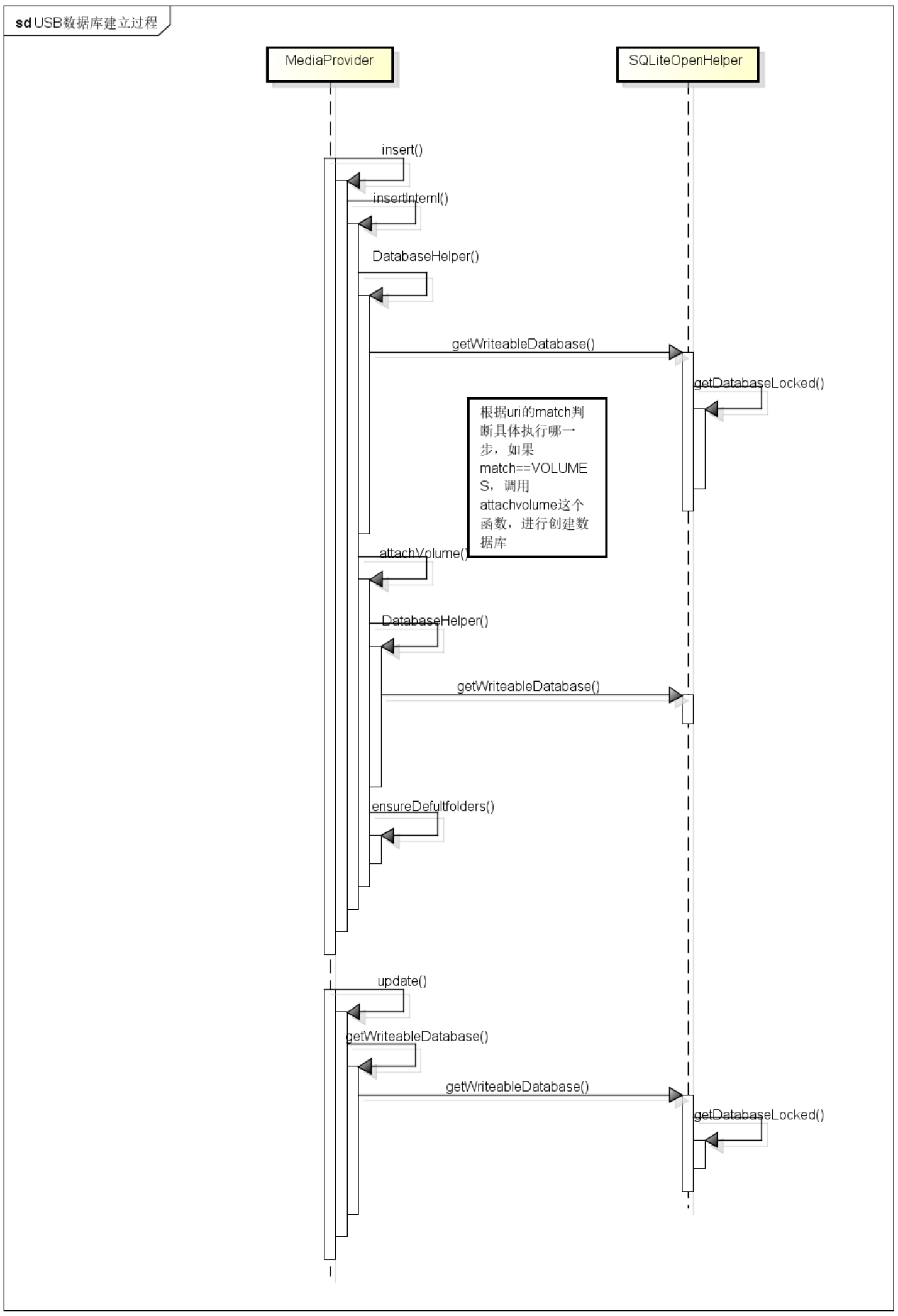
获取数据库

入口： `openDatabase(volumeName);`

```
private void openDatabase(String volumeName) {
       try {
           ContentValues values = new ContentValues();
```

```
            values.put("name", volumeName);
            getContentResolver().insert(Uri.parse("content://media/"), values);
        } catch (IllegalArgumentException ex) {
            Log.w(TAG, "failed to open media database");
        }
    }
```

这个会调用insert会最终实现在Media Provider中，然后调用它的insert，因为uri是"content:\media"，所以在匹配uri的时候会匹配到VOLUMES，走的流程就是下面时序图中的流程，调用attachVolume创建数据库。



这里大概说一下数据库创建，先看父类的SqliteDatabasHelper，首先getWritableDatabase，这里需要提一下的是，Android源码里面提供了两个方法，一个是这个getWritableDatabase，还有是一个getReadableDatabase,这是调用数据库操作的两种方式，前一个可读可写，后面的那个只能可读，不能更改数据，然后会调用到getDatabaseLocked，这个方法是实现数据库创建的根本方法，这里面会调用到两个方法，一个是onCreate和onUpgrade方法，这两个方法的作用就是建表。到这里数据库库就创建完了。

继续执行，发送扫描开始广播，调用scanDirectories函数了，这个函数之前是new了一个MediaScanner对象的，所以理所当然下面就是进入到MediaScanner中了，这部处理完之后，也就意味着扫描完成了，之后会发送一个结束广播。这个过程主要的工作其实是启动一个实例化一个对象，进行扫描

```
try (MediaScanner scanner = new MediaScanner(this, volumeName)) {
          scanner.scanDirectories(directories);
}
```

通过这部操作，启动扫描，调用MediaScanner对象的scanDirectories方法。

4 进入MS

在MS的构造函数中，我们会发现在代码开始就有一个加载JNI动态库的操作

```
static {
        System.loadLibrary("media_jni");
        native_init();
    }
```

说明MediaScanner是与Native层有交互，里面有很多带native关键字的函数，这些函数的最终实现都是在native中，下面会继续分析。在MediaScanner中会重新对一些参数进行赋值。然后执行scanDirectories，它首先会perscan，会发送一个插入广播（ACTION_SCAN_INSERT），打开MediaProvider。对数据库进行操作。这里面如果是文件他会在够五百个之后才会操作。当然这里面有个参数是mclient这个对象是调用native层的。

5 执行prescan

进行预扫描操作，根据传进来的路径，将数据库中原有的媒体文件放到一个数据结构中临时存储起来。根据从数据库中拿出来的数据，然后判断这个路径中数据是否存在，如果不存在，就执行delete，如果存在放在特殊的数据结构里，记录版本，这里有个更改时间，完了这个更改时间会用来判断需不需要update，然后跟据传进来的prescanFiles，判断扫描的是的文件夹还是文件。如果是文件会给这个文件给一个具体的路径，如果不是文件，他会在上一步中继续操作。归根到底，这部分就是一个查询操作。

6 进入JNI层

继续执行，会用到一个native层的方法，processDirectory，这个方法已经在这个类中用native关键字声明了，通过调用进入JNI层，native与java层的交互主要有两种方式（一种是直接调用，一种是加native关键字）。进入android_media_MediaScanner.cpp，通过对后面函数的声明以及注册，我们可以找到processDirectory为android_media_MediaScanner_processDirectory函数，在这部分定义了一个client，

```
MyMediaScannerClient myClient(env, client)
```

并且将java层的client传了进去，这个client后面是会用到的，后面的函数会回调java层的函数。这个函数的具体定义在MeidaScanner.cpp中，进入MediaScanner。

7 进入MS.cpp

先去它的.h文件很容易就能理解它是怎么调用， 看函数，我们很容易就可以确定是这个函数，

```
MediaScanResult MediaScanner::ProcessDirectory
```

上面函数的作用就是调用doProcessDirectory函数，

```
MediaScanResult MediaScanner::doProcessDirectory
```

定义了一些参数，path扫描文件夹路径然后以'/'结尾，pathRemaining为路径长度与路径最大长度的差值，防止扫描路径超出范围，client是是STEP6中实例化的MyMediaScannerClient对象，后面两个参数是一些异常处理不用关心。它完成的是遍历文件夹并找到有相应extensions 里面后缀的文件fileMatchesExtension(path, extensions)，如果文件大小大于0就调用client.scanFile(path, statbuf.st_mtime, statbuf.st_size);来进行文件读取扫描，这里才会读文件的实际内容。这里面调用的函数很多的，这里就不放代码了，因为这个整个类就是处理文件夹的，这里只是简单的讲一下它的功能。

8 调用JNI层的scanFile

这里用到了Myclient，在回调Java层的client，这部分主要就是获取几个参数，路径，最后一次修改时间，文件大小，其中这个文件的最后一次修改时间是在上一次修改与现在的时间差。

9 调用MS的scanFile

传进来路径，文件最后一次修改以及文件大小等信息。然后调用doScanFile，这里面有几个重要的函数（beginFile，processfile，endfile），在beginFile中构建了一个FileCacheEntry对象，存储文件的一些基本信息，并且将其放在一个哈希map中。根据此文件是否需要来判定是否processFile，这个方法得也是一个native方法，进入native层。

10 又一次进入native层

```
android_media_MediaScanner_processFile
```

通过判断是上面的函数，通过一些处理，上面的的函数最终调用如下

```
MediaScanResult result = mp->processFile(pathStr, mimeTypeStr, myClient);
```

这个函数实现在stagefrightMediaScanner.java。

11 进入StagefrightMediaScanner.cpp

他是stagefright中的一部分，stagefright是android系统media处理的框架，它是ProcessFile的最终实现。这一部分的具体内容就是媒体扫描。这个processFile经过一系列操作，最终会执行到解析器，这部分由很多相关媒体的解析器，解析完之后会通过一个keyMap，通过handleStringTag将信息传回java层。到这部分媒体扫描就基本完成了，后面的部分就是将解析出来的MediaInfo放入数据库中。这部分详细调用参见C++层的时序图。

12 继续回到MS

在回到java层，doScanFile，执行endfile，这个方法的最终实现就是将扫描到的数据存入数据库，这里由两个方法，insert和update，这两个方法，大家都能理解，如果是新文件，肯定是insert，如果是数据库中已经存在，肯定就是update了，这个新旧文件的判断需要用到上面prescan扫描到的数据，通过一个参数rowId来判断，然后就是数据库的操作了

13 进入MediaProvider

这一步，就是最终数据库的操作了，在MediaProvider创建开始，就会执行onCreate方法，这个方法里面有一个attchVolume这个方法，这个方法的具体作用这里提一下，字面意思连接数据库，在一开是调用数据的时候，就会创建好，或者说是链接好，这里就会涉及到数据库的对应问题，在正常情况下，只有两个，一个是internal.db，一个是external.db 这两个数据库前一个对应得是系统内部，一个是外部存储，一般是sd卡，但是在插入USB或者别的存储外设的时候，系统也会收到广播扫描，这个时候，数数据库的对应调用了VolumeInfo里面的VolumeID方法，通过这个ID能够正确识别和对应数据库，关于attachVolume部分代码这里就不列出了。但是有关USB识别以及链接和创建都是在这部分完成，感兴趣的可以看看源码。

继续前面，在endFile中用到两个方法，insert和update，这两个方法就是操作数据库的，功能前面已经提到了。最终实现是在MediaProvider中，具具体的处理也不仅仅只是擦汗数据库，在插入数据库之前做的事很复杂，比如说处理audio的artwork以及在数据库中的各个id，很复杂，这里这是简单讲一下具体流程，详细在以后的章节说明。

注，这篇关于MediaScanner的文章，是本人参考网上的一些知识，以及阅读源码的理解，错误之处应该不少。还有对于文章涉及的代码这里都没有展示，不过代码都是Android源码，感兴趣的同学可以自己下载研究，具体目录文章开始已经列出。关于JNI知识可以看看深入理解android卷I，讲解的已经很清楚了。