(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 108124183 B
(45)授权公告日 2020.06.19

(21)申请号 201611070036.2

(22)申请日 2016.11.29

(65)同一申请的已公布的文献号
申请公布号 CN 108124183 A

(43)申请公布日 2018.06.05

(73)专利权人 达升企业股份有限公司
地址 中国台湾新北市汐止区大同路1段263号4楼

(72)发明人 吴昌育 许顺翔 许耀中 吴启宏

(74)专利代理机构 深圳中一专利商标事务所 44237
代理人 阳开亮

(51)Int.Cl.
H04N 21/43(2011.01)
H04N 21/439(2011.01)
H04N 21/44(2011.01)
H04N 21/4402(2011.01)
H04N 21/8547(2011.01)
H04L 29/06(2006.01)

审查员 周立秋

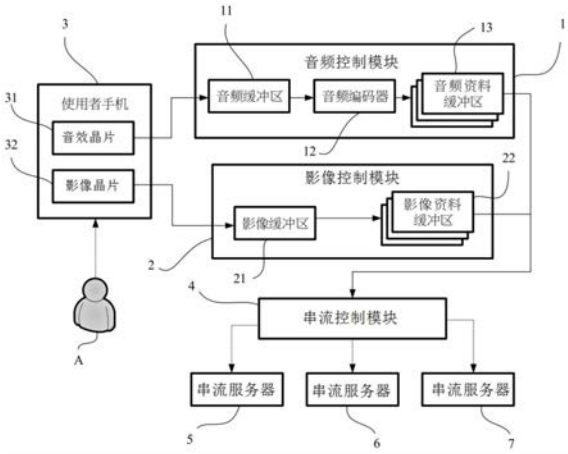
权利要求书2页 说明书16页 附图3页

(54)发明名称

以同步获取影音以进行一对多影音串流的方法

(57)摘要

本发明为一种一对多影音串流方法,其主要包括:从行动装置的影像与声音芯片组分别获取影像及声音信息源,并重新解析相关讯号源数据,然后对影像及声音数据的时间戳校正,使输出的影像与声音同步化,从而解决串流译码端服务器因声音或影像时间轴不同步所造成的兼容性问题。本发明通过行动装置进行影像与声音获取时,将无需通过ffmpeg开源码套件进行格式转换,可大幅降低手机中央处理器的运算量及耗电量,使行动装置影像与声音可以同时对多个影音串流服务器进行实时影音传送。



CN 108124183 B

1.一种同步获取影音以进行一对多影音串流的方法,其特征在于,所述方法包含以下步骤:

通过音频控制模块执行音频同步程序,用经过硬件抽象层自行动装置的音效芯片获取经编码的音频编码数据,并取得对应编码程序的音频编码参数,所述音频控制模块将所述音频编码数据针对相异的多个串流服务器暂存;

通过所述音频控制模块执行音频校正程序,对暂存的所述音频编码数据依据音频编码器的固定输出帧率进行时间戳校正;

通过影像控制模块执行影像同步程序,用硬件抽象层自行动装置的影像芯片直接获取经编码的影像编码数据,并取得对应编码程序的影像编码参数,所述影像控制模块将所述影像编码数据针对相异的多个串流服务器暂存;

通过所述影像控制模块执行影像校正程序,对暂存的所述音频编码数据依据所述串流服务器的实际接收时间进行时间戳校正;

通过串流控制模块建立需先于影音串流被传送至所述串流服务器的影像及音频描述参数;

通过所述串流控制模块针对相异的所述串流服务器认证联机;

所述串流控制模块选择性地通过信息封装格式将所述音频编码参数加入经时间戳校正后的所述音频编码数据中以封装产生多个音频串流封包;

所述串流控制模块依线程解析所述影像编码数据内容,分析出时间戳及资料大小,并根据上次影像数据封包的时间戳和本次影像数据封包的时间戳相减得出差值,然后根据取得的这些信息,所述串流控制模块选择性地建构对应影像串流封包格式的数据文件头,并将分析出来的时间戳差值、数据大小等填入封包的数据文件头中,再将所述影像数据置于所述数据文件头之后,以完成所述影像串流封包的建构,再根据一次能传输的最大信息封包大小进行切割所述影像串流封包,产生多个所述影像串流封包;以及

所述串流控制模块先将所述影像及音频描述参数传输至相异的多个所述串流服务器,再将针对所述串流服务器封装的所述音频串流封包以及所述影像串流封包,依序同时传输至相异的多个所述串流服务器,以实施对所述串流服务器进行一对多影音串流。

2.如权利要求1所述的同步获取影音以进行一对多影音串流的方法,其特征在于,所述音频同步程序至少包含以下步骤:

所述音频控制模块自行动装置的音效芯片直接获取原始音频数据,并将获取的原始音频数据输入音频缓冲区;

所述音频控制模块套用默认的音频关联参数对音频编码器进行设定;

所述音频控制模块依据音频编码接收线程,通过设定后的音频编码器的硬件抽象层,用自输出队列中获取存于音频缓冲区中的原始音频数据经音频编码器编码后产生的所述音频编码数据,并将所述音频编码数据分别传送到针对多个串流服务器所设置的多个音频数据缓冲区当中;以及

所述音频控制模块记录所述音频编码数据以及所述音频编码数据经音频编码器编码时套用的所述音频编码参数。

3.如权利要求1所述的同步获取影音以进行一对多影音串流的方法,其特征在于,所述影像同步程序至少包含以下步骤:

所述影像控制模块套用默认的影像关联参数对影像编码器进行设定；

所述影像控制模块依据影像编码接收线程，通过行动装置影像芯片的硬件抽象层从输出队列中直接获取经编码的所述影像编码数据；

所述影像控制模块将所述影像编码数据输入影像缓冲区；

所述影像控制模块记录所述影像编码数据经影像编码器编码时套用的所述影像编码参数；以及

所述影像控制模块将影像缓冲区中的所述影像编码数据分别传送到针对多个串流服务器所设置的多个影像数据缓冲区中。

4.如权利要求1所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述信息封装格式至少包含以下种类：

FMT0数据文件头，其包含有封包的时间戳、信息长度、信息种类ID、信息串流ID等部分，表示封包为串流中的一个独立信息；

FMT1资料文件头，其用以表示封包为同一串流中的信息；

FMT2资料文件头，其用以表示封包为同一串流中的信息，且数据内容类及大小与先前信息相同；以及

FMT3数据文件头，其用以表示封包传送被拆成多笔分送的同一个信息，后续的封包可沿用第一笔信息的格式译码。

5.如权利要求2所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述音频编码接收线程的优先度高于一般线程，以使所述音频控制模块通过所述音频编码接收线程连续地将原始音频数据输入音频缓冲区。

6.如权利要求1至3项中任一项所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述音频关联参数包含使用于编码的音频格式、音频取样频率、音频信道数量或音频编码比特率的其中之一的或两者以上的组合。

7.如权利要求1至3项中任一项所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述音频编码参数为一种音频专用配置ASC参数，其包含使用于编码的音频格式、音频取样频率或音频信道数量的其中之一或两者以上的组合。

8.如权利要求1至3项中任一项所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述影像关联参数为进行影像编码所需的视频编码专家组VCEG影像编码标准参数，其包含：影像宽度、影像高度、影像编码帧率FPS、影像的图像群组GOP、影像编码比特率或影像编码格式的其中之一或两者以上的组合。

9.如权利要求1至3项中任一项所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述影像编码参数为序列参数集SPS和图像参数集PPS参数，包含使用于编码的影像格式、编码的影像格式级别、编码影像的长宽，去区块滤波器种类的其中之一或两者以上的组合。

10.如权利要求1至3项中任一项所述的同步获取影音以进行一对多影音串流的方法，其特征在于，所述影像及音频描述参数包含影像关联参数、影像编码参数、音频关联参数以及音频编码参数。

以同步获取影音以进行一对多影音串流的方法

技术领域

[0001] 本发明属于影音串流处理的技术领域,特别涉及一种对手机影像与声音硬件信息源进行同步化获取,进而达成以一个影音信息源同时对多个相异影音服务器传递串流的方法。

背景技术

[0002] 随着科技的日新月异以及网络时代的信息需求,关于多媒体影音与通讯传输的技术蓬勃发展,几年前仅有少数选择的网络实时影音传递或直播服务,现已成为非常普遍的一种因特网应用,当然,此种服务所需的技术相应地也成为提供因特网业者的兵家必争之地。

[0003] 最传统的影音获取技术,是通过体积庞大携带不便的摄影机,在预先决定的拍摄地点进行特定目标的拍摄,接着在返回工作室,将拍摄下的档案通过计算机整理压缩,再上传至对应的网络媒体平台令用户可以在网络上阅览摄影画面,此种拍摄的过程十分繁复,且专业摄影设备价格高昂,一般的使用者难以负担。

[0004] 但随着科技演进,现今的硬件技术与上述最传统的影音获取技术相比,两者已不可同日而语,近年蔚为风潮的智能型手机或平板计算机等等一般用户可以轻易携带使用的行动装置,其影音摄录功能大多已能够获取清晰度足够且流畅的内容,辅以随处可得的无线网络,令用户通过这些行动装置获取的影音内容不须繁复手续即可以被上传至媒体服务器或平台,故平台的用户可以更快的速度获得实时的影音内容,与传统的技术相较之下,显然影音技术已产生重大突破。

[0005] 然而,上述通过一般用户的行动装置进行影音串流的技术,仍然受限于诸多状况,并不完善;首先,目前智能型手机或平板计算机等行动装置,为通过网络与一组实时传送信息通讯协议RTMP服务器传递实时影音串流,为了达到上述影音串流功能,大多数的行动装置皆通过安装一组以ffmpeg开源码核心所开发的影音串流应用程序,这种影音串流应用程序会先取得装置上影像和声音的原始数据,再把这些影像和声音转换压缩为数字型态,如此才可使获取的影像与声音数据符合RTMP服务器所需的数字流格式。

[0006] 但上述影音串流技术需经过数量庞大的影音数据压缩以及转换,故其对处理装置的中央处理器和内存造成的负载极大,尤其因为行动装置的中央处理器和内存功能更加受限,因此对负载直播的影音数据处理考验更大,另外,上述开发应用程序的ffmpeg开发工具包除了有程序代码数量相对庞大的问题外,开发工具包本身更存在多种复杂功能,故其在执行状态当下,一般都会连带在装置上启动若干需消耗大量效能的项目,造成行动装置的中央处理器负载更加剧,进而会导致行动装置耗电量飙升。

[0007] 而由于了解采用ffmpeg套件会导致上述问题,长久以来,若通过上述技术实施影音串流,同一时间内仅能用一个行动装置对一个RTMP服务器进行影音串流,若需要上传影音串流至另一个RTMP服务器,则必须反复进行同一步骤,显然,在此种实施流程下,无法使多个RTMP服务器之间的影音串流数据实时同步,由此可以了解,若要通过前述的技术手段

来实施用一个行动装置同时对多个RTMP服务器的多任务串流，是无法实现的。

[0008] 综上可知，已知技术中，以行动装置等进行实时同步影音串流播放的技术，仍存在着若干问题，将有待本领域中的研究者进一步的研发与创新。

[0009] 为此，本案发明人构思解决上述问题的方法，经过慎密的研究计算，以及长期的研发与实验后，终能完成本发明，即一种同步获取影音以进行一对多影音串流的方法。

发明内容

[0010] 本发明的主要目的，在于提供种同步获取影音以进行一对多影音串流的方法，可以令使用者仅通过一个行动装置，即可实现对多个RTMP服务器进行影音数据的同时多任务串流，使用者仅需上传一次影音数据，其他用户即可在不同的多媒体播放平台实时同步阅听到相同的影音内容。

[0011] 本发明主要提供了一种采影音信息源分离式的线程处理架构，其实施方法是用音频控制模块和影像控制模块，通过智能型手机、平板计算机等行动装置的影音硬件的硬件抽象层(Hardware Abstraction Layer)，分别获取影像数据(H.26x)和未经编码的音频数据。

[0012] 而由于不同厂商所出产的行动装置所采用的影音芯片模块不尽相同，故本发明的音频控制模块和影像控制模块需针对其各自获取的影音数据中含有的不规则时间戳(Timestamp)进行校正，以对应不同的芯片，将不同时间戳的影音数据整合成时间轴完全同步的影音数据。

[0013] 其次，为了实现同时分送影像和音频信息至不同串流服务器的目的，本发明需要经过音频控制模块来对音频编码器做特别的设定，然而，即便需要向多组相异的串流服务器进行发送，本发明的方法仍仅需针对一组音频编码器做初始化设定，以便合理地减少整体流程的工作负担。

[0014] 然后，为了达成节省中央处理器效能，以及内存使用最大化这两个目的，本发明的影像控制模块和音频控制模块，需要针对相异的串流服务器但利用相同的硬件编码器进行编码，但其在针对不同的串流服务器传输串流之前，可先指定多个彼此之间独立的影音缓冲区来储存资料，一方面可以避免过多的编码器同时运作而拖垮系统效能，另一方面可以达成在传送不同串流资料封包时，亦不会因为其中一个服务器异常，而使封包的传递产生相互影响。

[0015] 而为了达成本发明的目的，本发明的方法根据使用者指定的相异的服务器网址以及各自的密钥，以将影像和音频数据分别同时传输至不同的串流服务器进行播放，而为了达到穿越防火墙来分送数据的目的，也可以启用不同端口传输数据，本发明为了避免数据在传输时因壅塞所导致的丢失状况，通过串流控制模块在传输起始时针对不同的串流服务器设置个别的Socket缓冲区，并且依据不同串流服务器需求启动独立的线程，以此进行封包的传递，以达成数据吞吐量的最大化。

[0016] 最后，本发明的音频控制模块以及影像控制模块在接收到经编码后的影像或声音数据时，会进行数据分析，再根据数据内容判断格式后，将信息加入文件头以打包成统一的流格式，再交由串流控制模块，启动不同串流服务器所专属的传递封包线程，将封包放入预先建立的传送缓冲队列中，等待装置对多服务器的同步传送程序开始。

[0017] 详细来说，本发明的同步获取影音以进行一对多影音串流的方法，主要包含下列步骤：

[0018] 音频获取部分：首先，通过音频控制模块执行音频同步程序，用硬件抽象层自行动装置的音效芯片获取经编码的音频编码数据，并取得对应编码程序的音频编码参数，所述音频控制模块并将所述音频编码数据针对相异的多个串流服务器暂存；再通过所述音频控制模块执行音频校正程序，对暂存的所述音频编码数据依据音频编码器的固定输出帧率进行时间戳校正。

[0019] 影像获取部分：首先，通过影像控制模块执行影像同步程序，用硬件抽象层自行动装置的影像芯片直接获取经编码的影像编码数据，并取得对应编码程序的影像编码参数，所述影像控制模块并将所述影像编码数据针对相异的多个串流服务器暂存；再通过所述影像控制模块执行影像校正程序，对暂存的所述音频编码数据依据所述串流服务器的实际接收时间进行时间戳校正。

[0020] 上述音频获取部分和影像获取部分可同时进行。

[0021] 接着，本发明通过串流控制模块建立需先于影音串流被传送至所述串流服务器的影像及音频描述参数，并通过所述串流控制模块针对相异的所述串流服务器认证联机。

[0022] 所述串流控制模块选择性地通过信息封装格式将所述音频编码参数加入经时间戳校正后的所述音频编码数据中以封装产生多个音频串流封包；相对的，所述串流控制模块选择性地通过所述信息封装格式将所述影像编码参数加入经时间戳校正后的所述音频编码数据中以封装产生多个音频串流封包。

[0023] 最后，所述串流控制模块先将所述影像及音频描述参数传输至所述串流服务器，再将针对所述串流服务器封装的所述音频串流封包以及所述影像串流封包，依序同时传输至所述串流服务器，以实施对所述串流服务器进行一对多影音串流。

[0024] 综上可知，本发明为通过以上程序以及模块，用行动装置同步获取影音，来进行一对多影音串流的方法。

附图说明

[0025] 图1为本发明中同步获取影音以进行一对多影音串流的方法的模块示意图。

[0026] 图2为本发明中同步获取影音以进行一对多影音串流的方法的模块示意图。

[0027] 图3为本发明中同步获取影音以进行一对多影音串流的方法的模块示意图。

[0028] 图中：A：使用者；1：音频控制模块；2：影像控制模块；3：使用者手机；4：串流控制模块；5：串流服务器；6：串流服务器组；7：串流服务器；11：音频缓冲区；12：音频编码器；13：音频数据缓冲区；21：影像缓冲区；22：影像数据缓冲区；31：音效芯片；32：影像芯片；S01～S17：步骤流程。

具体实施方式

[0029] 为使本发明实施例的目的、技术方案和优点更加清楚，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例是本发明的部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域中的一般技术人员所能思轻易想到的所有其他实施例，皆属本发明保护的范围。

[0030] 请参阅图1，本发明提供的一种同步获取影音以进行一对多影音串流的方法通过图1中模块实施，说明概述如下：

[0031] 本发明在使用者A通过使用者手机3上传影音数据时，通过音频控制模块1从用户手机3中的音效芯片31的硬件抽象层获取未经编码的原始音频数据，并输入音频缓冲区11，接着，当获取的原始音频数据通过套用预先设定的音频编码器12被编码输出时，音频控制模块1自输出队列获取经编码后的音频编码数据，音频控制模块1整理音频编码数据并输入多个分别对应相异串流服务器的音频数据缓冲区13，音频控制模块1同时记录音频编码器12编码时所使用的对应音频编码参数。

[0032] 另外，与音频获取程序同时进行地，本发明通过影像控制模块2从用户手机3当中的影像芯片32的硬件抽象层输出队列，获取已经过编码的影像编码数据，并输入影像缓冲区，影像控制模块2对影像编码数据进行整理，记录编码时所使用的对应影像编码参数，并将影像编码数据输入多个分别对应相异串流服务器的影像数据缓冲区22。

[0033] 接着，在此实施例中，本发明通过串流控制模块4对三组相异的串流服务器5、串流服务器6、串流服务器7建立连结，以将音频控制模块1和影像控制模块2对应串流服务器5、串流服务器6、串流服务器7各自校正打包好的影音串流封包同步分送出去，以完成本发明的同步获取影音以进行一对多影音串流的方法。

[0034] 再请参照本发明的图2，为本发明同步获取影音以进行一对多影音串流的方法的步骤流程图，说明分述如下：

[0035] 步骤S01、开始步骤：

[0036] 本发明的方法，为了使行动装置的中央处理器(CPU)与图形处理器(GPU)在串流程序中能达成较佳的使用效率和平衡，须对行动装置硬件产生的影像及音频各自使用两个专属的线程来进行处理，分别为：编码接收线程，以及串流封装线程。

[0037] 其中，编码接收线程其自影音硬件的硬件抽象层获取数据，再存放于特定的数据缓冲区内的线程；而串流封装线程将数据自数据缓冲区中取出，再封装成流格式后输出，分别使用这两个专属线程处理影像和音频具有以下优点：(a)分属两个线程处理，可避免应用程序耗费过多时间等待获取硬件抽象层的数据，而导致无法及时将特定缓冲区内的数据封装送出的状况；(b)另一方面，当网络联机状况不佳时，应用程序会忙碌于传送封装后的串流数据，而此时向硬件抽象层获取数据的动作常会因此被延迟。

[0038] 当上述两种情况发生时，很可能导致影像或音频数据产生不连续现象，甚或是使应用程序崩溃，而本发明采用专属的线程分别处理影像和音频则可以避免影像和音频的间的相互影响所造成的不稳定与负载过重，并节省装置的硬件效能和维持操作系统稳定性。

[0039] 进行本步骤的程序代码范例如下所示：

```
            data_produce = new Thread(new Runnable() {
                @Override
                public void run() {
                    Log.i(TAG, clsName + " data_produce running");
                    //android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_URGENT_AUDIO);
                    while (true) {
                        Packet p = fillPacket();
                        if (p == null) {
                            Log.w(TAG, clsName + " data_produce exit");
                            break;
                        }

                        try {
                            dataQueue.put(p);
                        } catch (InterruptedException e) {
                            Log.e(TAG, clsName + " data_produce interrupted");
                            break;
                        }
                    }

                    Log.i(TAG, clsName + " data_produce exited.");
                }
            });
```

[0040]

```
public void run() {
    //android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_URGENT_AUDIO);
    Log.i(TAG, clsName + " data_consume running");

    while(true) {

        //try_delay();
        try {
            Packet p = dataQueue.take();
            if (p.data == null) {
                Log.w(TAG, clsName + " data_consume exit");
                break;
            }
            //calculate_fps();
            writefile(p.data, p.timestamp);
            sendPacket(p);
            //Thread.sleep(100);
        } catch (InterruptedException e) {
            Log.e(TAG, clsName + " data_consume interrupted");
            break;
        }
    }
```

[0041] 步骤S02、通过音频控制模块取得未经编码的原始音频数据：

[0042] 由于从声音获取方面来说，音频控制模块并不需要额外根据相异的串流服务器之间的需求来进行设定，故本发明的音频控制模块在整体程序启动时，仅需设定一次串流服务器的共享音频来源相关属性，其称作音频关联参数，参数内容可以包括：取样率(Sample Rate)、取样通道个数(Stereo/Mono)、获取音频的通道来源(例如：麦克风或其它装置)、音频数据的位数(8bit或16bit)等等，而音频信息若在编码程序中产生了丢失，会明显地造成阅听者在阅听时产生音频不连续的状况，故本发明的音频控制模块额外对音频信息设立数据缓冲区，用以先暂存所获取的音频资料；另外，为了确保获取到的音频数据能够不间断地提供给音频编码器进行处理，本发明进行获取音频资料的线程，其优先权将被设定高于其他的一般线程，旨在避免其它的线程抢断音频获取的工作，使操作系统在进行线程排程时，可以保证中央处理器提供较多时间给获取音频数据的工作。

[0043] 进行本步骤的程序代码范例如下所示：

```
android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_URGENT_AUDIO);
try {
    final int min_buffer_size = AudioRecord.getMinBufferSize(
            SAMPLE_RATE, CHANNEL_IN_STEREO,
            ENCODING_PCM_16BIT);
    int buffer_size = SAMPLES_PER_FRAME * FRAMES_PER_BUFFER;
    if (buffer_size < min_buffer_size)
        buffer_size = ((min_buffer_size / SAMPLES_PER_FRAME) + 1) * SAMPLES_PER_FRAME * 2;
```

[0044]

```
    AudioRecord audioRecord = null;
    try {
        audioRecord = new AudioRecord(
                MIC, SAMPLE_RATE,
                CHANNEL_IN_STEREO, ENCODING_PCM_16BIT, buffer_size);
        if (audioRecord.getState() != AudioRecord.STATE_INITIALIZED)
            audioRecord = null;
    } catch (final Exception e) {
        audioRecord = null;
    }
```

[0045] 步骤S03、对音频编码器进行音频关联参数设定：

[0046] 本发明的音频控制模块，将先设定音频编码器的音频关联参数，包括：音频编码器所使用的编码格式(Profile)、音频编码的采样率(Sample Rate)、音频编码时所取的声道数目(单声道或是立体声)、音频编码比特率，音频控制模块将此设定值套用至音频编码器上，再启动音频编码器进行编码程序。

[0047] 进行本步骤的程序代码范例如下所示：

```
Format = MediaFormat.createAudioFormat(MIME_TYPE, SAMPLE_RATE, 2);
format.setInteger(KEY_AAC_PROFILE, AACObjectLC);
format.setInteger(KEY_CHANNEL_MASK, CHANNEL_IN_STEREO);
format.setInteger(KEY_BIT_RATE, BIT_RATE);
```

[0048]

```
format.setInteger(KEY_CHANNEL_COUNT, 2);

mEncoder = createEncoderByType(AUDIO_MIME_TYPE);
mEncoder.configure(format, null, null, CONFIGURE_ENCODE);
mEncoder.start();
```

[0049] 步骤S04、获取经编码的音频编码数据：

[0050] 本发明的音频控制模块建立音频编码接收线程，其在获取经编码器编码的音频编码数据前，将先通过同步锁机制确认未经编码的原始音频数据已储存在音频缓冲区中，若确认完成，则通过音频编码器的硬件输出层获取输出队列的音频编码数据，再分送至对应不同串流服务器所属的个别线程的音频数据缓冲区中，音频控制模块并记录下音频数据的数据量大小，并移除不需要的音频数据文件头，只取实际所需的音频数据，等待输出音频串流封包的线程依序来取出使用。

[0051] 进行本步骤的程序代码范例如下所示：

```
synchronized (mSync) {
    try {
        mSync.wait();
    } catch (final InterruptedException e) {
    }
}

int ret = -1;
ByteBuffer[] encoderOutputBuffers = mEncoder.getOutputBuffers();
while (true) {
    index = mEncoder.dequeueOutputBuffer(mBufferInfo, TIMEOUT_US);
    if (index == INFO_OUTPUT_FORMAT_CHANGED) {
        mPacketizer.addFormat(mEncoder.getOutputFormat());
    } else if (index == INFO_TRY_AGAIN_LATER) {
        //break;
    } else if (index < 0) {
        Log.e(TAG, "something happen?");
        break;
    } else {
        encodedData = encoderOutputBuffers[index];
        if (encodedData == null) {
            throw new RuntimeException("encoderOutputBuffer " + index + " was null");
        }
```

[0052]

[0053] 步骤S05、整理并记录音频编码数据及编码时的音频关联参数：

[0054] 在启动音频编码器的音频编码程序后，音频控制模块将可得到针对此次音频编码程序所产生的音频专用配置ASC(Audio Specific Config，ASC)信息，其可提供给后端译码器参考使用，此ASC参数用来描述此次音频编码程序所利用的音频格式、音频取样频率或音频信道数量等，此参数需受保存，以在稍后开始传递音频数据封包串流时，作为第一个传递的音频封包，才能提供给译码端译码时参考使用。

[0055] 进行本步骤的程序代码范例如下所示：

```
private byte[] makeAsc(int sampleRateIndex)
{
    byte asc[] = new byte[2];
    asc[0] = (byte) ( 0x10 | ((sampleRateIndex>>1) & 0x3) );
    asc[1] = (byte) ( ((sampleRateIndex & 0x1)<<7) | ((channelCount & 0xF) << 3) );

    logfile.WritetoFile(Thread.currentThread().getStackTrace()[2], "asc:" + printBuffer(asc, 0, asc.length));
    Log.i(TAG, "asc:" + printBuffer(asc, 0, asc.length));
    return asc;
}
```

[0056]

[0057] 步骤S06、进行音频编码数据的时间戳校正补偿：

[0058] 本发明的音频控制模块会将音频编码数据自专属的音频缓冲区中取出以进行时间戳校正，由于音频编码器是采取每秒固定帧率的输出，故若使用收到音频数据时的统时间作为时间戳(Timestamp)，译码端服务器则会依据所述时间戳进行译码，其结果将导致音频听起来不连续，所以在记录音频的时间戳时，本发明利用了音频编码器的输出帧率固定的特性计算帧与帧之间的时间间隔，再乘以目前的累计帧数，即为实时的音频时间戳值。

[0059] 进行本步骤的程序代码范例如下所示：

```
// audio timestamp calculation is
// interval = 1024 * 1000(ms) / sample rate(hz)
// so we use packet number multiple intervalas AAC time stamp
tmp = fillFrame_media_codec();
current_ts = (int) (audio_num * 10240 / audiostream.get_sampling_rate()) + timeBase;
audio_num++;
```

[0060]

[0061]

```
now_timestamp=System.currentTimeMillis();

if (avctype == 0) {
    currentTime = 0;
}
else {
    if(bUseCommonTimestamp == true)
        currentTime = (int) (now_timestamp - timeBase);
    else
        currentTime = (int)(timestamp - timeBase);

    //Log.e(TAG, "audio ts: " + currentTime);
}

send(makeMessageFromTag(new Tag(IoConstants.TYPE_AUDIO, currentTime, bodysize, body, prevSize)));
prevSize = bodysize;
```

[0062] 步骤S07、对影像编码器进行影像关联参数设定：

[0063] 本发明的影像控制模块，将预先被设定有影像编码时所需的影像参数，其称作影像关联参数，其内容可以包括：影像宽度、影像高度、影像编码帧率(Frame per Second，FPS)、影像的图像群组(Group of Pictures，GOP)、影像编码比特率、影像编码格式等等，影像控制模块通过所述影像关联参数来建立影像获取来源和影像编码器的间的关联，以使影像编码器可以开始进行影像编码程序。

[0064] 进行本步骤的程序代码范例如下所示：

[0065]

```
Format = createVideoFormat(MIME_TYPE, width, height);
format.setInteger(KEY_COLOR_FORMAT, COLOR_FormatSurface);
format.setInteger(KEY_BIT_RATE, 1024*1024);
format.setInteger(KEY_FRAME_RATE, FRAME_RATE);
format.setInteger(KEY_I_FRAME_INTERVAL, IFRAME_INTERVAL);

Log.d(TAG, "created video format: " + format);
mEncoder = createEncoderByType(MIME_TYPE);
mEncoder.configure(format, null, null, CONFIGURE_ENCODE);
mSurface = mEncoder.createInputSurface();
Log.d(TAG, "created input surface: " + mSurface);
mEncoder.start();
```

[0066] 步骤S08、通过影像控制模块取得经编码的影像编码数据：

[0067] 通过步骤S07后，本发明的影像控制模块建立起获取影像的影像编码接收线程，而本发明的影像控制模块将先检查影像编码器的编码输出队列是否有经编码后的影像数据，若结果为是，则影像控制模块将通过影像编码器的硬件抽象层获取实时的影像编码数据，影像控制模块亦可额外记录影像编码数据的大小、影像是否为关键帧(I-frame)、影像数据的时间戳、或将不需要的影像数据文件头移除只留下实际所需的编码数据，影像控制模块再将影像编码数据以及这些信息储存于其专属的影像缓冲区的中。

[0068] 进行本步骤的程序代码范例如下所示：

```
ret = videostream.recordVirtualDisplay();
if(false && ret == -1)
{
    Log.e(TAG, "release_media_codec");
    videostream.release_media_codec();
    return null;
}

naluLength = videostream.getEncodedLength();
header[0] = (byte) (((naluLength - 4) >> 24) & 0xff);
header[1] = (byte) (((naluLength - 4) >> 16) & 0xff);
header[2] = (byte) (((naluLength - 4) >> 8) & 0xff);
header[3] = (byte) (((naluLength - 4) >> 0) & 0xff);

byte[] tmp = new byte[naluLength];
videostream.getEncodedData(tmp);
videostream.putEncodedData();

if(videostream.isStreaming() == false)
    return null;

naluLength -= 4;
byte[] nalu = new byte[naluLength + header.length];
System.arraycopy(header, 0, nalu, 0, header.length);
System.arraycopy(tmp, 4, nalu, header.length, naluLength);

 byte[] nalu = packet.data;
 int nalType = nalu[4] & 0x1F;

 if (nalType == 5 && !sentConfig) {
     Log.i(TAG, "Send configuration one time");
     byte[] conf = configurationFromSpsAndPps();
     writemetadata();
     writeVideoNalu(conf, System.currentTimeMillis(), 0, true);
     sentConfig = true;
 }

 if (nalType == 7 || nalType == 8) {
     Log.w(TAG, "Received SPS/PPS frame, ignored");
 }
 //System.currentTimeMillis();
 if(sentConfig == true)
     writeVideoNalu(nalu, packet.timestamp, 1, (nalType == 5));
```

[0069]

[0070] 步骤S09、整理并记录影像编码数据及编码时的影像编码参数：

[0071] 在影像编码器启动影像编码后，本发明的影像控制模块可获取到此次影像编码程序所产生的序列参数集SPS(Sequence Parameter Set)和图像参数集PPS(Picture Parameter Set)参数，参数内容包含使用于编码的影像格式、编码的影像格式级别、编码影像的长宽，去区块Deblock滤波器的种类等等信息；此两参数皆需受到保存，以在稍后传递影像数据封包的串流时，包装为第一个传递出去的影像封包，才可将用以描述初始化影像译码器所需要的信息参数提供给译码端。

[0072] 进行本步骤的程序代码范例如下所示：

```
encodedData = mEncoder.getOutputBuffer(index);
if ((mBufferInfo.flags & CODEC_CONFIG) != 0) {
    encodedData.clear();
    encodedData.get(data);
    for(int i = 0; i < data.length && found < 2; i++) {
        if(data[i] == 0x00 && data[i+1] == 0x00 && data[i+2] == 0x00 && data[i+3] == 0x01 && found == 0)
        {
            sps_index = i;
            sps_length = 1;
            found++;
            continue;
        }

        if(data[i] == 0x00 && data[i+1] == 0x00 && data[i+2] == 0x00 && data[i+3] == 0x01 && found == 1) {
            pps_index = i;
            sps_length = (i - sps_index - 4);
            pps_length = (data.length - pps_index - 4);
            found++;
            continue;
        }
    }

    System.arraycopy(data, (sps_index+4), sps, 0, sps_length);
    System.arraycopy(data, (pps_index+4), pps, 0, pps_length);

    ((H264Packetizer)mPacketizer).setStreamParameters(pps, sps);
    mBufferInfo.size = 0;
}
```

[0073]

```
IoBuffer conf = IoBuffer.allocate(11+sps.length+pps.length);
conf.put((byte) 1); // version
conf.put(sps[1]);   // profile
conf.put(sps[2]);   // compat
conf.put(sps[3]);   // level

conf.put((byte) 0xff);  // 6 bits reserved + 2 bits nal size length - 1 (11)
conf.put((byte) 0xe1);  // 3 bits reserved + 5 bits number of sps (00001)

conf.put(be16((short)sps.length));
conf.put(sps);

conf.put((byte)1);
conf.put(be16((short)pps.length));
conf.put(pps);


return conf.array();
```

[0074] 步骤S10、进行影像编码数据的时间戳校正补偿：

[0075] 本发明的影像控制模块，将根据本发明欲分送信息的相异串流服务器，以将影像编码数据从专属的影像缓冲区中取出，再分送至不同的串流服务器所属的个别线程的影像数据缓冲区，但由于各行动装置所使用的影像编码器型态有可能不同，其中，有若干影像编码器并非采用固定帧率(FPS)输出，而是会随着目前执行的程序来改变编码输出的帧率，所以在记录影像数据的时间戳(Timestamp)时，若以固定的时间间隔来进行计算，则使影像观看起来有不连续或忽快忽慢的状况发生，因此本发明的影像控制模块将影像编码数据所使用的时间戳根据状况修改为实际收到影像编码数据的时间，以达到校正时间使影像连续的目的。

[0076] 进行本步骤的程序代码范例如下所示：

[0077]

```
now_timestamp=System.currentTimeMillis();

if(avctype == 0) {
    currentTime = 0;
}
else {
    if(bUseCommonTimestamp == true)
        currentTime = (int) (now_timestamp - timeBase);
    else
        currentTime = (int)(timestamp - timeBase);

    //Log.e(TAG, "video ts: " + currentTime);
}

send(makeMessageFromTag(new Tag(IoConstants.TYPE_VIDEO, currentTime, bodysize, body, prevSize)));
```

[0078] 接着，请参照本发明的图3，其为接续图2的本发明同步获取影音以进行一对多影音串流的方法的步骤流程图。

[0079] 步骤S11、整合并建立影像与声音的影像及音频描述参数：

[0080] 串流协议建立后，在进行影像参数封包和音频参数封包的传递前，本发明的串流控制模块须先传送关于此次串流将要传递的影像与声音的影像及音频描述参数(Metadata)，其中，参数内容包含有影像宽度(Width)、影像高度(Height)、影像编码数据量(Video Data Rate)、影像帧率(Frame Rate)、影像编码格式(Video Codec Id)、音频编码数据量(Audio Data Rate)、音频取样率(Audio Sample Rate)、音频取样数(Audio SampleSize)、音频通道数目(Audio Channels)、音频为单声道(Mono)或是立体声道(Stereo)、音频编码格式(Audio Codec Id)，所述信息在影像编码和音频编码建立时都已获得，即为音频编码参数以及影像编码参数，需在串流控制模块将影像或音频封包送出前，先送出此影像及音频描述参数，以供译码端在串流建立时能参考使用。

[0081] 进行本步骤的程序代码范例如下所示：

[0082]

```
private void writemetadata() throws IOException {
    IoBuffer buf = IoBuffer.allocate(192);
    buf.setAutoExpand(true);
    Output out = new Output(buf);
    out.writeString("@setDataFrame");
    out.writeString("onMetaData");
    Map<Object, Object> params = new HashMap<Object, Object>();
    params.put("duration", Integer.valueOf(0));
    params.put("filesize", Integer.valueOf(0));
    params.put("width", Integer.valueOf(v_width));
    params.put("height", Integer.valueOf(v_height));
    params.put("videocodecid", "avc1");
    params.put("videodatarate", Integer.valueOf(1000));
    params.put("framerate", Integer.valueOf(30));
    params.put("audiocodecid", "mp4a");
    params.put("audiodatarate", Integer.valueOf(128));
    params.put("audiosamplerate", Integer.valueOf(44100));
    params.put("audiosamplesize", Integer.valueOf(16));
    params.put("audiochannels", Integer.valueOf(2));
    params.put("stereo", Boolean.TRUE);
    out.writeObject(params, new Serializer());
    buf.flip();
    Tag onMetaData = new Tag(IoConstants.TYPE_METADATA, 0, 100, buf, 0);
    send(makeMessageFromTag(onMetaData));
}
```

[0083] 步骤S12、检查与标记音频编码数据：

[0084] 在串流控制模块对音频数据缓冲区内经校正的音频编码数据封装成串流数据前，需先确认音频编码数据的取样率、立体声或单声道、音频位数或音频编码格式，在封装时，串流控制模块需在串流数据中标记这些信息，以便译码服务器端作译码参考。

[0085] 进行本步骤的程序代码范例如下所示：

[0086]
```
byte tagType = (byte) ((IoConstants.FLAG_FORMAT_AAC << 4)) | (IoConstants.FLAG_SIZE_16_BIT << 1
tagType |= IoConstants.FLAG_RATE_44_KHZ << 2;
// FIXME: AudioStream already fixed the channel count is 1, so mono only!
tagType |= (channelCount == 2 ? IoConstants.FLAG_TYPE_STEREO : IoConstants.FLAG_TYPE_MONO);

body.setAutoExpand(true);
body.put(tagType);
body.put((byte) avctype);
```

[0087] 步骤S13、检查与标记影像编码数据：

[0088] 在串流控制模块对影像数据缓冲区内经校正的影像编码数据封装成串流数据前，首先需检查影像帧是否为关键帧(I-Frame)、是否为SPS或PPS封包、影像编码器的编码格式等相关信息，在封装时，串流控制模块需在串流数据中标记这些信息，译码端才可根据这些数据配置所需的影像译码资源进行译码。

[0089] 进行本步骤的程序代码范例如下所示：

[0090]
```
if (keyframe) {
    flag |= (IoConstants.FLAG_FRAMETYPE_KEYFRAME << 4);
}else {
    flag |= (IoConstants.FLAG_FRAMETYPE_INTERFRAME << 4);
}

body.setAutoExpand(true);
body.put(flag);
body.put((byte) avctype);
// TODO: if x264 come with B-frame, delay must set to correct value.
body.put(be24(delay));
body.put(nalu);
body.flip();
body.limit(bodysize);
```

[0091] 步骤S14、设定服务器串流联机：

[0092] 本发明的串流控制模块将针对相异的串流服务器进行联机的初始设定，首先，先初始化联机状态的Socket并分配联机时所需资源，再设定串流服务器网址、端口以及密钥，接着启动串流控制模块与串流服务器的间联机，并设定联机状态为认证阶段，经由双方的握手(Handshake)认证机制确认后，设定传输时的参数，例如：最大信息封包大小(chunk size)或是带宽(Bandwidth)，至此，串流联机的初始设定完成，串流控制模块再随后将联机状态变更为可以传递数据的阶段。

[0093] 进行本步骤的程序代码范例如下所示：

```
public void setHost(String host) {
    this.host = host;
}

public void setPort(int port) {
    this.port = port;
}

public void setApp(String app) {
    this.app = app;
}

public int getState() {
    return currentState;
}

synchronized void setState(int state) {
    this.currentState = state;
    Log.i(TAG, "RTMP state:" + state);
}
public synchronized void start(String publishName, String publishMode, Object[] params) {
    setState(CONNECTING);
    this.publishName = publishName;
    this.publishMode = publishMode;
    rtmpClient = new RTMPClient();

    Map<String, Object> defParams = rtmpClient.makeDefaultConnectionParams(host, port, app);
    rtmpClient.connect(host, port, defParams, this, params);
}
```

[0094]

```
switch (header.getDataType()) {
    case TYPE_CHUNK_SIZE:
        onChunkSize(conn, channel, header, (ChunkSize) message);
        break;
    case TYPE_INVOKE:
    case TYPE_FLEX_MESSAGE:
        onInvoke(conn, channel, header, (Invoke) message, (RTMP) session.getAttribute(ProtocolState.SESSION_KEY));
        IPendingServiceCall call = ((Invoke) message).getCall();
        if (message.getHeader().getStreamId() != 0 && call.getServiceName() == null && StreamAction.PUBLISH.equals(call.getServiceMethodName())) {
            if (stream != null) {
                // Only dispatch if stream really was created
                ((IEventDispatcher) stream).dispatchEvent(message);
            }
        }
        break;
    case TYPE_NOTIFY: // just like invoke, but does not return
        if (((Notify) message).getData() != null && stream != null) {
            // Stream metadata
            ((IEventDispatcher) stream).dispatchEvent(message);
        } else {
            onInvoke(conn, channel, header, (Notify) message, (RTMP) session.getAttribute(ProtocolState.SESSION_KEY));
        }
        break;
    case TYPE_FLEX_STREAM_SEND:
        if (stream != null) {
            ((IEventDispatcher) stream).dispatchEvent(message);
        }
        break;
```

```
case TYPE_AUDIO_DATA:
case TYPE_VIDEO_DATA:
    //mark the event as from a live source
    //log.trace("Marking message as originating from a Live source");
    message.setSourceType(Constants.SOURCE_TYPE_LIVE);
    // NOTE: If we respond to "publish" with "NetStream.Publish.BadName",
    // the client sends a few stream packets before stopping. We need to ignore them.
    if (stream != null) {
        ((IEventDispatcher) stream).dispatchEvent(message);
    }
    break;
case TYPE_FLEX_SHARED_OBJECT:
case TYPE_SHARED_OBJECT:
    onSharedObject(conn, channel, header, (SharedObjectMessage) message);
    break;
case Constants.TYPE_CLIENT_BANDWIDTH: //onBWDone
    log.debug("Client bandwidth: {}", message);
    break;
case Constants.TYPE_SERVER_BANDWIDTH:
    log.debug("Server bandwidth: {}", message);
    break;
default:
    log.debug("Unknown type: {}", header.getDataType());
```

[0095]

[0096] 步骤S15、进行数据封装：

[0097] 由于传送串流封包时，需要通过特定格式的数据文件头(Message Header)提供给解码端解析，故串流控制模块必须根据目前处理的串流资料的内容，判断需选择哪种数据文件头来进行数据封装，而目前可能使用的档头包含下列四种格式(Format＝0、1、2、3)：

[0098] 格式0(FMT0)的数据文件头长度为11字节，其可包含时间戳差值(3bytes)、信息长度(3bytes)、信息种类ID(1bytes)、信息串流ID(4bytes)等部分，其代表此封包为串流中个独立的信息。

[0099] 格式1(FMT1)的数据文件头长度为7字节，其可包含时间戳差值(3bytes)、信息长度(3bytes)、信息种类ID(1bytes)，信息串流ID将被去除，其代表此封包为同个串流中的信息。

[0100] 格式2(FMT2)的数据文件头长度为3字节，其可包含时间戳差值(3bytes)，去除掉信息长度、信息种类ID、信息串流ID等部分，其表示此封包不但是同一串流中的信息，而且数据种类及大小与先前信息相同。

[0101] 格式3(FMT3)的数据文件头长度为0字节，使用此种数据文件头表示一个信息被拆成多笔分送，后续的数据在译码端皆可沿用第一笔信息的时间戳差值、信息长度、信息种类ID以及信息串流ID。

[0102] 串流控制模块自影像及音频描述参数分析出校正过的时间戳及资料大小等，再根据这些信息建构一般串流封包格式的数据文件头，举例来说，可为HEADER FORMAT(0)、CHUNK STREAM ID(3)、DATA MESSAGE TYPE ID(18)，并将分析出的时间戳差值及数据大小填入封包的数据文件头，再将

[0103] 所要传送的数据接于数据文件头的后，即完成数据串流封包的建构。

[0104] 进行本步骤的程序代码范例如下所示：

```
case Constants.TYPE_STREAM_METADATA:
    log.trace("Meta data");
    Notify notify = new Notify(((Notify) msg).getData().asReadOnlyBuffer());
    notify.setHeader(header);
    notify.setTimestamp(header.getTimer());
    data.write(notify);
    break;
```

[0105]

[0106] 步骤S16、进行音频编码数据的串流封装：

[0107] 经编码过的音频编码数据，将被音频控制模块送入线程所专属的音频数据缓冲区，待串流控制模块依线程解析数据内容，分析出时间戳及资料大小，并根据上次音频资料封包的时间戳和本次音频数据封包的时间戳相减得出差值，然后根据取得的这些信息，串流控制模块选择性地建构对应音频串流封包格式的数据文件头，例如：HEADER FORMAT(0～3)、CHUNK STREAM ID(4)、AUDIO MESSAGE TYPE ID(8)，加上分析出来的时间戳差值和数据大小等填入封包的资料文件头中，再将音频数据置于数据文件头后，即完成音频串流封包的建构，而声音部份因为数据量相对小，每个Frame应不会超出一次所能传输的最大信息封包大小(于步骤S14协议时所定义的chunk size)，所以音频串流封包可以直接交由Socket传送至串流服务器。

[0108] 本步骤的程序代码截图如下：

```
case Constants.TYPE_AUDIO_DATA:
    log.trace("Audio data");
    buf = ((AudioData) msg).getData();
    if (buf != null) {
        AudioData audioData = new AudioData(buf.asReadOnlyBuffer());
        audioData.setHeader(header);
        audioData.setTimestamp(header.getTimer());
        audioData.setSourceType(((AudioData)msg).getSourceType());
        audio.write(audioData);
    } else {
        log.warn("Audio data was not found");
    }
    break;
```

[0109]

[0110] 步骤S17、进行影像编码数据的串流封装：

[0111] 经编码过的影像编码数据，将被影像控制模块送入线程所专属的影像数据缓冲区，待串流控制模块依线程解析数据内容，分析出时间戳及资料大小，并根据上次影像数据封包的时间戳和本次影像数据封包的时间戳相减得出差值，然后根据取得的这些信息，串流控制模块选择性地建构对应影像串流封包格式的数据文件头，举例来说，可为HEADER FORMAT(0～3)、CHUNK STREAM ID(5)、VIDEO MESSAGE TYPE ID(9)，并将分析出来的时间戳差值、数据大小等填入封包的数据文件头中，再将影像数据置于数据文件头之后，以完成影像串流封包的建构，但由于影像封包可能包含有关键帧的数据，其数据量有可能超过一次能传输的最大信息封包大小(于步骤S14协议时所定义的chunk size)，此时则需先切割影像串流封包(视实际需求指定步骤S15所列的不同数据文件头)，再交由Socket发送影像串流封包至串流服务器。

[0112] 进行本步骤的程序代码范例如下所示：

[0113]

```
synchronized public void pushMessage(IMessage message) throws IOException {
    if (getState() >= PUBLISHED && message instanceof RTMPMessage) {
        RTMPMessage rtmpMsg = (RTMPMessage) message;
        rtmpClient.publishStreamData(streamId, rtmpMsg);
    } else {
        frameBuffer.add(message);
    }
}

public static int getHeaderLength(byte headerSize) {
    switch (headerSize) {
        case HEADER_NEW:
            return 12;
        case HEADER_SAME_SOURCE:
            return 8;
        case HEADER_TIMER_CHANGE:
            return 4;
        case HEADER_CONTINUE:
            return 1;
        default:
            return -1;
    }
}
```

[0114]

```
case Constants.TYPE_VIDEO_DATA:
    log.trace("Video data");
    buf = ((VideoData) msg).getData();
    if (buf != null) {
        VideoData videoData = new VideoData(buf.asReadOnlyBuffer());
        videoData.setHeader(header);
        videoData.setTimestamp(header.getTimer());
        videoData.setSourceType(((VideoData)msg).getSourceType());
        video.write(videoData);
    } else {
        log.warn("Video data was not found");
    }
    break;
```

[0115] 接续步骤S16和步骤S17后，本发明的串流控制模块即可将封装过的音频和影像串流传输至不同的串流服务器，进而完成本发明的同步获取影音以进行一对多影音串流的方法。

[0116] 上列详细说明针对本发明的可行实施例的具体说明，所述实施例并非用以限制本发明的专利范围，凡未脱离本发明技艺精神所为的等效实施或变更，均应包含于本案的专利范围中。

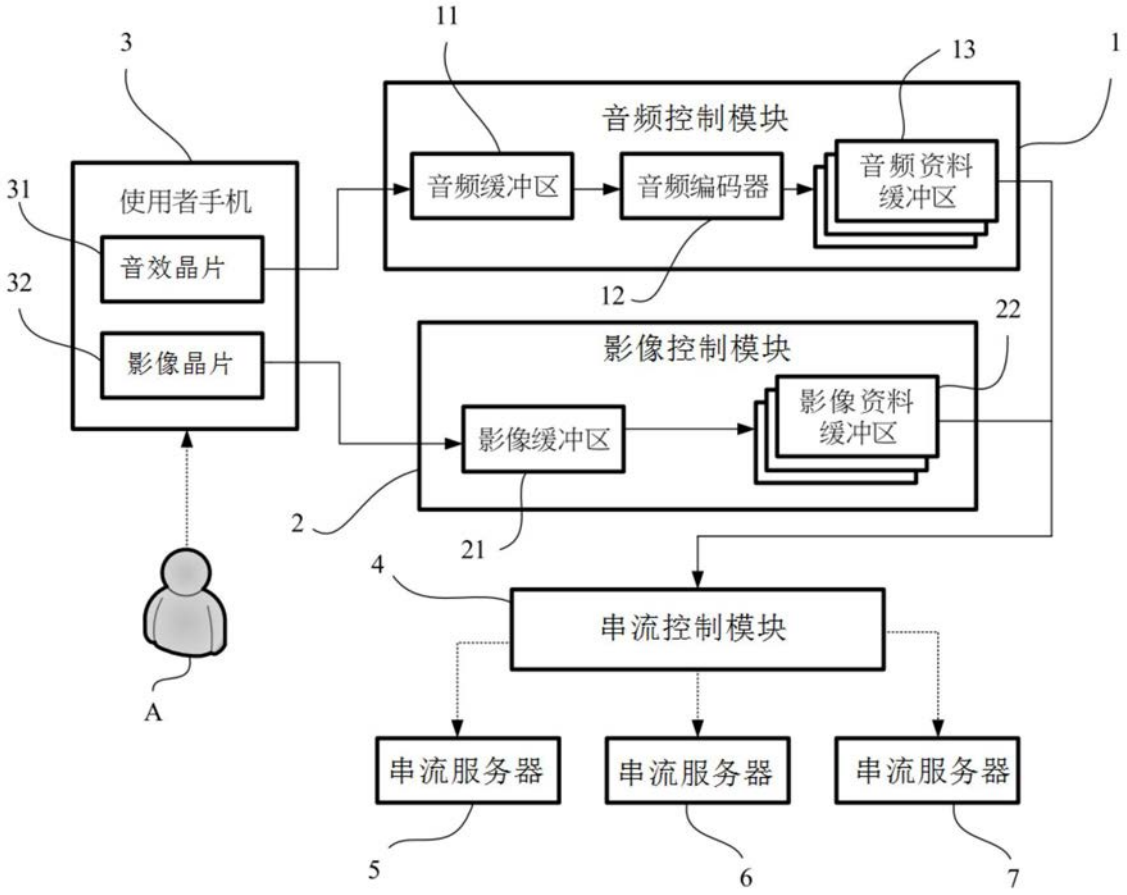

3
11
13
1
音频控制模块
音频缓冲区
音频编码器
音频资料缓冲区
31
使用者手机
音效晶片
32
影像晶片
12
22
影像控制模块
影像资料缓冲区
影像缓冲区
2
21
4
串流控制模块
A
串流服务器
串流服务器
串流服务器
5
6
7

图1

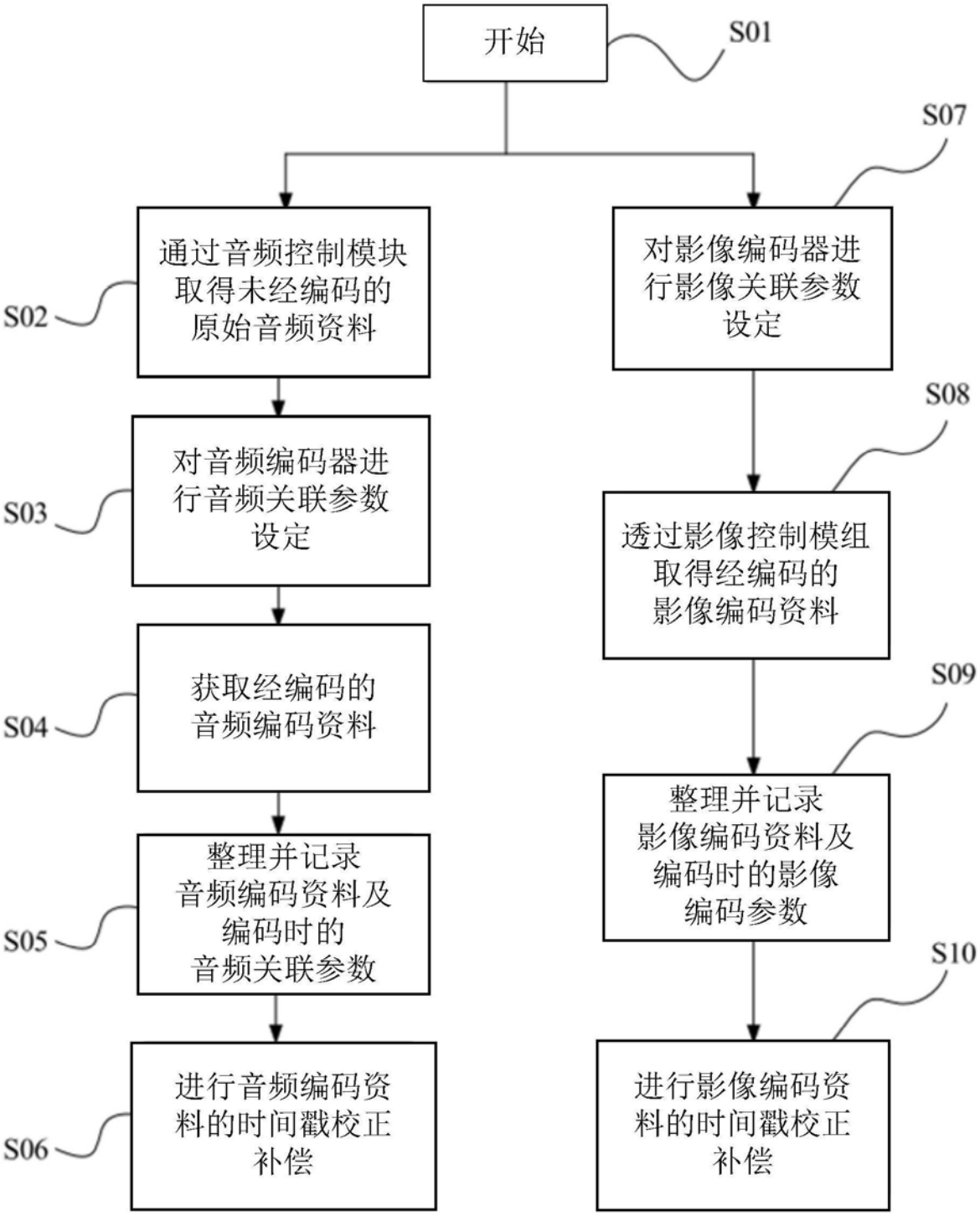

开始
S01
S07
通过音频控制模块取得未经编码的原始音频资料
S02
对影像编码器进行影像关联参数设定
S08
对音频编码器进行音频关联参数设定
S03
透过影像控制模组取得经编码的影像编码资料
S09
获取经编码的音频编码资料
S04
整理并记录影像编码资料及编码时的影像编码参数
S10
整理并记录音频编码资料及编码时的音频关联参数
S05
进行影像编码资料的时间戳校正补偿
进行音频编码资料的时间戳校正补偿
S06

图2

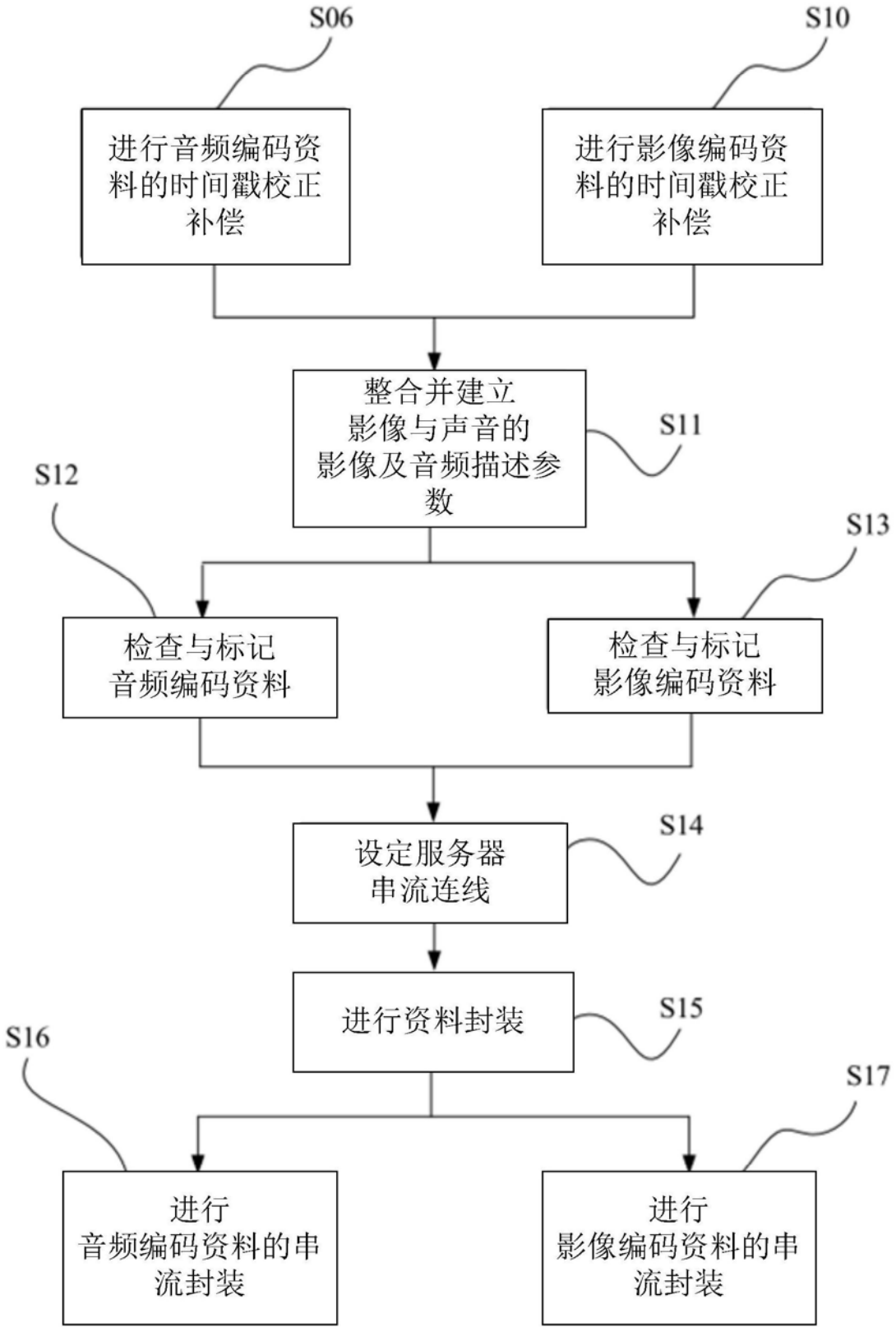

S06
进行音频编码资料的时间戳校正补偿
S10
进行影像编码资料的时间戳校正补偿
整合并建立影像与声音的影像及音频描述参数
S11
S12
检查与标记音频编码资料
S13
检查与标记影像编码资料
设定服务器串流连线
S14
进行资料封装
S15
S16
进行音频编码资料的串流封装
S17
进行影像编码资料的串流封装

图3