第 5 章

HLS 流媒体协议

HLS 与 RTMP 都是流媒体协议。RTMP 由 Adobe 开发，广泛应用于低延时直播，也是编码器和服务器对接的实际标准协议，在 PC(Flash)上有最佳观看体验和最佳稳定性。HLS 由苹果公司开发，本身是 Live(直播)的，但也支持 VoD(点播)。HLS 是苹果平台的标准流媒体协议，和 RTMP 在 PC 上一样支持得非常完善。HLS 协议的详细内容可以参考网址 https://datatracker.ietf.org/doc/html/draft-pantos-http-live-streaming-08。

5.1 HLS 协议简介

HLS，全称 HTTP Live Streaming，是一种由苹果公司提出的基于 HTTP 的流媒体网络传输协议，是 QuickTime X 和 iPhone 软件系统的一部分。它的工作原理是把整个流分成一个个小的基于 HTTP 的文件来下载，每次只下载一些。当媒体流正在播放时，客户端可以选择从许多不同的备用源中以不同的速率下载同样的资源，允许流媒体会话适应不同的数据速率。在开始一个流媒体会话时，客户端会下载一个包含元数据的 extended m3u/m3u8 playlist 文件，用于寻找可用的媒体流。HLS 只请求基本的 HTTP 报文，与实时传输协议(RTP)不同，HLS 可以穿过任何允许 HTTP 数据通过的防火墙或者代理服务器。它也很容易使用内容分发网络来传输媒体流。HLS 的网络框架结构如图 5-1 所示。

(1) 服务器将媒体文件转换为 m3u8 及 TS 分片；对于直播源，服务器需要实时动态更新。

(2) 客户端请求 m3u8 文件，根据索引获取 TS 分片；点播与直播服务器不同的地方是，直播的 m3u8 文件会不断更新，而点播的 m3u8 文件是不会变的，只需客户端在开始时请求一次。

5.1.1 HLS 的索引文件的嵌套

HLS 协议中的索引文件可以嵌套，一般只有一级索引和二级索引；媒体流封装的分片格式只支持 MPEG-2 传输流(TS)、WebVTT 文件或 Packed Audio 文件。

索引文件(m3u8)和媒体分片(TS)之间的关系如图 5-2 所示。一级 m3u8 嵌套二级 m3u8，二级 m3u8 描述 TS 分片。

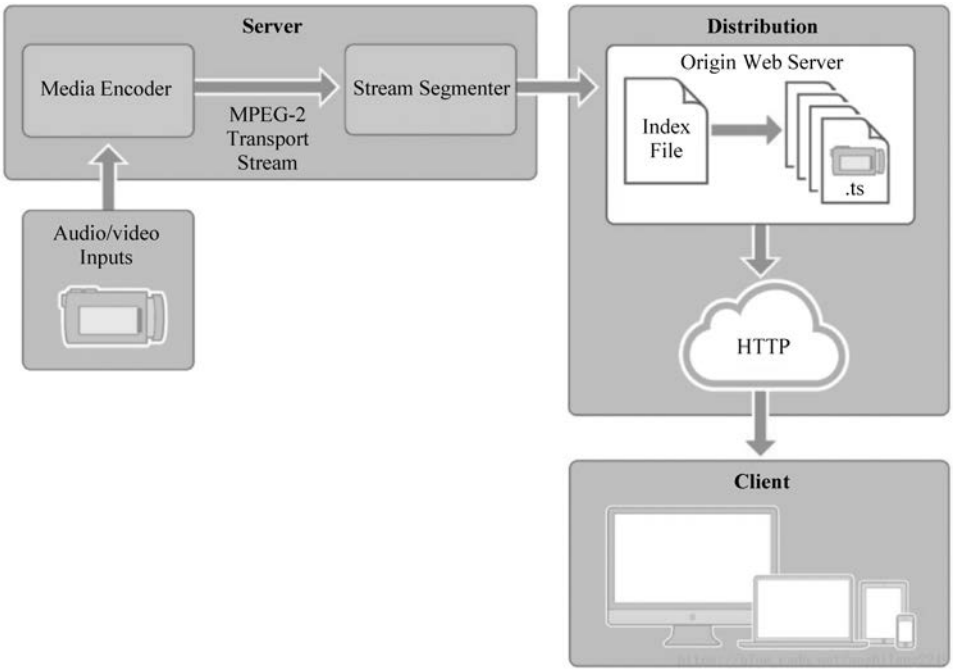


图 5-1　HLS 框架

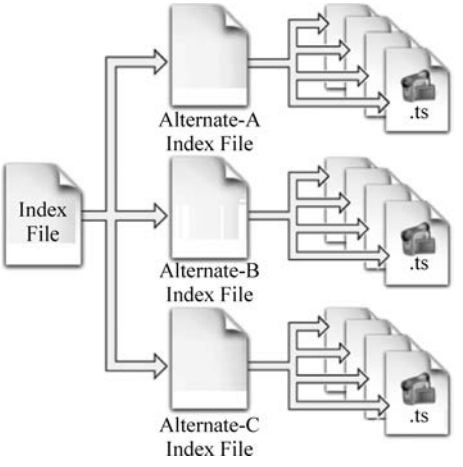


图 5-2　HLS 索引文件的嵌套关系

5.1.2　HLS 服务器端和客户端工作流程

在服务器端，流媒体文件被切割成一个一个的小分片，这些小分片有着相同的时长（常为 10s，也可以灵活设置），每个小分片是一个 TS 文件。同时会产生一个索引文件（m3u8），

索引文件里存放了 TS 文件的 URL。

客户端请求方式分两种，一种是点播(VoD)，一种是直播(Live)。

(1) VoD：全称 Video on Demand，即视频点播，有求才播放。客户端一次获取整个 m3u8 文件，按照里面的 URL 获取 TS 文件，采用 HTTP。

(2) Live：由于 m3u8 文件是实时更新的，所以客户端每隔一段时间会获取 m3u8 文件，再根据里面的 URL 获取 TS 文件，采用 HTTP。

客户端与服务器端通过 HTTP 进行交互，以两级 m3u8 嵌套为例，客户端先 GET 请求到一级 m3u8，一级 m3u8 里面包含了服务器端可以用于传播的一个或多个不同带宽的 URL，这个 URL 可以获取二级 m3u8；二级 m3u8 包含了多个 TS 分片的 duration 及其 URL，最后通过这个 URL 下载 TS 分片。HLS 服务器端和客户端的交互流程如图 5-3 所示。

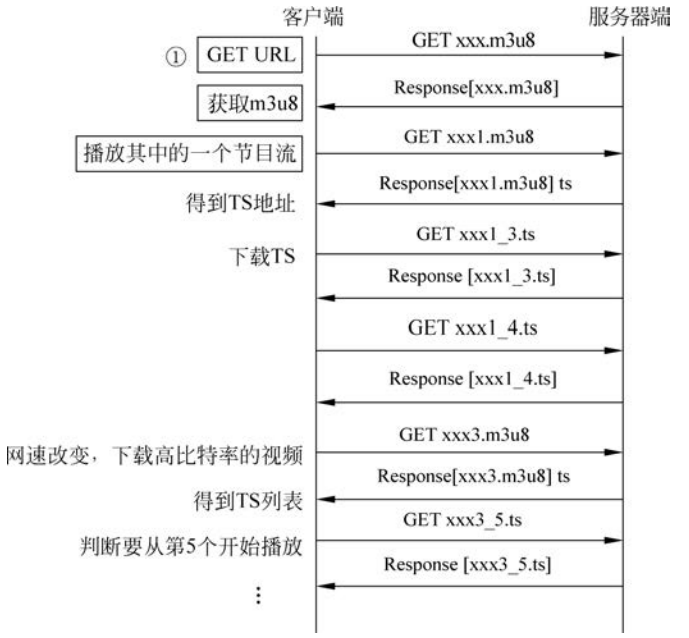


图 5-3　HLS 服务器端和客户端的交互流程

5.1.3　HLS 优势及劣势

HLS 的优势主要包括以下几点：

(1) 客户端支持简单，只需支持 HTTP 请求，HTTP 协议无状态，只需按顺序下载媒体片段。

(2) 使用 HTTP 协议网络兼容性好，HTTP 数据包也可以方便地通过防火墙或者代理服务器。

(3) 当媒体流正在播放时，客户端可以选择从许多不同的备用源中以不同的速率下载

同样的资源(多码流自适应),允许流媒体会话适应不同的数据速率。

HLS的劣势:因其自身的实现方式,HLS存在延迟(最少有一个分片),对于直播等对实时敏感的场景,延迟比较大,用户体验不太好。

5.1.4 HLS主要的应用场景

HLS主要的应用场景,包括以下几个方面。

(1) 跨平台:PC主要的直播方案是RTMP,而如果选一种协议能跨PC/Android/iOS,那就是HLS。

(2) iOS上苛刻的稳定性要求:iOS上最稳定的是HLS,稳定性不差于RTMP在PC Flash上的表现。

(3) 友好的CDN分发方式:目前CDN对于RTMP也是基本协议,但是HLS分发的基础是HTTP,所以CDN的接入和分发会比RTMP更加完善。能在各种CDN之间切换,RTMP也能,只是可能需要对接测试。

(4) 简单:HLS作为流媒体协议非常简单,苹果公司的产品支持也很完善。Android对HLS的支持也会越来越完善。

5.2 HLS协议详细讲解

HLS协议规定,视频的封装格式是TS;视频的编码格式为H.264;声频编码格式为MP3、AAC或者AC-3。除了TS视频文件本身,还定义了用来控制播放的m3u8文件(文本文件)。

HLS需要提供一个m3u8播放地址,苹果公司的Safari浏览器直接就能打开m3u8地址,例如http://demo.srs.com/live/livestream.m3u8。Android的浏览器不能直接打开,需要使用HTML5的video标签,然后在浏览器中打开这个页面即可,代码如下:

```
//chapter5/hls.h5.video.html
<!-- livestream.html -->
<video width="640" height="360"
        autoplay controls autobuffer
        src="http://demo.srs.com/live/livestream.m3u8"
        type="application/vnd.apple.mpegurl">
</video>
```

5.2.1 m3u8简介

HLS协议中的m3u8,是一个包含TS列表的文本文件,目的是告诉客户端或浏览器可以播放这些TS文件。m3u8的一些主要标签解释如下。

(1) EXTM3U:每个m3u8文件的第一行必须是这个tag,提供标识作用。

(2) EXT-X-VERSION：用以标示协议版本。例如这里是3，表明这里用的是HLS协议的第3个版本，此标签只能有0或1个，不写代表使用版本1。

(3) EXT-X-TARGETDURATION：所有切片的最大时长，如果不设置这个参数，则有些苹果设备就会无法播放。

(4) EXT-X-MEDIA-SEQUENCE：切片的开始序号。每个切片都有唯一的序号，相邻序号+1。这个编号会继续增长，保证流的连续性。

(5) EXTINF：TS切片的实际时长。

(6) EXT-X-PLAYLIST-TYPE：类型，VoD表示点播，Live表示直播。

(7) EXT-X-ENDLIST：文件结束符号，表示不再向播放列表文件添加媒体文件。

一个典型的m3u8示例文件，代码如下：

```
//chapter5/hls.sample1.m3u8
#EXTM3U                                  //开始标识
#EXT-X-VERSION:3                         //版本为3
#EXT-X-ALLOW-CACHE:YES                   //允许缓存
#EXT-X-TARGETDURATION:13                 //切片最大时长
#EXT-X-MEDIA-SEQUENCE:430                //切片的起始序列号
#EXT-X-PLAYLIST-TYPE:VoD                 //VoD表示点播
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1280000
http://example.com/low.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2560000
http://example.com/mid.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=7680000
http://example.com/hi.m3u8
#EXTINF:11.800
news-430.ts
#EXTINF:10.120
news-431.ts
#EXT-X-DISCONTINUITY
#EXTINF:11.952
news-430.ts
#EXTINF:12.640
news-431.ts
#EXTINF:11.160
news-432.ts
#EXT-X-DISCONTINUITY
#EXTINF:11.751
news-430.ts
#EXTINF:2.040
news-431.ts
#EXT-X-ENDLIST                           //结束标识
```

(1) BANDWIDTH 用于指定视频流的比特率。

(2) #EXT-X-STREAM-INF 的下一行是二级 index 文件的路径,可以用相对路径,也可以用绝对路径。上文例子中用的是相对路径。这个文件中记录了不同比特率视频流的二级 index 文件路径,客户端可以判断自己的现行网络带宽,以此来决定播放哪一个视频流,也可以在网络带宽变化时平滑地切换到和带宽匹配的视频流。二级文件实际负责给出 TS 文件的下载网址,这里同样使用了相对路径。

5.2.2 HLS 播放模式

点播 VoD 的特点是当前时间点可以获取所有 index 文件和 TS 文件,二级 index 文件中记录了所有 TS 文件的地址。这种模式允许客户端访问全部内容。在上面的例子中是一个点播模式下的 m3u8 的结构。

Live 模式是实时生成 m3u8 和 TS 文件。它的索引文件一直处于动态变化中,播放时需要不断下载二级 index 文件,以获得最新生成的 TS 文件播放视频。如果一个二级 index 文件的末尾没有 #EXT-X-ENDLIST 标志,则说明它是一个 Live 视频流。

客户端在播放 VoD 模式的视频时其实只需下载一次一级 index 文件和二级 index 文件便可以得到所有 TS 文件的下载网址,除非客户端进行比特率切换,否则无须再下载任何 index 文件,只需按顺序下载 TS 文件并播放就可以了。

Live 模式下略有不同,因为播放的同时,新 TS 文件也在被生成中,所以客户端实际上是下载一次二级 index 文件,然后下载 TS 文件,再下载二级 index 文件(此时这个二级 index 文件已经被重写,记录了新生成的 TS 文件的下载网址),然后下载新 TS 文件,如此反复进行播放。

5.2.3 TS 文件

TS 文件是传输流文件(MPEG2-Transport Stream),视频编码主要格式为 H.264/MPEG4,声频为 AAC/MP3。TS 文件分为 3 层,包括 TS 层(Transport Stream)、PES 层(Packet Elemental Stream)和 ES 层(Elementary Stream)。ES 层是原始的音视频压缩数据,PES 层是在音视频数据 ES 上加了时间戳(PTS/DTS)等对数据帧的说明信息,TS 层是在 PES 层加入数据流的识别和传输所需的信息,这 3 层结构如图 5-4 所示。

(1) TS 包大小固定为 188B,TS 层分为 3 个部分,即 TS Header、Adaptation Field 和 Payload。TS Header 固定为 4B; Adaptation Field 可能存在也可能不存在,主要作用是给不足 188B 的数据做填充; Payload 是 PES 数据。

(2) PES 是在每个视频/声频帧上加入了时间戳等信息,PES 包的内容很多,通常只留下最常用的。

(3) ES 层指的是音视频数据,例如 H.264 视频。

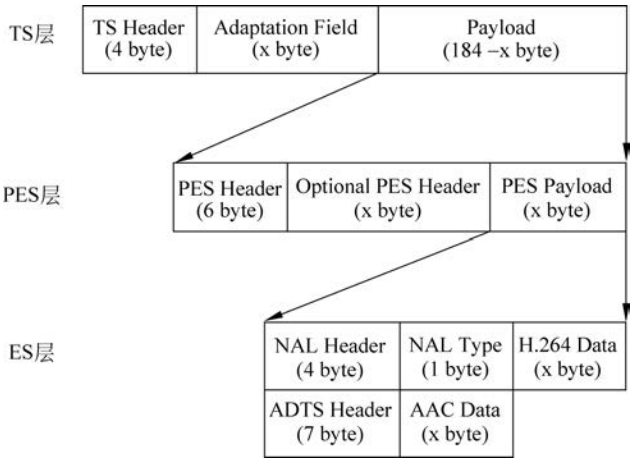


图 5-4　ES/PES/TS 三层结构

5.3　m3u8 格式讲解

HLS 协议很大一部分内容是对 m3u8 文本协议的描述。m3u8 即播放索引文件，也称为 Playlist，是由多个独立行组成的文本文件，必须通过 URI(. m3u8 或. m3u)或者 HTTP Content-Type 来识别(application/vnd. apple. mpegurl 或 audio/mpegurl)。

m3u8 文件实际上是一个播放列表(Playlist)，可能是一个媒体播放列表(Media Playlist)，也可能是一个主列表(Master Playlist)，但无论是哪种播放列表，其内部文字使用的都是 UTF-8 编码。当 m3u8 文件作为媒体播放列表(Media Playlist)时，其内部信息记录的是一系列媒体片段资源，按顺序播放该片段资源，即可完整展示多媒体资源，由此可知，整个视频的总时长是各个 TS 切片资源的时长之和。

m3u8 的每行由用\n 或者\r\n 来标识换行。每一行可以是一个 URI、空白行或是一个以＃号开头的字符串。以＃开头的是 tag 或者注释，以＃EXT 开头的是 tag，其余的为注释，在解析时应该忽略。URI 表示一个 TS 分片地址或是 Playlist 地址。URI 可以用绝对地址或者相对地址，如果使用相对地址，则是相对于当前 Playlist 的地址。有些 tag 带有属性值，多个属性用逗号分隔。

一个常见的一级 m3u8 示例文件，代码如下：

```
//chapter5/hls.firstLevel.sample.m3u8
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=700,000
http://xxx.itv.cmvideo.cn/low.m3u8?channel-id=bstvod&Contentid=4007432528
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1300,000
http://xxx.itv.cmvideo.cn/mid.m3u8?channel-id=bstvod&Contentid=4007432527
```

```
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2300,000
http://xxx.itv.cmvideo.cn/high.m3u8?channel-id=bstvod&Contentid=4007432526
```

一个常见的二级 m3u8 示例文件，代码如下：

```
//chapter5/hls.secondLevel.sample.m3u8
#EXTM3U
#EXT-X-VERSION:1
#EXT-X-TARGETDURATION:11
#EXT-X-MEDIA-SEQUENCE:19674922
#EXT-X-PROGRAM-DATE-TIME:2019-03-28T04:33:40Z
#EXTINF:10,
19674922.ts?
#EXT-X-PROGRAM-DATE-TIME:2019-03-28T04:33:50Z
#EXTINF:10,
19674923.ts?
#EXT-X-PROGRAM-DATE-TIME:2019-03-28T04:34:00Z
#EXTINF:10,
19674924.ts?
```

Master Playlist 是指一级 m3u8；Media Playlist 是指二级 m3u8，携带 TS 分片 URL 的 m3u8；Media Segment 是指 TS 分片；Attribute Lists 是指属性列表，是一个用逗号分隔的 attribute/value 对列表，格式为 AttributeName=AttributeValue。

tag 以#EXT 开头，主要分为以下几类。

1. Basic Tags

Basic Tags，可以用在 Media Playlist 和 Master Playlist 里面。EXTM3U 必须出现在文件的第一行，标识这是一个 Extended M3U Playlist 文件。EXT-X-VERSION 表示 Playlist 兼容的版本。

2. Media Segment Tags

Media Segment Tags，每个 Media Segment 通过一系列的 Media Segment Tags 跟一个 URI 来指定。有的 Media Segment Tags 只应用于下一个 Segment，有的则是应用于所有下面的 Segments。一个 Media Segment Tag 只能出现在 Media Playlist 里面。

(1) EXTINF：用于指定 Media Segment 的 duration。

(2) EXT-X-BYTERANGE：用于指定 URI 的 sub-range。

(3) EXT-X-DISCONTINUITY：表示后续分片属性发生变化，如文件格式/编码/序号。

(4) EXT-X-KEY：表示 Media Segment 已加密，该值用于解密。

(5) EXT-X-MAP：表示 Media Segment 的头部信息，例如 PAT/PMT 或者 WebVTT 头。

(6) EXT-X-PROGRAM-DATE-TIME：和 Media Segment 的第 1 个 sample 一起来确定时间戳。

3. Media Playlist Tags

Media Playlist Tags，用于描述 Media Playlist 的全局参数。同样地，Media Playlist Tags 只能出现在 Media Playlist 里面。

(1) EXT-X-TARGETDURATION：用于指定最大的 Media Segment Duration。

(2) EXT-X-MEDIA-SEQUENCE：用于指定第 1 个 Media Segment 的序号。

(3) EXT-X-DISCONTINUITY-SEQUENCE：用于不同的 Variant Stream 之间同步。

(4) EXT-X-ENDLIST：表示 Media Playlist 结束。

(5) EXT-X-PLAYLIST-TYPE：可选，指定整个 Playlist 的类型。

(6) EXT-X-I-FRAMES-ONLY：表示每个 Media Segment 均为 I-frame。

4. Master Playlist Tags

Master Playlist Tags，用于定义 Variant Streams、Renditions 和其他显示的全局参数。Master Playlist Tags 只能出现在 Master Playlist 中。

(1) EXT-X-MEDIA：用于关联同一个内容的多个 Media Playlist 的多种翻译。

(2) EXT-X-STREAM-INF：用于指定下级 Media Playlist 的相关属性。

(3) EXT-X-I-FRAME-STREAM-INF：与 EXT-X-STREAM-INF 类似，但指向的下级 Media Playlist 包含 Media Segment 均为 I-frame。

(4) EXT-X-SESSION-DATA：可以随意存放一些 session 数据。

5. Media or Master Playlist Tags

Media or Master Playlist Tags，这里的 Tags 可以出现在 Media Playlist 或者 Master Playlist 中。但是如果同时出现在同一个 Master Playlist 和 Media Playlist 中，则必须为相同值。

(1) EXT-X-INDEPENDENT-SEGMENTS：表示每个 Media Segment 可以独立解码。

(2) EXT-X-START：标识一个优选的点来播放这个 Playlist。

5.4 TS 与 PS 格式简介

据传输媒体的质量不同，MPEG-2 中定义了两种复合信息流，即传送流(Transport Stream，TS)和节目流(Program Stream，PS)。TS 流与 PS 流的区别在于 TS 流的包结构是固定长度的，而 PS 流的包结构是可变长度的。PS 包与 TS 包在结构上的这种差异，导致了它们对传输误码具有不同的抵抗能力，因而应用的环境也有所不同。TS 码流由于采用了固定长度的包结构，当传输误码破坏了某一 TS 包的同步信息时，接收机可在固定的位置检测它后面包中的同步信息，从而恢复同步，避免了信息丢失，而 PS 包由于长度是变化的，一旦某一 PS 包的同步信息丢失，接收机无法确定下一包的同步位置，这就会造成失步，导致严重的信息丢失，因此，在信道环境较为恶劣或传输误码较高时，一般采用 TS 码流，而在信道环境较好和传输误码较低时，一般采用 PS 码流。由于 TS 码流具有较强的抵抗传输误码的能力，因此目前在传输媒体中进行传输的 MPEG-2 码流基本上采用了 TS 码流的包格式。MPEG2-PS 主要应用于存储具有固定时长的节目，如 DVD 电影，而 MPEG-TS 则主要应用

于实时传送的节目，例如实时广播的电视节目。

5.4.1 ES、PES、PS、TS

MPEG是活动图像专家组(Moving Picture Expert Group)的缩写，于1988年成立。目前MPEG已颁布了3个活动图像及声音编码的正式国际标准，分别称为MPEG-1、MPEG-2和MPEG-4，而MPEG-7和MPEG-21仍在研究中。

MPEG-2是运动图像专家组(MPEG)组织制定的视频和声频有损压缩标准之一，它的正式名称为“基于数字存储媒体运动图像和语音的压缩标准”。与MPEG-1标准相比，MPEG-2标准具有更高的图像质量、更多的图像格式和传输码率的图像压缩标准。MPEG-2标准不是MPEG-1的简单升级，而是在传输和系统方面做了更加详细的规定和进一步的完善。它是针对标准数字电视和高清晰电视在各种应用下的压缩方案，传输速率为3Mb/s～10Mb/s。MPEG-2标准目前分为9个部分，统称为ISO/IEC 13818国际标准。各部分的内容描述如下。

(1) 第1部分，ISO/IEC 13818-1，System：系统，描述多个视频、声频和数据基本码流合成传输码流和节目码流的方式。

(2) 第2部分，ISO/IEC 13818-2，Video：视频，描述视频编码方法。

(3) 第3部分，ISO/IEC 13818-3，Audio：声频，描述与MPEG-1声频标准反向兼容的声频编码方法。

(4) 第4部分，ISO/IEC 13818-4，Compliance：符合测试，描述测试一个编码码流是否符合MPEG-2码流的方法。

(5) 第5部分，ISO/IEC 13818-5，Software：软件，描述了MPEG-2标准的第1、2、3部分的软件实现方法。

(6) 第6部分，ISO/IEC 13818-6，DSM-CC：数字存储媒体-命令与控制，描述交互式多媒体网络中服务器与用户间的会话信令集。

以上6个部分均已获得通过，成为正式的国际标准，并在数字电视等领域中得到了广泛的实际应用。此外，MPEG-2标准还有3个部分，其中第7部分规定不与MPEG-1声频反向兼容的多通道声频编码；第8部分现已停止；第9部分规定了传送码流的实时接口。

下面介绍几个重要概念，包括ES、PES、PTS与DTS、PS、TS等。

(1) ES，即原始流(Elementary Streams)，是直接从编码器出来的数据流，可以是编码过的视频数据流(H.264/MJPEG等)、声频数据流(AAC/AC3/MP3)或其他编码数据流的统称。ES流经过PES打包器之后，被转换成PES包。ES是只包含一种内容的数据流，如只含视频或只含声频等，打包之后的PES也是只含一种性质的ES，如只含视频ES的PES，或只含声频ES的PES等。每个ES都由若干个存取单元(Access Unit，AU)组成，每个视频AU或声频AU都由头部和编码数据两部分组成，1个AU相当于编码的1幅视频图像或1个声频帧，也可以说，每个AU实际上是编码数据流的显示单元，即相当于解码的1幅视频图像或1个声频帧的取样。

(2) PES,即分组的 ES(Packetized Elementary Stream),是用来传递 ES 的一种数据结构。PES 流是 ES 流经过 PES 打包器处理后形成的数据流,在这个过程中完成了将 ES 流分组、打包、加入包头信息等操作(对 ES 流的第一次打包)。PES 流的基本单位是 PES 包。PES 包由包头和 Payload 组成。

(3) PTS 与 DTS,PTS 即显示时间标记(Presentation Timestamp)表示显示单元出现在系统目标解码器(H.264、MJPEG 等)的时间。DTS 即解码时间标记(Decoding Time Stamp)表示将存取单元的全部字节从解码缓存器移走的时间。PTS/DTS 是打在 PES 包的包头里面的,这两个参数是解决音视频同步显示,防止解码器输入缓存上溢或下溢的关键。每个 I(关键帧)、P(预测帧)、B(双向预测帧)帧的包头都有一个 PTS 和 DTS,但 PTS 与 DTS 对于 B 帧不一样;对于 I 帧和 P 帧,显示前一定要存储于视频解码器的重新排序缓存器中,经过延迟(重新排序)后再显示,所以一定要分别标明 PTS 和 DTS。

(4) PS,即节目流(Program Stream),由 PS 包组成,而一个 PS 包又由若干个 PES 包组成(到这里,ES 经过了两层的封装)。PS 包的包头中包含了同步信息与时钟恢复信息。1 个 PS 包最多可包含具有同一时钟基准的 16 个视频 PES 包和 32 个声频 PES 包。

(5) TS,即传输流(Transport Stream),由定长的 TS 包组成(188B),而 TS 包是对 PES 包的一个重新封装(到这里,ES 也经过了两层的封装)。PES 包的包头信息依然存在于 TS 包中。单一性是指 TS 流的基本组成单位,是长度为 188B 的 TS 包。混合性是指 TS 流由多种数据组合而成,1 个 TS 包中的数据可以是视频数据、声频数据、填充数据、PSI/SI 表格数据等(由唯一的 PID 对应)。

5.4.2 PS/TS 编码基本流程

从 ES 到 PES 再到 PS/TS 的基本编码流程如图 5-5 所示。

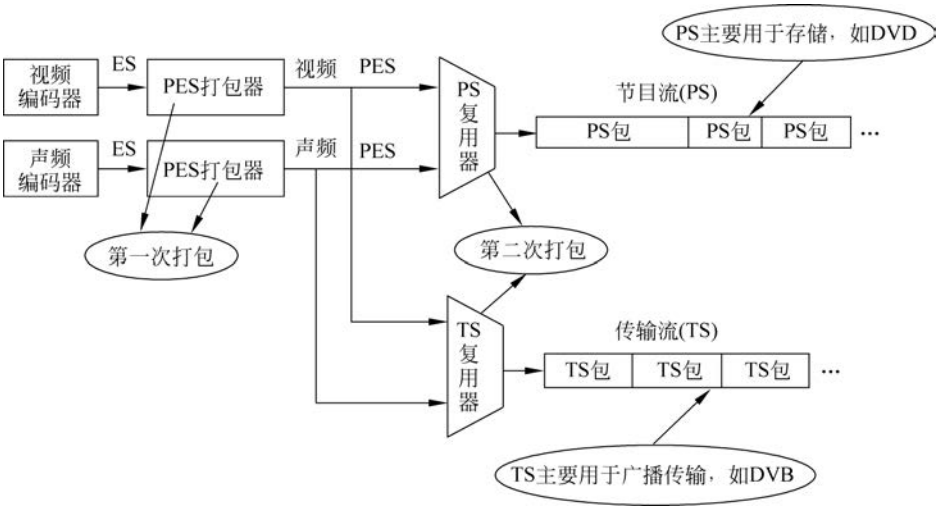


图 5-5 ES/PES/TS 编码流程

(1) A/D转换后,通过MPEG-2压缩编码得到的ES基本流。这个数据流很大,并且只是I、P、B的这些视频帧或声频取样信息。

(2) 通过PES打包器,打包并在每个帧中插入PTS/DTS标志,变成PES。原来是流的格式,现在成了数据包的分割形式。

(3) PES根据需要打包成PS或TS包进行存储(DVD)或传输(DVB)。因每路音/视频只包含一路的编码数据流,所以每路PES也只包含相应的数据流。

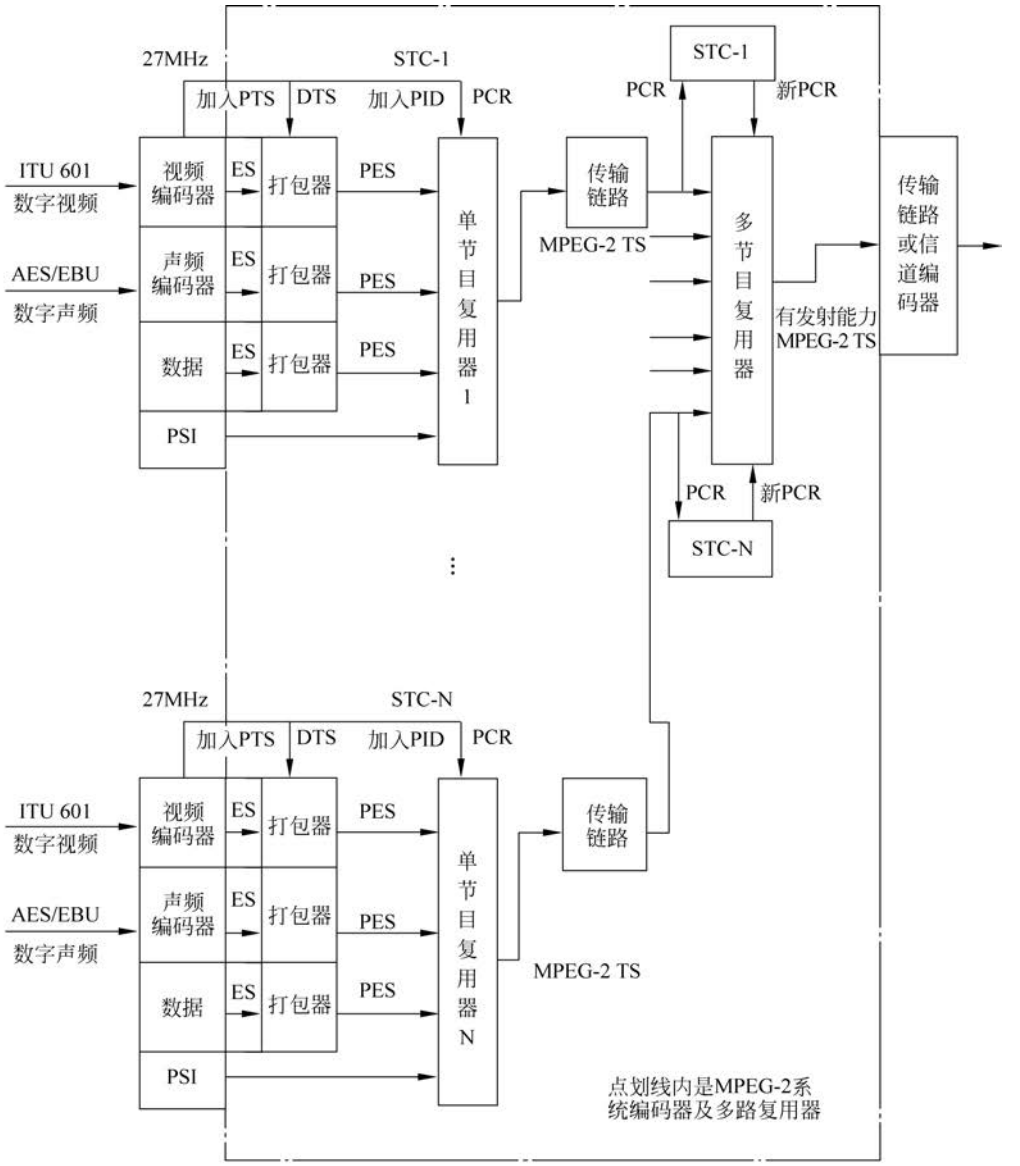
5.4.3 PS/TS码流小结

MPEG-2作为一个数字视声频的一种压缩标准一直被广泛地运用于多媒体、数字存储及数字传输(如数字电视)等领域。其规范主要包括声频编码、视频编码、系统、数字存储规范、复用和测试等几个部分。其中音、视频和系统(音视频同步)为主要部分,解决音视频的压缩问题并提供一种不同码流间的复用规范。根据传输媒体质量的不同,MPEG-2中定义了两种复合信息流,即传送流(TS)和节目流(PS)。在MPEG-2系统中,信息复合/分离的过程称为系统复接/分接,由视频、声频的ES流和辅助数据复接生成的用于实时传输的标准信息流(例如实时广播的电视节目)称为MPEG-2传送流(MPEG2-TS),而MPEG-2节目流(MPEG2-PS)主要应用于存储具有固定时长的节目,如DVD电影。

TS流与PS流的区别在于TS流的包结构是固定长度的,而PS流的包结构是可变长度的。PS包与TS包在结构上的这种差异,导致了它们对传输误码具有不同的抵抗能力,因而应用的环境也有所不同。TS码流由于采用了固定长度的包结构,当传输误码破坏了某一TS包的同步信息时,接收机可在固定的位置检测它后面包中的同步信息,从而恢复同步,避免了信息丢失,而PS包由于长度是可变化的,一旦某一PS包的同步信息丢失,接收机无法确定下一包的同步位置,这就会造成失步,导致严重的信息丢失,因此,在信道环境较为恶劣或传输误码较高时,一般采用TS码流,而在信道环境较好和传输误码较低时,一般采用PS码流。由于TS码流具有较强的抵抗传输误码的能力,因此目前在传输媒体中进行传输的MPEG-2码流基本上采用了TS码流的包。节目流主要用于误码相对较低的演播室和数字存储(如DVD)中;传输流主要用于传输中,它有固定长度的明显特点。这种数据结构运用于数字视频广播(Digital Video Broadcasting,DVB)的传输层中。

ES流是音、视频信号经过编码器之后或数据信号的基本码流。ES是只包含一种内容的数据流,如只含视频或只含声频等,打包之后的PES也是只含一种性质的ES,如只含视频的ES的PES或只含声频ES的PES。PES(Packetized Elementary Stream)是打包的基本码流,ES经过打包后的码流,其长度可变。视频一般一帧一个包,声频长度一般不超过64KB。

ES、PES、PS、TS的关系,如图5-6所示。



- PES-Packetized Elementary Stream(打包基本流)
- STC-System Time Clock(系统时钟)
- PCR-Program Clock Reference(节目时钟基准)
- PSI-Program Specific Information(节目专用信息)
- PID-Packet Identifier(包识别)
- PTS-Presentation Time Stamp(显示时间标记)
- EBU-European Broadcasting Union(欧洲广播联盟)
- DTS-Decoding Time Stamp(解码时间标记)
- AES-Audio Engineering Society(声频工程学会)
- ITU-International Telecommunications Union(国际电信联盟)

图 5-6　ES、PES、TS、PS 关系框架图

5.5 TS 码流详细讲解

学习数字电视机顶盒的开发，从 MPEG-2 到 DVB 会出现很多数据结构，如 PAT、PMT、CAT 等。数字电视机顶盒接收的是一段段的码流，称为传输流（Transport Stream，TS），每个 TS 流都携带一些信息，如 Video、Audio、PAT、PMT 等信息，因此，首先需要了解 TS 流是什么，以及 TS 流是怎样形成、有着怎样的结构，下面开始详细解释这些内容。

ES 流是基本码流，不分段的声频、视频或其他信息的连续码流。PES 流把基本流 ES 分割成段，并加上相应头文件打包形成的打包基本码流。PS 流将具有共同时间基准的一个或多个 PES 组合（复合）而成单一数据流（用于播放或编辑系统，如 m2p）。TS 流将具有共同时间基准或独立时间基准的一个或多个 PES 组合（复合）而成单一数据流（用于数据传输）。由于 TS 码流具有较强的抵抗传输误码的能力，因此目前在传输媒体中进行传输的 MPEG-2 码流基本上采用了 TS 码流的包。

TS 流的产生过程，如图 5-7 所示。

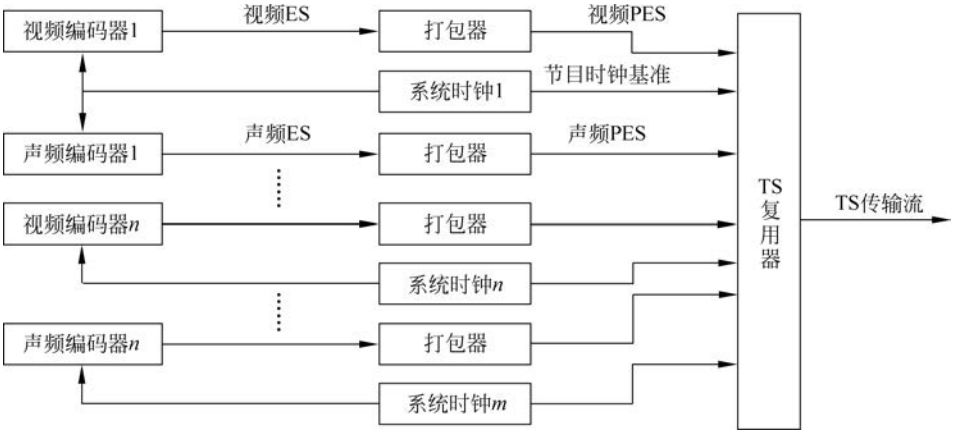


图 5-7 TS 流的产生过程

可以看出，视频 ES 和声频 ES 通过打包器和共同或独立的系统时间基准形成一个个 PES，通过 TS 复用器复用形成传输流。注意这里的 TS 流是位流格式，也就是说 TS 流是可以按位读取的。

5.5.1 TS 包格式

TS 流是基于 Packet 的位流格式，每个包是 188B（或 204B，在 188B 后加上了 16B 的 CRC 校验数据，其他格式相同）。整个 TS 流组成形结构如图 5-8 所示。

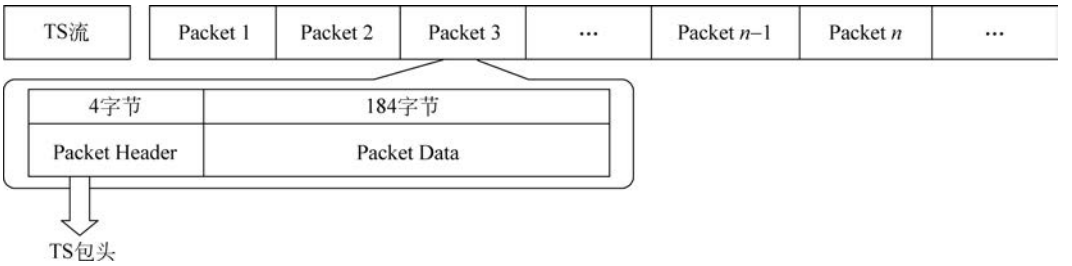


图 5-8　TS 流与 TS 包

TS Packet Header(包头)字段结构,代码如下:

```
//chapter5/ts.packet.header.txt
struct ts_header{
    char syn_Byte:8;                          //包头同步字节,0x47
    char transport_error_indicator:1;         //传送数据包差错指示器
    char payload_unit_start_indicator:1;      //有效净荷单元开始指示器
    char transport_priority:1;                //传送优先级
    int PID:13;                               //包 ID
    char transport_scrambling_control:2;      //传送加扰控制
    char adaptation_field_control:2;          //调整字段控制
    char continuity_counter:4;                //连续计数器 0～15
};
```

(1) syn_Byte:值为 0x47,是 MPEG-2TS 的传送包标识符。

(2) transport_error_indicator:值为 1 时,表示相关的传送包中至少有一个不可纠正的错误位,只有错误位纠正后,该位才能置 0。

(3) payload_unit_start_indicator:表示 TS 包的有效净荷带有 PES 或 PSI 数据的情况;当 TS 包的有效净荷带有 PES 包数据时,payload_unit_start_indicator 为 1,表示 TS 包的有效净荷以 PES 包的第 1 字节开始;payload_unit_start_indicator 为 0,表示 TS 包的开始不是 PES 包。

(4) 当 TS 包带有 PSI 数据时,payload_unit_start_indicator 为 1,表示 TS 包带有 PSI 部分的第 1 字节,即第 1 字节带有指针 pointer_field,payload_unit_start_indicator 为 0,表示 TS 包不带有一个 PSI 部分的第 1 字节,即在有效净荷中没有指针 pointer_field;对于空包的包,play_unit_start_indicator 应该置为 0。

(5) transport_priority:置 1 表示相关的包比其他具有相同 PID 但 transport_priority 为 0 的包有更高的优先级。

(6) PID 是 TS 流中唯一识别标志,Packet Data 是什么内容由 PID 决定。如果一个 TS 流中的 1 个 Packet 的 Packet Header 中的 PID 是 0x0000,则这个 Packet 的 Packet Data 是 DVB 的 PAT 表,而非其他类型数据(如 Video、Audio)或其他业务信息。

一些 TS 表的 PID 值是固定的,不允许用于更改,如表 5-1 所示。

表 5-1 TS 表的名称及对应的 PID 值

表	PID 值	表	PID 值
PAT	0x0000	EIT/ST	0x0012
CAT	0x0001	RST/ST	0x0013
TSDT	0x0002	TDT/TOT/ST	0x0014

TS 流最经典的应用是平时生活中的数字高清电视。电视码流是 TS 封装格式的码流，电视码流发送过来后，就会由机顶盒进行解封装和解码，然后传给电视机进行播放。这里就有一个问题，例如看电视，有很多的频道和节目，那么对应码流是怎么区分的呢？TS 流引入了 PAT 和 PMT 两张表格的概念来解决这个问题。

TS 流是以 188B 为一包，可以称为 TS Packet。这个 TS Packet 有可能是音视频数据，也有可能是表格(PAT/PMT/...)。举例说明，TS 流的包顺序，代码如下：

```
PAT,PMT,DATA,DATA,,,,,,PAT,PMT,DATA,DATA,,,,,,
```

每隔一段时间，发送一张 PAT 表，紧接着发送一张 PMT 表，接着发送 DATA(音视频)数据。PAT 表格里面包含所有 PMT 表格的信息，一个 PMT 表格对应一个频道，例如中央电视台综合频道，而一个 PMT 里面包含所有节目的信息，例如 CCTV-1 到 CCTV-14。在实际情况中有很多频道，所以 PMT 表格可不止一张，有可能是如下形式，代码如下：

```
PAT,PMT,PMT,PMT,,,DATA,DATA,,,,PAT,PMT,PMT,,,DATA,DATA
```

除了这个设定外，每个频道或节目都有自己的标识符(PID)，这样当得到一个 DATA，解析出里面的 PID，就知道是什么节目了，并且也知道所属频道是什么了。在看电视时，会收到所有节目的 DATA，当选择某个节目时，机顶盒会把这个节目的 DATA 单独过滤出来，其他的舍弃。

TS Packet 的长度是 188B，分为 TS Header 和 TS Body。其中 TS Header 里面会有个 PID 字段，标识着当前 TS Body 的类型。TS Body 有可能是表格，也有可能是 DATA，表格很好理解，接下来讲解 DATA 的结构。DATA 包其实是 PES 包，而 PES 包是对 ES 的封装，PES 包分为 PES 头加 ES。这里的 ES 是原始流，是指经过压缩后的 H.264、AAC 等格式的音视频数据。

这里介绍一下帧数据、PES 包、TS Packet 包的对应关系。一帧数据封装成一个 PES 包(含 PES 头和 ES)，这个 PES 包如果小于 188B，则一个 TS Packet 就可以放下。最终 TS Packet 包的格式是 TS Header+Padding+PES Packet(PES Header+ES)。Padding 为填充字节的意思，如果 TS Header 加上 PES 包后仍不满 188B，则此时需要填充，使其凑满 188B 后再发送。

视频帧是很大的，往往大于 188B，同样需要把一个视频帧放入一个 PES 包，然后分别放在几个 TS Packet 包即可。PES 头加上这些部分 ES，是一个 PES 包，伪代码如下：

```
第 1 个 TS Packet:TS Header + PES 头 + 部分 ES
第 2 个 TS Packet:TS Header + 部分 ES
...
最后一个 TS Packet:TS Header + 填充字节 + 部分 ES
```

5.5.2 TS 码流分析工具

Tsr(TS 码流分析工具)是一款非常好用的解析软件,是专门为 TS 码流所设计的。读者可通过它对多种 TS 流文件进行查询,进而了解其中所包含的详细信息,以便用户后续处理。该软件的启动界面如图 5-9 所示。

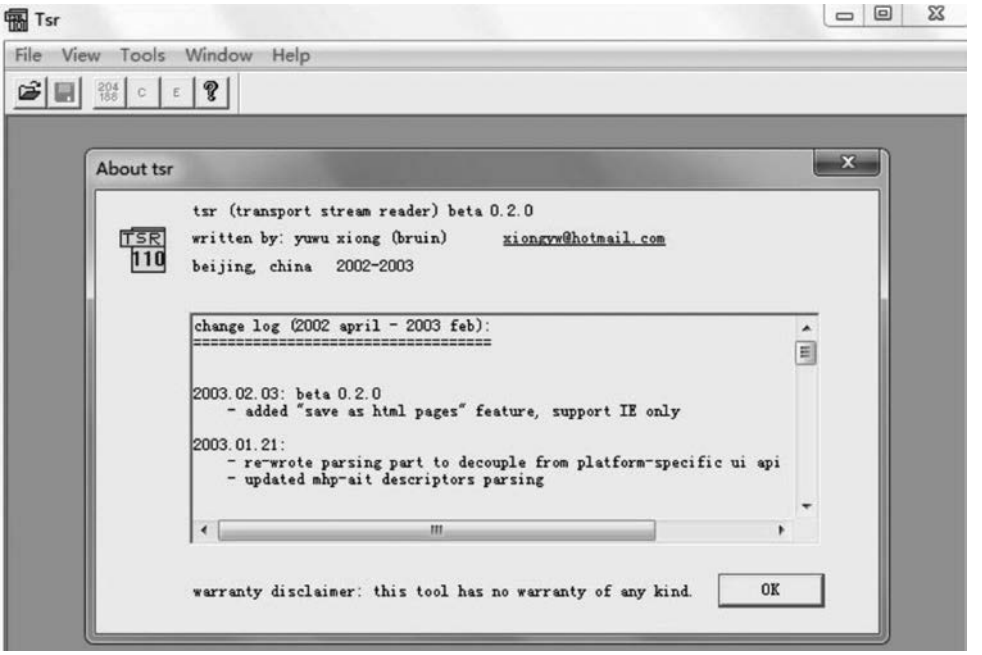


图 5-9 TS 码流分析工具

TS 流是分包发送的,每个包长为 188B。在 TS 流里可以填入很多类型的数据,如视频、声频、自定义信息等。它的包的结构:包头为 4B,负载为 184B。将一个 TS 视频文件拖曳到该软件中,即可自动分析 TS 格式,如图 5-10 所示。

5.5.3 TS 码流结构分析

MPEG-2 中规定 TS 传输包的长度为 188B,包头为 4B,负载为 184B,但通信媒介会为包添加错误校验字节,从而有了不同于 188B 的包长。

(1) 在 DVB 规定中,使用 204B 作为包长。通过调制器时,在每个传输包后增加了 16B 的里德所罗门前向纠错码,因而形成了 204B 的数据包,调制后总存在 204B 的数据包。调

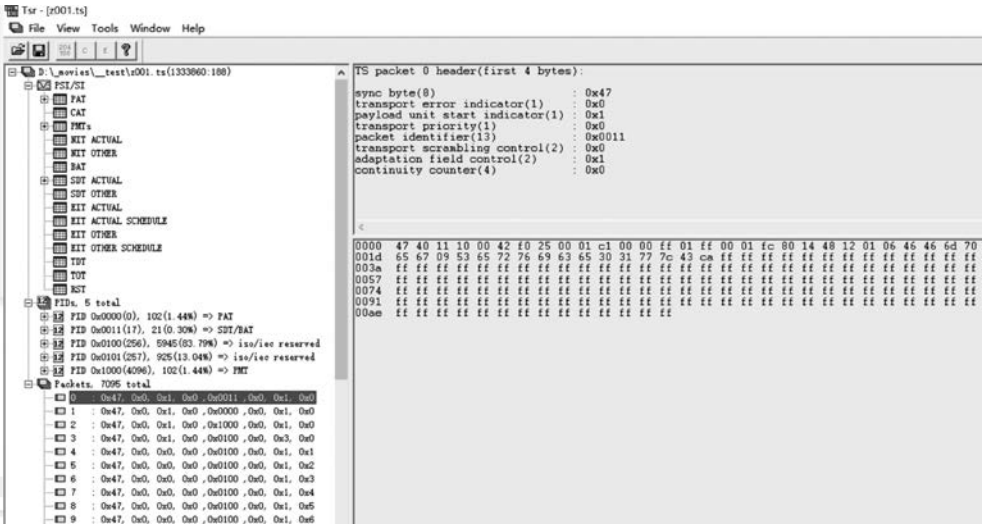


图 5-10　TS 文件的码流结构

制之前在复用器插入 RS 码或虚构的 RS 码。

(2) 在 ATSC 规定中,使用 208B 作为包长,即添加 20B 的 RS(Reed-Solomon)前向纠错码。与 DVB 不同,ATSC 规定 RS 码只能出现在调制的 TS 流中。所有的 TS 包都分为包头和净荷(负载、Payload)部分。TS 包中可以填入很多东西(填入的东西都被填到净荷部分),包括视频、声频、数据(包括 PSI、SI 及其他任何形式的数据)。TS 包的语法结构如图 5-11 所示。

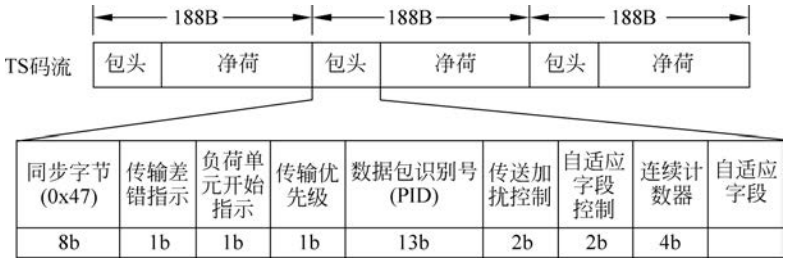


图 5-11　传输包(TS)码流语法结构

1. TS 包头

TS 包的包头提供关于传输方面的信息,包括同步、有无差错、有无加扰、PCR(节目参考时钟)等标志。TS 包的包头长度不固定,前 32b(4 字节)是固定的,后面可能跟有自适应字段(适配域),因此 32b(4 字节)是最小包头。TS 包也可以是空包,空包用来填充 TS 流,可能在重新进行多路复用时被插入或删除。

TS 包头字段结构如下:

(1) sync_Byte (同步字节)：固定为 0100 0111(0x47)，该字节由解码器识别，使包头和有效负载可相互分离。

(2) transport_error_indicator(传输差错指示)：1 表示在相关的传输包中至少有一个不可纠正的错误位。当被置为 1 后，在错误被纠正之前不能重置为 0。

(3) payload_unit_start_indicator(开始指示)：为 1 时，在前 4 字节之后会有一个调整字节，其数值为后面调整字段的长度 length，因此有效载荷开始的位置应再偏移 1＋[length]字节。为 0 时，则在 4 字节之后，直接是有效载荷。

(4) transport_priority(传输优先级)：1 表明优先级比其他具有相同 PID，但此位没有被置为 1 的分组高。

(5) PID：指示存储与分组有效负载中数据的类型。PID 值 0x0000～0x000F 保留。其中 0x0000 为 PAT 保留；0x0001 为 CAT 保留；0x1fff 为分组保留，即空包。

(6) transport_scrambling_control(加扰控制)：表示 TS 流分组有效负载的加密模式。空包为 00，如果传输包的包头中包括调整字段，则不应被加密。

(7) adaptation_field_control(自适应字段控制)：表示包头是否有调整字段或有效负载。00 表示为 ISO/IEC 未来使用保留；01 表示仅含有效载荷，无调整字段；10 表示无有效载荷，仅含调整字段；11 表示调整字段后为有效载荷，调整字段中的前一字节表示调整字段的长度 length，有效载荷开始的位置应再偏移[length]字节。空包应为 10。

(8) continuity_counter(连续计数器)：随着每个具有相同 PID 的 TS 流分组而增加，当它达到最大值后又回复到 0。范围为 0～15。

(9) adaptation_field(自适应字段)：根据自适应控制字段填充负载。

2. 节目专用信息 PSI

在系统复用时，视频、声频的 ES 流需进行打包形成视频、声频的 PES 流，辅助数据(如图文电视信息)不需要打成 PES 包。PES 包非定长，声频的 PES 包小于或等于 64K，视频一般为一帧一个 PES 包。一帧图像的 PES 包通常要由许多个 TS 包来传输。

MPEG-2 中规定，一个 PES 包必须由整数个 TS 包来传输。如果承载一个 PES 包的最后一个 TS 包没能装满，则用填充字节来填满；目前一个新的 PES 包形成时，需用新的 TS 包来开始传输。

节目专用信息(Program Specific Information，PSI)，用来管理各种类型的 TS 数据包，需要有些特殊的 TS 包来确立各个 TS 数据包之间的关系。这些特殊的 TS 包里所包含的信息是节目专用信息。在不同的标准中它有不同的名字：MPEG-2 中称为 PSI；DVB 标准根据实际需要，对 PSI 扩展，称为 SI 信息；ATSC 标准中称为 PSIP 信息。

MPEG-2 中，规定的对 PSI 信息的描述方法有以下几种：

(1) 表(Table)，对节目信息的结构性描述，包括节目关联表(Program Association Table，PAT)、节目映射表(Program Map Tables，PMT)、条件接收表(Conditional Access Table，CAT)、加密解密、网络信息表(Network Information Table，NIT)、传送流描述表(Transport Stream Description Table，TSDT)等。其中，PAT、CAT、NIT 的 PID 值是固定

的，分别为 0x0000、0x0001、0x0010，而 PMT 的 PID 值是在 PAT 中定义的。

(2) 节(Section)，将表格的内容映射到 TS 流中，包括专用段 Private_section 等。

(3) 描述符(Descriptor)，提供有关节目构成(视频流、声频流、语言、层次、系统时钟和码率等多方面)的信息；ITU-T Rec. H. 222. 0 | ISO /IEC 13818-1 中定义的 PSI 表可被分成一段或多段置于传输流分组中。一段是一个语法结构，用来将 ITU-T Rec. H. 222. 0 | ISO /IEC 13818-1 中定义的 PSI 表映射到传输流分组中。

5.5.4 PAT 及 PMT 表格式

下面介绍两个重要的表结构，包括 PAT 和 PMT。

1. PAT

TS 流中包含一个或者多个 PAT。PAT 由 PID 为 0x0000 的 TS 包传送，其作用是为复用的每一路传送流提供所包含的节目和节目编号，以及对应节目的 PMT 的位置，即 PMT 的 TS 包的 PID 值，同时还提供 NIT 的位置，即 NIT 的 TS 包的 PID 的值。TS 码流解析从 PAT 开始。PAT 定义了当前 TS 流中所有的节目，其 PID 为 0x0000，它是 PSI 的根节点，要查找节目必须从 PAT 开始查找。PAT 主要包含频道号码和每个频道对应的 PMT 的 PID 号码，这些信息在处理 PAT 时会保存起来，以后会使用这些数据。下面给出 PAT 的字段结构，代码如下：

```
//chapter5/ts_pat_program.txt
typedef struct TS_PAT_Program
{
    unsigned program_number :16;                    //节目号
    unsigned program_map_PID:13;                    //节目映射表的 PID,每个节目对应一个
}TS_PAT_Program;
```

PAT 数据包分为两部分，包括 PAT 数据包头部(前 8B)和循环部分，代码如下：

```
//chapter5/ts_program_association_section.txt
/*头部部分 8 字节*/
program_association_section()
{
    unsigned table_id           : 8;            //固定为 0x00,标志是该表为 PAT
    unsigned section_syntax_indicator     : 1;  //段语法标志位,固定为 1
    unsigned '0'                : 1;            //0
    unsigned reserved_1         : 2;            //保留位
    unsigned section_length     : 12;           //段长度字节

    //该传输流的 ID,区别于一个网络中其他多路复用的流
    unsigned transport_stream_id: 16;
    unsigned reserved_2         : 2;            //保留位
    unsigned version_number     : 5;            //范围为 0～31,表示 PAT 的版本号
    unsigned current_next_indicator: 1;         //发送的当前 PAT 是否有效
```

```
    //PAT 可能分为多段传输,第一段为 00,以后每个分段加 1,最多可能有 256 个分段
    //给出 section 号,在 sub_table 中,第 1 个 section 的 section_number 为"0x00"
    //每增加一个 section,section_number 加 1
    unsigned section_number       : 8;              //分段的号码

    //最后一个分段的号码,sub_table 中最后一个 section 的 section_number
    unsigned last_section_number  : 8;

    /* 循环部分 4 个 Byte */
    for(i = 0;i < N;i++)
    {
    program_number                :16;              //节目号
    Reserved                      :3;               //保留位
    //网络信息表(NIT)的 PID,节目号为 0 时对应的 PID 为 network_PID
    //其余情况是 program_map_PID(PMT 的 PID)
    network_id 或 program_map_PID :13;
    }
    CRC_32                        :32;              //校验码
}
```

各个字段的含义如下。

(1) table_id：固定为 0x00,标志该表是 PAT。

(2) section_syntax_indicator：段语法标志位,固定为 1。

(3) section_length：表示此字节后面有用的字节数,包括 CRC32。

(4) 节目套数：(section_length－9)/4。

(5) transport_stream_id：表示该 TS 流的 ID,区别于同一个网络中其他多路复用流。

(6) version_number：表示 PAT 的版本号。

(7) current_next_indicator：表示发送的 PAT 是当前有效还是下一个有效。

(8) section_number：表示分段的号码。PAT 可能分多段传输,第一段为 0,以后每个分段加 1,最多可能有 256 个分段。

(9) last_section_number：表示 PAT 最后一个分段的号码。

(10) program_number：节目号。

(11) network_PID：网络信息表(NIT)的 PID,节目号为 0 时对应的 ID 为 network_PID。

(12) program_map_PID：节目映射表(PMT)的 PID,节目号为大于或等于 1 时,对应的 ID 为 program_map_PID。一个 PAT 中可以有多个 program_map_PID。

(13) CRC_32：32 位字段,CRC32 校验码 Cyclic Redundancy Check。

使用 Tsr 码流分析工具,可以很方便地解析出对应的 PAT 结构,如图 5-12 所示。

PAT 的解析函数,代码如下：

图 5-12　PAT 结构

```
//chapter5/ts.pat.analysis.c
//TS_PAT 结构体参考 program_association_section()
//C 语言的位操作符
int adjust_PAT_table( TS_PAT * packet, unsigned char * buffer)
{
    packet->table_id                    = buffer[0];
    packet->section_syntax_indicator    = buffer[1] >> 7;
    packet->zero                        = buffer[1] >> 6 & 0x1;
    packet->reserved_1                  = buffer[1] >> 4 & 0x3;
    packet->section_length              = (buffer[1] & 0x0F) << 8 | buffer[2];

    packet->transport_stream_id         = buffer[3] << 8 | buffer[4];

    packet->reserved_2                  = buffer[5] >> 6;
    packet->version_number              = buffer[5] >> 1 & 0x1F;
    packet->current_next_indicator      = (buffer[5] << 7) >> 7;
    packet->section_number              = buffer[6];
    packet->last_section_number         = buffer[7];

    int len = 0;
    len = 3 + packet->section_length;
    packet->CRC_32 = (buffer[len-4] & 0x000000FF) << 24
  | (buffer[len-3] & 0x000000FF) << 16
  | (buffer[len-2] & 0x000000FF) << 8
  | (buffer[len-1] & 0x000000FF);
```

```
    int n = 0;
    ///循环次数
    for ( n = 0; n < packet->section_length - 12; n += 4 )
    {
        unsigned program_num = buffer[8 + n ] << 8 | buffer[9 + n ];
        packet->reserved_3      = buffer[10 + n ] >> 5;

        packet->network_PID = 0x00;
        if ( program_num == 0x00)
        {
            packet->network_PID = (buffer[10 + n ] & 0x1F) << 8 | buffer[11 + n ];

            TS_network_Pid = packet->network_PID;        //记录该 TS 流的网络 PID

            TRACE(" packet->network_PID % 0x /n/n", packet->network_PID );
        }
        else
        {
         TS_PAT_Program PAT_program;                     //队列
           PAT_program.program_map_PID =
               (buffer[10 + n] & 0x1F) << 8 | buffer[11 + n];
           PAT_program.program_number = program_num;
           packet->program.push_back( PAT_program );
           //向全局 PAT 节目数组中添加 PAT 节目信息
           TS_program.push_back( PAT_program );
        }
    }
    return 0;
}
```

在上述代码中，从 for 循环开始，描述了当前流中的频道数目和每个频道对应的 PMT 的 PID 值。解复用程序需要接收所有的频道号码和对应的 PMT 的 PID，并把这些信息在缓冲区中保存起来。在后部的处理中需要使用 PMT 的 PID。

2. PMT

PMT 在传送流中用于指示组成某一套节目的视频、声频和数据在传送流中的位置，即对应的 TS 包的 PID 值，以及每路节目的节目时钟参考(PCR)字段的位置。PMT 流结构，代码如下：

```
//chapter5/ts.TS_PMT_Stream.txt
typedef struct TS_PMT_Stream
{
unsigned stream_type: 8;              //指示特定 PID 的节目元素包的类型
unsigned elementary_PID: 13;          //该域指示 TS 包的 PID 值，包含相关的节目元素
unsigned ES_info_length: 12;          //前两位是 00，指示跟随其后的相关节目元素的字节数
unsigned descriptor;
}TS_PMT_Stream;
```

PMT 包含以下信息：

(1) 当前频道中包含的所有 Video 数据的 PID。

(2) 当前频道中包含的所有 Audio 数据的 PID。

(3) 和当前频道关联在一起的其他数据的 PID(如数字广播、数据通信等使用的 PID)。

只要处理了 PMT，就可以获取频道中所有的 PID 信息，如当前频道包含多少个 Video、共多少个 Audio 和其他数据，还能知道每种数据对应的 PID 值。如果要选择其中一个 Video 和 Audio 收看，则只需把要收看的节目的 Video PID 和 Audio PID 保存起来，在处理 Packet 的时候进行过滤便可实现。

PMT 表结构，代码如下：

```
//chapter5/ts.TS_program_map_section.txt
TS_program_map_section() {
    table_id                    :8;     //固定为 0x02 标识 PMT 表
    section_syntax_indicator    :1;     //固定为 0x01
    '0'                         :1;     //
    reserved                    :2;     //保留位
    section_length              :12     //该字段的头两位必为'00',剩余 10 位指定该分段的字节
//数,紧随 section_length 字段开始,并包括 CRC.此字段中的值应不超过 1021(0x3FD)
    program_number              :16     //指出 TS 流中 Program Map Section 的版本号
    reserved                    :2      //保留位
    version_number              :5      //指出 TS 流中 Program Map Section 的版本号
    current_next_indicator      :1      //当该位被置 1 时,当前传送的 Program Map Section 可用;
//当该位被置 0 时,指示当前传送的 Program Map Section 不可用,下一个 TS 流的 Program Map
//Section 有效
    section_number              :8      //固定为 0x00
    last_section_number         :8      //固定为 0x00
    reserved                    :3      //保留
    PCR_PID                     :13     //指明 TS 包的 PID 值,该 TS 包含 PCR 域
    //该 PCR 值对应于由节目号指定的对应节目
    //如果对于私有数据流的节目定义与 PCR 无关,这个域的值将为 0x1FFF
    reserved                    :4      //保留位
    program_info_length         :12     //节目信息长度.该字段的头两位必为'00',剩余 10 位指
//定紧随 program_info_length 字段的描述符的字节数
//(之后的是 N 个描述符结构,一般可以忽略,这个字段就代表描述符总的长度,单位是 Bytes)紧接
//着是频道内部包含的节目类型和对应的 PID

    for (i = 0; i < N; i++) {
        descriptor()
    }
    for (i = 0; i < N1; i++) {
        stream_type             :8      //流类型,标志是 Video、Audio,还是其他数据
        reserved                :3      //保留位
        elementary_PID          :13     //该节目的声频或视频 PID
        reserved                :4      //保留位
```

```
        ES_info_length      :12      //该字段的头两位必为'00',剩余 10 位指示紧随 ES_info_
//length 字段的相关节目元描述符的字节数
        for (i = 0; i < N2; i++) {
        descriptor()
    }
    }
    CRC_32                  :32
}
```

各个字段结构解释如下。

(1) table_id：固定为 0x02,标志该表是 PMT。

(2) section_syntax_indicator：对于 PMT 表,设置为 1。

(3) section_length：表示此字节后面有用的字节数,包括 CRC32。

(4) program_number：它指出该节目对应于可应用的 Program Map PID。

(5) version_number：指出 PMT 的版本号。

(6) current_next_indicator：当该位被置 1 时,当前传送的 Program Map Section 可用；当该位被置 0 时,指示当前传送的 Program Map Section 不可用,下一个 TS 流的 Program Map Section 有效。

(7) section_number：总是置为 0x00(因为 PMT 里表示一个 Service 的信息,一个 Section 的长度足够)。

(8) last_section_number：该域的值总是 0x00。

(9) PCR_PID：节目中包含有效 PCR 字段的传送流中的 PID。

(10) program_info_length：12b,前两位为 00。该域指出跟随其后对节目信息的描述符的字节数。

(11) stream_type：8b,指示特定 PID 的节目元素包的类型。该处 PID 由 elementary_PID 指定。

使用 Tsr 码流分析工具,可以很方便地解析出对应的 PMT 结构,如图 5-13 所示。

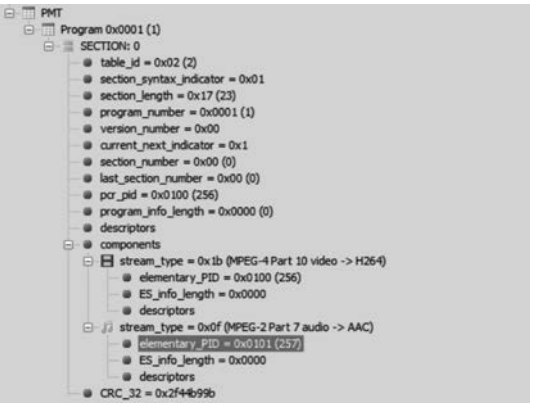


图 5-13　PMT 结构

PMT 的解析函数，代码如下：

```
//chapter5/ts_adjust_PMT_table.c
int adjust_PMT_table ( TS_PMT * packet, unsigned char * buffer )
{
    //读取各个字段
    packet->table_id                    = buffer[0];
    packet->section_syntax_indicator    = buffer[1] >> 7;
    packet->zero                        = buffer[1] >> 6 & 0x01;
    packet->reserved_1                  = buffer[1] >> 4 & 0x03;
    packet->section_length              = (buffer[1] & 0x0F) << 8 | buffer[2];
    packet->program_number              = buffer[3] << 8 | buffer[4];
    packet->reserved_2                  = buffer[5] >> 6;
    packet->version_number              = buffer[5] >> 1 & 0x1F;
    packet->current_next_indicator      = (buffer[5] << 7) >> 7;
    packet->section_number              = buffer[6];
    packet->last_section_number         = buffer[7];
    packet->reserved_3                  = buffer[8] >> 5;
    packet->PCR_PID                     = ((buffer[8] << 8) | buffer[9]) & 0x1FFF;

    PCRID = packet->PCR_PID;

    packet->reserved_4                  = buffer[10] >> 4;
    packet->program_info_length         = (buffer[10] & 0x0F) << 8 | buffer[11];
    //Get CRC_32
    int len = 0;
    len = packet->section_length + 3;
    packet->CRC_32                      = (buffer[len-4] & 0x000000FF) << 24
  | (buffer[len-3] & 0x000000FF) << 16
  | (buffer[len-2] & 0x000000FF) << 8
  | (buffer[len-1] & 0x000000FF);

    int pos = 12;
    //program info descriptor             //节目信息描述符
    if ( packet->program_info_length != 0 )
        pos += packet->program_info_length;
    //Get stream type and PID
    for ( ; pos <= (packet->section_length + 2 ) - 4; )
    {
    TS_PMT_Stream pmt_stream;             //流信息
    pmt_stream.stream_type = buffer[pos];
    packet->reserved_5 = buffer[pos+1] >> 5;
pmt_stream.elementary_PID = ((buffer[pos+1] << 8) | buffer[pos+2]) & 0x1FFF;
    packet->reserved_6 = buffer[pos+3] >> 4;
    pmt_stream.ES_info_length = (buffer[pos+3] & 0x0F) << 8 | buffer[pos+4];

  pmt_stream.descriptor = 0x00;           //描述符
```

```
    if (pmt_stream.ES_info_length != 0)
    {
     pmt_stream.descriptor = buffer[pos + 5];

     for( int len = 2; len <= pmt_stream.ES_info_length; len ++)
     {
      pmt_stream.descriptor = pmt_stream.descriptor << 8 | buffer[pos + 4 + len];
     }
     pos += pmt_stream.ES_info_length;
    }
    pos += 5;
    packet->PMT_Stream.push_back( pmt_stream );          //存储下来
    TS_Stream_type.push_back( pmt_stream );
      }
  return 0;
}
```

5.6 PS 码流详细讲解

PS 文件分为 3 层，包括 PS 层(Program Stream)、PES 层(Packet Elemental Stream)和 ES 层(Elementary Stream)。ES 层是音视频数据层，PES 层在音视频数据上加了时间戳等对数据帧的说明信息，PS 层在 PES 层上加入了数据流识别和传输的必要信息。

5.6.1 PS 码流结构

一个完整的 MPEG-2 文件是一个 PS 流文件，PS 码流结构如图 5-14 所示。

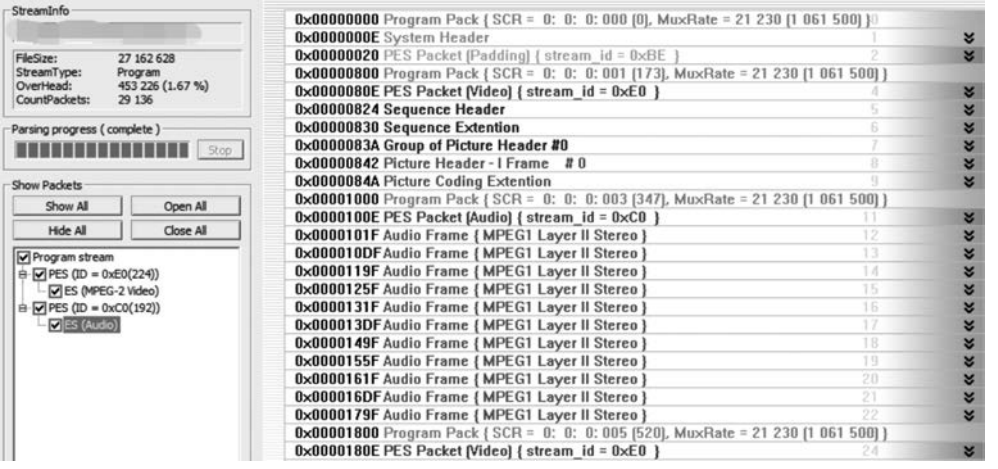


图 5-14 PS 码流结构

可以看出，整个文件分为 3 层。首先被分为一个个的 Program Pack，然后 Program Pack 里面包含了 Program Pack Header 和 PES 包，PES 包里又包含了 PES Header 和声频编码数据或视频编码数据。PS 流由很多个 PS 包组成，PS 包的结构，代码如下：

```
PS Header + SYS Header(I 帧) + PSM Header(I 帧) + PES Header + PES Packet
```

PS 流总是以 0x000001BA 开始，以 0x000001B9 结束，对于一个 PS 文件，有且只有一个结束码 0x000001B9，但对于网传的 PS 流，则没有结束码。

PS、PES 和 ES 的关系，如图 5-15 所示。

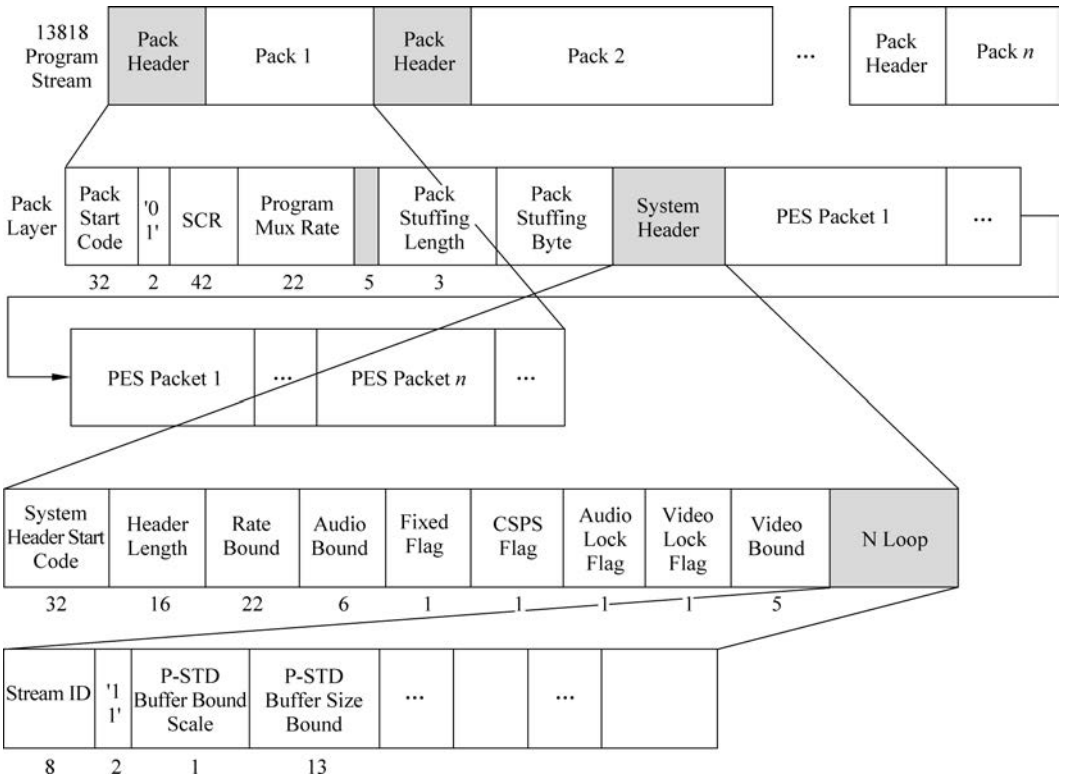


图 5-15 PS、PES、ES 的关系

PS 层主要由 Pack Header 和数据组成，Pack Header 中各个字段，如图 5-16 所示。其中，bslbf(bit string, left bit first)为比特串，左位在先；uimsbf(unsigned integer, most significant bit first)为无符号整数，高位在先。

Pack Header 结构体采用了位域来定义，代码如下：

```
//chapter/ps.pack_header.txt
struct pack_header //Pack Header 结构体
{
```

字段名称	位数	比特串
pack_header() {		
pack_start_code	32	bslbf
'01'	2	bslbf
system_clock_reference_base [32..30]	3	bslbf
marker_bit	1	bslbf
system_clock_reference_base [29..15]	15	bslbf
marker_bit	1	bslbf
system_clock_reference_base [14..0]	15	bslbf
marker_bit	1	bslbf
system_clock_reference_extension	9	uimsbf
marker_bit	1	bslbf
program_mux_rate	22	uimsbf
marker_bit	1	bslbf
marker_bit	1	bslbf
reserved	5	bslbf
pack_stuffing_length	3	uimsbf
for (i=0; i<pack_stuffing_length;i++) {		
stuffing_byte	8	bslbf
}		
if(nextbits()==system_header_start_code){		
system_header ()		
}		
}		

图 5-16　PS 包头结构

```
unsigned char pack_start_code[4];                    //起始码
unsigned char system_clock_reference_base21 : 2;     //参考时钟
unsigned char marker_bit : 1;
unsigned char system_clock_reference_base1 : 3;
unsigned char fix_bit : 2;
unsigned char system_clock_reference_base22;
unsigned char system_clock_reference_base31 : 2;
unsigned char marker_bit1 : 1;
unsigned char system_clock_reference_base23 : 5;
unsigned char system_clock_reference_base32;
unsigned char system_clock_reference_extension1 : 2;
unsigned char marker_bit2 : 1;
unsigned char system_clock_reference_base33 : 5;
unsigned char marker_bit3 : 1;
unsigned char system_clock_reference_extension2 : 7;
unsigned char program_mux_rate1;
unsigned char program_mux_rate2;
unsigned char marker_bit5 : 1;
unsigned char marker_bit4 : 1;
unsigned char program_mux_rate3 : 6;
unsigned char pack_stuffing_length : 3;
```

```
unsigned char reserved : 5;

pack_header()
{
    pack_start_code[0] = 0x00;          //起始码固定为 0x000001BA
    pack_start_code[1] = 0x00;
    pack_start_code[2] = 0x01;
    pack_start_code[3] = 0xBA;
    fix_bit = 0x01;
    marker_bit = 0x01;
    marker_bit1 = 0x01;
    marker_bit2 = 0x01;
    marker_bit3 = 0x01;
    marker_bit4 = 0x01;
    marker_bit5 = 0x01;
    reserved = 0x1F;
    pack_stuffing_length = 0x00;
    system_clock_reference_extension1 = 0;
    system_clock_reference_extension2 = 0;
}
//获取参考时钟
void getSystem_clock_reference_base(UINT64 &_ui64SCR)
{
    _ui64SCR = (system_clock_reference_base1 << 30) | (system_clock_reference_base21 << 28)
        | (system_clock_reference_base22 << 20) | (system_clock_reference_base23 << 15)
        | (system_clock_reference_base31 << 13) | (system_clock_reference_base32 << 5)
        | (system_clock_reference_base33);
}

void setSystem_clock_reference_base(UINT64 _ui64SCR)
{
    system_clock_reference_base1 = (_ui64SCR >> 30) & 0x07;
    system_clock_reference_base21 = (_ui64SCR >> 28) & 0x03;
    system_clock_reference_base22 = (_ui64SCR >> 20) & 0xFF;
    system_clock_reference_base23 = (_ui64SCR >> 15) & 0x1F;
    system_clock_reference_base31 = (_ui64SCR >> 13) & 0x03;
    system_clock_reference_base32 = (_ui64SCR >> 5) & 0xFF;
    system_clock_reference_base33 = _ui64SCR & 0x1F;
}

void getProgram_mux_rate(unsigned int &_uiMux_rate)
{
    _uiMux_rate = (program_mux_rate1 << 14) | (program_mux_rate2 << 6) | program_mux_rate3;
}

void setProgram_mux_rate(unsigned int _uiMux_rate)
```

```
    {
        program_mux_rate1 = (_uiMux_rate >> 14) & 0xFF;
        program_mux_rate2 = (_uiMux_rate >> 6) & 0xFF;
        program_mux_rate3 = _uiMux_rate & 0x3F;
    }
};
```

对于 DVD 而言，一般开始的 Pack 里面还有一个 System Header，如图 5-17 所示。

字段含义	位数	比特串
system_header(){		
system_header_start_code	32	bslbf
header_length	16	uimsbf
marker_bit	1	bslbf
rate_bound	22	uimsbf
marker_bit	1	bslbf
audio_bound	6	uimsbf
fixed_flag	1	bslbf
CSPS_flag	1	bslbf
system_audio_lock_flag	1	bslbf
system_video_lock_flag	1	bslbf
marker_bit	1	bslbf
video_bound	5	uimsbf
packet_rate_restriction_flag	1	bslbf
reserved_bits	7	bslbf
while (nextbits()=='1') {		
stream_id	8	uimsbf
'11'	2	bslbf
P-STD_buffer_bound_scale	1	bslbf
P-STD_buffer_size_bound	13	uimsbf
}		
}		

图 5-17　PS 的系统头结构

PES 层由编码的声频或视频数据(ES)加上 PES 头组成，PES 头主要通过 PTS 和 DTS 来提供音视频同步的信息。PES 头之后紧跟着的是编码的声频或视频数据，对于 DVD 而言，一个 Program Pack 的大小为 0x800，所以一帧 MPEG-2 视频被分在多个 PES 包里，不够一个包的就写在下一帧的第 1 个 Pack 里，或在 PES Header 后面填充 0xFF(PES_header_data_length 要加上填充的字节数)。

5.6.2　PS 码流的解析流程

解析 PS 码流的流程，需要经历几个步骤，包括读取源文件、查找起始码、解析 PS 头、解析 PS 系统头、解析 PSM、解析 PES 包等，如图 5-18 所示。

(1) 开辟缓存空间，从源码文件中读取指定大小的内容，如图 5-19 所示。

(2) 可能会存在冗余数据，需要逐字节往后查找，直到下一个起始码，如图 5-20 所示。

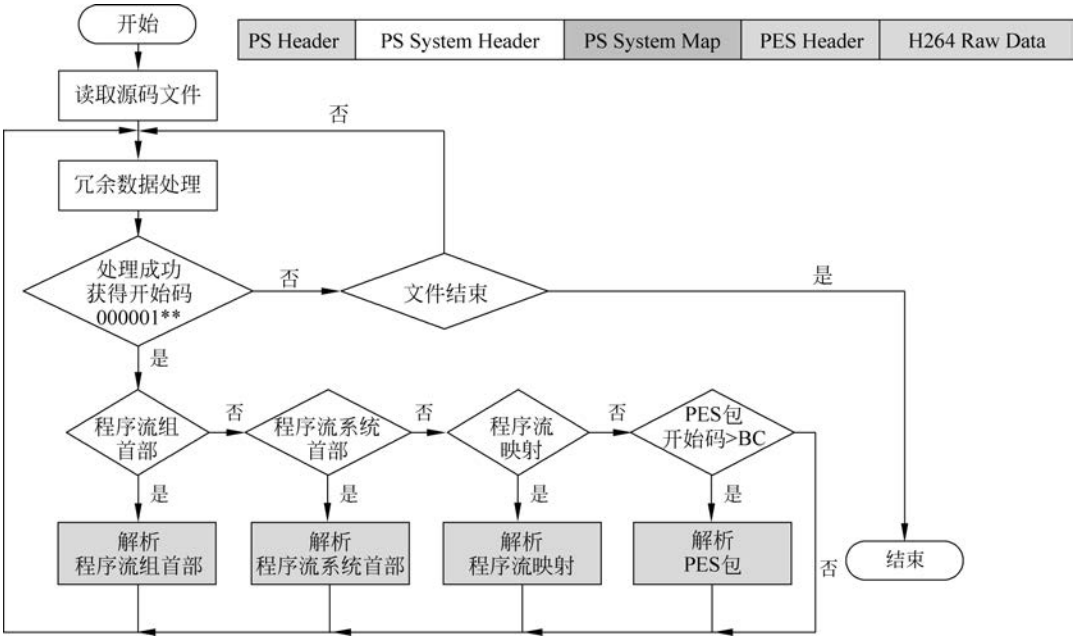


图 5-18　PS 码流解析流程

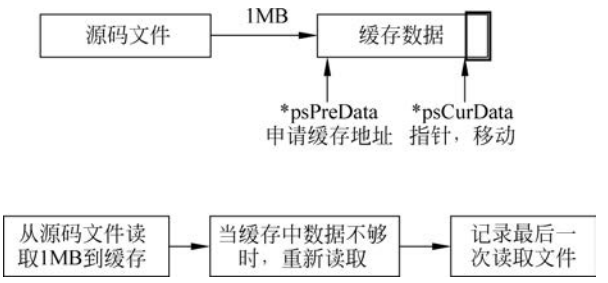


图 5-19　PS 码流解析之读取源文件

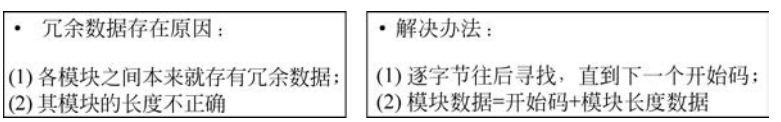


图 5-20　PS 码流解析之处理冗余数据

(3) 解析 PS Header，如图 5-21 所示。

(4) 解析 PS System Header(可有可无)，如图 5-22 所示。

(5) 解析 PS System Map(只有 I 帧才需要)，如图 5-23 所示。

(6) 解析 PES 包，如图 5-24 所示。

(7) 保存裸数据，如图 5-25 所示。

4字节	6字节	3字节	1字节	填充字节数目
开始码	base(33)+ext(9)	程序流速率	填充字节数目	填充字节
0x000001BA				

帧号

- 开始码：000001BA
- 长度最短：10字节
- 总长度为：10+填充字节数目
- *psCurData指针移动(10+填充字节数目)

图 5-21　PS 码流解析之 PS 头

4字节	2字节	3字节	1字节	1字节	1字节	1+2字节
开始码	系统首部长度	Max(程序流速率)	处理声频流最大数目(6)+比特率(1)+CSPS(1)	声频采样率(1)+视频图像速率(1)+处理视频流最大数目(5)	保留字节	Stream_id(1)+P-STD-buffer输入缓冲区(13)
0x000001BB						

- 开始码：000001BB
- 总长度为：2+系统首部长度
- *psCurData指针移动(2+系统首部长度)

图 5-22　PS 码流解析之 PS 系统头

4字节	2字节	1字节	1字节	2字节	描述子N	2字节	1+1+2+N2	4字节
开始码	PSM长度	PSM当前可用(1)+PSM版本号(5)	保留字节	描述子总长度N	描述子	PS所有原始流信息的总字节数	原始流类型+PES_id+描述子长度+描述子	CRC_32
0x000001BC								

- 开始码：000001BC
- 总长度为：2+PSM长度
- *psCurData指针移动(2+PSM长度)
- 注意：①PSM长度<1024；②描述子长度<PSM长度；③原始流信息总字节数<PSM长度

图 5-23　PS 码流解析之 PSM

PES包头　　H.264裸数据

4字节	2字节	2字节	1字节	PES包头长	5字节	……
开始码	PES包长	PES包头识别标志	PES包头长	可选信息	NAL_type	……
E0视频；C0声频					与0x1F按位“与”	……

- 开始码：000001** (**>BC)
- 总长度为：2+PES包长
- *psCurData指针移动(2+PES包长)
- 注意：PES包长不能为0

图 5-24　PS 码流解析之 PES

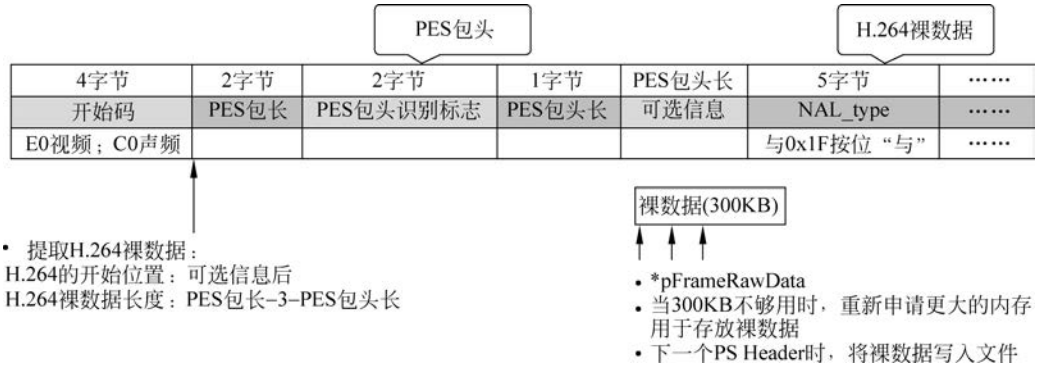


图 5-25　PS 码流解析之裸数据

（8）保存时间戳，如图 5-26 所示。

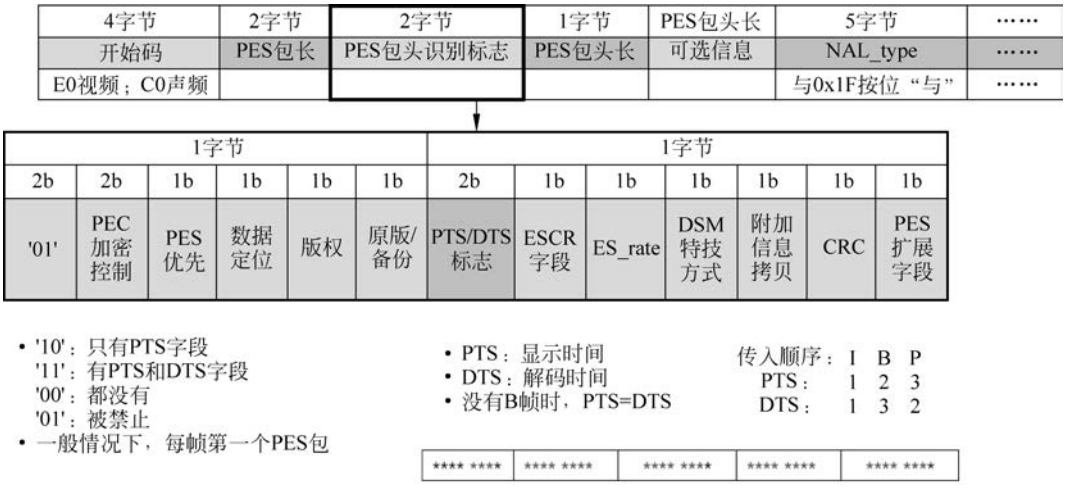


图 5-26　PS 码流解析之时间戳

（9）保存帧类型，如图 5-27 所示。

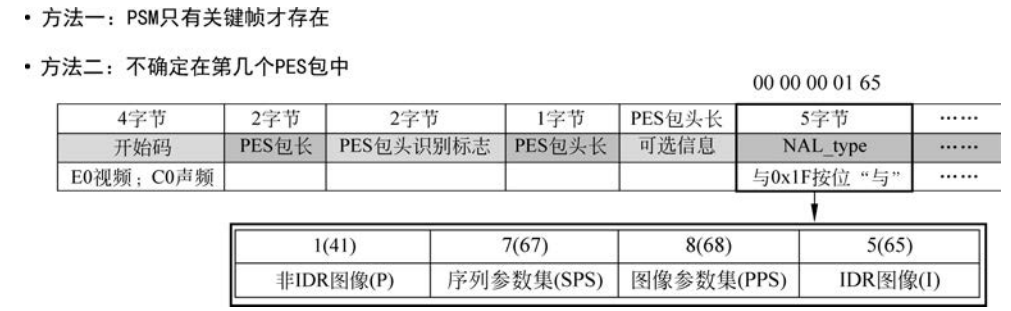


图 5-27　PS 码流解析之帧类型

PSM 提供了对 PS 流中的原始流和它们之间的相互关系的描述信息；PSM 作为一个 PES 分组出现，当 stream_id==0xBC 时，说明此 PES 包是一个 PSM；PSM 紧跟在系统头部后面；PSM 作为 PS 包的 Payload 存在。

解析 PS 包，要先找到 PS 包的起始码 0x000001BA 位串，然后解析出系统头部字段，之后进入 PS 包的负载，判断是否有 PSM，根据 PSM 确定 Payload 的 PES 包中所负载的 ES 流类型，然后根据 ES 流类型和 ID 从 PES 包中解析出具体的 ES 流；解包过程则相反，若要从 PS 流中找出帧类型，必须将 PS 包解析成 ES 并组成完整的帧，然后在帧数据的开始根据 NAL 头进行帧的类型判断。

PSM 只有在关键帧打包时，才会存在；IDR 包含了 SPS、PPS 和 I 帧；每个 IDR NALU 前一般会包含 SPS、PPS 等 NALU，因此将 SPS、PPS、IDR 的 NALU 封装为一个 PS 包，包括 PS 头、PS System Header、PSM、PES，所以一个 IDR NALU PS 包由外到内的顺序是：PS Header| PS System Header | PSM| PES。

对于其他非关键帧的 PS 包，直接加上 PS 头和 PES 头就可以了。顺序为 PS Header | PES Header | H.264 Raw Data。以上是对只有视频 Video 的情况，如果要把声频 Audio 也打包进 PS 封装，只需将数据加上 PES Header 放到视频 PES 后。

5.7 TS 格式与 m3u8 切片

HLS 是由苹果公司提出的基于 HTTP 的流媒体播出协议。由于它只使用 HTTP，因此具有开放、简洁、能穿越防火墙、与 CDN 系统对接方便等特点。在终端类型上，所有 iOS 终端(包括 Phone、iPod Touch、iPad、Mac)都支持 HLS 流媒体播放，最新发布的 Android 系统也开始加入对 HLS 的支持。

读者可能接触过 DVD，DVD 节目中的 MPEG2 格式，确切地说是 MPEG2-PS，MPEG2-PS 主要应用于存储具有固定时长的节目，如 DVD 电影，而 MPEG-TS 则主要应用于实时传送的节目，例如实时广播的电视节目。这两种格式有一些区别，将 DVD 上的 VoB 文件的前面截掉(或者干脆使数据损坏)，就会导致整个文件无法解码，而电视节目是可以在任何时候都能打开电视机进行解码(收看)的，所以 MPEG2-TS 格式的特点是要求从视频流的任一片段开始都是可以独立解码的。

大多数视频网站采用渐进式下载，将视频下载到播放设备上。视频一般采用流式传输，不只是下载 1 个文件，而是下载很多小包(本书指的是.ts 传输流切片文件)。服务器使用 HTTP 响应头 Accept-Range 标识自身支持范围请求(Partial Request)。字段的具体值用于定义范围请求的单位。当浏览器发现 Accept-Range 头时，可以尝试继续中断了的下载，而不是重新开始。Accept-Range 的值可以为 Bytes 或 None。Bytes 范围请求的单位是

Bytes(字节)。None 表示不支持任何范围请求单位,由于其等同于没有返回此头部,因此很少使用。不过一些浏览器,例如 IE 9,会依据该头部去禁用或者移除下载管理器的暂停按钮。

HLS 的工作原理是把整个流分成一个个小的基于 HTTP 的文件来下载,每次只下载一些。当媒体流正在播放时,客户端可以选择从许多不同的备用源中以不同的速率下载同样的资源,允许流媒体会话适应不同的数据速率。在开始一个流媒体会话时,客户端会下载一个包含元数据的 extended m3u(m3u8)列表文件,用于寻找可用的媒体流。HLS 只请求基本的 HTTP 报文,与实时传输协议(RTP)不同,HLS 可以穿过任何允许 HTTP 数据通过的防火墙或者代理服务器。它也很容易使用内容分发网络 CDN 来传输媒体流。HLS 流由众多 TS 小文件和 m3u8 索引文件组成,m3u8 切片工具实现 TS 文件的切片和索引文件生成。

m3u8 流切分工具需要支持的功能主要包括将声频或视频内容流化到 iPhone、iPod Touch、iPad 或者 Apple TV 上;不需要任何特殊的媒体服务器支持便可以将现场直播信号通过 HLS 输出到互联网上,实现具有加密和授权需求的 VoD 业务。

请求 m3u8 播放列表的方法,通过 m3u8 的 URI 进行请求,则该文件必须以. m3u8 或. m3u 结尾;通过 HTTP 进行请求,则请求头 Content-Type 必须设置为 application/vnd. apple. mpegurl 或者 audio/mpegurl。

可以使用 ffmpeg 命令行实现视频文件的切片,命令如下:

```
ffmpeg -i XXX.mp4 -c:v libx264 -c:a copy -f hls XXX.m3u8
```

其中,XXX. mp4 为本地视频文件,XXX. m3u8 为最终生成的播放索引列表,与此同时还有多个 TS 文件。

如果想开发直播流切片工具和文件切片工具,则应分别满足 HLS 直播流和点播流的切片需求,具体描述如下:

(1) 直播流切片工具(Stream Segmenter),从网络上读取直播数据,通过在线实时切分,将符合 HLS 规格的直播流输出到互联网上。它一般通过 UDP 接收由编码器或其他系统输出的 TS 流,将 TS 流实时地切分成具有固定播出长度的小文件。这些从连续直播流中分离出来的小文件在播出结构上具有严密的连续性,可以被无缝地重新封装以满足 HLS 播出要求。该工具必须同时生成 m3u8 索引文件,直播流 m3u8 索引文件随着新片段文件的不断生成进行不断更新,以符合 HLS 直播规范的要求。切分出的小文件以 TS 文件格式存放,索引文件以具有. m3u8 后缀的 m3u8 文件格式存放。

(2) 文件切片工具,实现将视频或声频文件切分成符合 HLS 规范要求的片段文件,这些文件能够通过 HLS 协议对外提供点播服务。文件切片工具与流切片工具的工作内容相似,区别是一个用于切分直播流,另一个用于切分多媒体文件。文件切分工具需要支持

MP4、TS、MOV、FLV 等多种文件格式。如果要切分的文件满足 HLS 对文件格式的要求(H.264 + AAC 或者 H.264+MP3),则不需要进行重新编码,可直接进行文件切片。否则需要对声频或视频内容进行重新编码,以满足 HLS 播出要求。文件切分工具具有“重新编码”和“不重新编码”的工作模式,使用时可以根据需要进行选择。

注意:有兴趣的读者可以编写代码实现以上功能,或者从网络上搜索一些开源软件。