# 音视频封装格式之TS(一)

## **前言：**

大家晚上，今天开始给大家分享音视频里面的各种封装格式解析，先给大家分享封装格式基本概念，后期再分析代码实现封装格式解析。

后期的ffmpeg源码解析，我也会快速把自己掌握的一部分，到时候分享给大家，也欢迎大家多多交流，一起进步，一起飞！

## **一、TS格式解析：**

### **1、TS流、PS流、PES流和ES流是啥?**

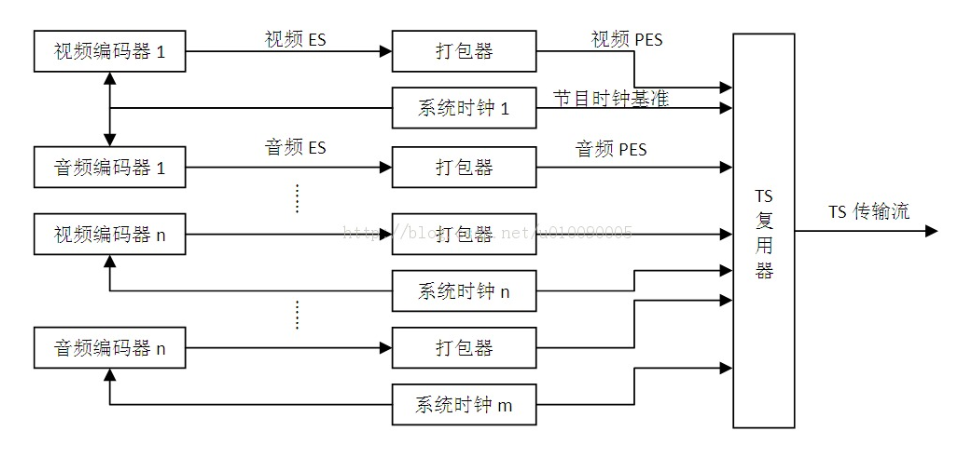
ES流:（Elementary Stream）：基本码流，不分段的音频、视频或其他信息的连续码流。

PES流：( Packet Elemental Stream)把基本流ES分割成段，并加上相应头文件打包成形的打包基本码流。

PS流（Program Stream）：节目流，将具有共同时间基准的一个或多个PES组合（复合）而成的单一数据流（用于播放或编辑系统，如m2p）。

TS流（Transport Stream）：传输流，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）而成的单一数据流（用于数据传输）。视频编码主要格式为 H264/MPEG4，⾳频为 AAC/MP3；

### **2、TS流产生过程：**

TS流产生过程

从上图可以看出，视频ES和音频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复用器复用形成的传输流。

注意这里的TS流是位流格式（分析Packet的时候会解释），也即是说TS流是可以按位读取的。

### **3、TS流格式：**

TS流是基于Packet的位流格式，每个包是188个字节（或204个字节，在188个字节后加上了16字节的CRC校验数据，其他格式一样）。ts流分为三个部分：ts header、adaptation field、payload。

ts header 固定 4 个字节；

adaptation field 可能存在也可能不存在，主要作⽤是给不⾜ 188 字节的数据做填充；payload 是 pes数据Packet Header（包头）信息说明：

Packet包头说明

PID是TS流中唯一识别标志，Packet Data是什么内容就是由PID决定的。如果一个TS流中的一个Packet的Packet Header中的PID是0x0000，那么这个Packet的Packet Data就是DVB的PAT表而非其他类型数据（如Video、Audio或其他业务信息）。下表给出了一些表的PID值，这些值是固定的，不允许用于更改。

PID值表

同时解析 ts 流要 先找到 PAT 表，只要找到 PAT 就可以找到 PMT，然后就可以找到⾳视频流了。PAT 表的和 PMT 表需 要定期插⼊ ts 流，因为⽤户随时可能加⼊ ts 流，这个间隔⽐较⼩，通常每隔⼏个视频帧就要加⼊ PAT 和 PMT。PAT 和 PMT 表是必须的，还可以加⼊其它表如 SDT（业务描述表）等，不过 hls 流只要有 PAT 和 PMT 就可以播放了。

PAT 表：主要的作⽤就是指明了

PMT 表的 PID 值。PMT 表：主要的作⽤就是指明了⾳视频流的 PID 值。

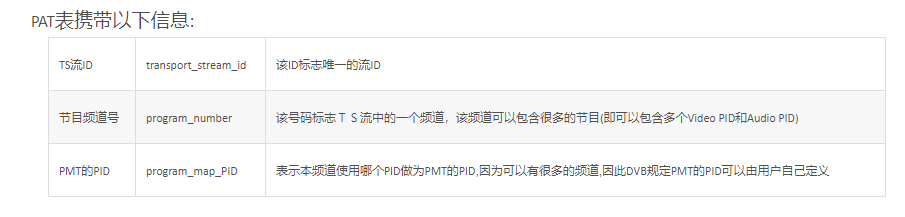
⾳频流/视频流：承载⾳视频内容。

### **4、什么是PAT、PAM表：**

PAT表（Program Association Table，节目关联表）

PMT表 （Program Map Table，节目映射表）

这里我们先来看PAT表，首先PAT表定义了当前TS流中所有的节目，其PID为0x0000，它是PSI的根节点，要查寻找节目必须从PAT表开始查找。



PAT表主要包含频道号码和每一个频道对应的PMT的PID号码,这些信息我们在处理PAT表格的时候会保存起来，以后会使用到这些数据。下面将PAT表的定义给出：

typedef struct TS\_PAT\_Program  
{  
    unsigned program\_number   :  16;  //节目号  
    unsigned program\_map\_PID :  13; // 节目映射表的PID，节目号大于0时对应的PID，每个节目对应一个  
}TS\_PAT\_Program

下面是PAT结构体定义：

typedef struct TS\_PAT  
{  
    unsigned table\_id                     : 8; //固定为0x00 ，标志是该表是PAT表  
    unsigned section\_syntax\_indicator     : 1; //段语法标志位，固定为1  
    unsigned zero                         : 1; //0  
    unsigned reserved\_1                   : 2; // 保留位  
     unsigned section\_length               : 12; //表示从下一个字段开始到CRC32(含)之间有用的字节数  
    unsigned transport\_stream\_id          : 16; //该传输流的ID，区别于一个网络中其它多路复用的流  
    unsigned reserved\_2                   : 2;// 保留位  
    unsigned version\_number               : 5; //范围0-31，表示PAT的版本号  
    unsigned current\_next\_indicator       : 1; //发送的PAT是当前有效还是下一个PAT有效  
    unsigned section\_number               : 8; //分段的号码。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  
    unsigned last\_section\_number          : 8;  //最后一个分段的号码  
   
    std::vector<TS\_PAT\_Program> program;  
    unsigned reserved\_3                    : 3; // 保留位  
    unsigned network\_PID                    : 13; //网络信息表（NIT）的PID,节目号为0时对应的PID为network\_PID  
    unsigned CRC\_32                        : 32;  //CRC32校验码  
} TS\_PAT;

PAI解析：

HRESULT CTS\_Stream\_Parse::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;  
    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;  
           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 );  
           TS\_program.push\_back( PAT\_program );//向全局PAT节目数组中添加PAT节目信息       
        }           
    }  
    return 0;  
}

具体工程代码后面再分享，今天暂时不分享，主要还是以概念为主。

接下来我们再来看PAM表,如果一个TS流中含有多个频道，那么就会包含多个PID不同的PMT表。

PMT表中包含的数据如下：

当前频道中包含的所有Video数据的PID

当前频道中包含的所有Audio数据的PID

和当前频道关联在一起的其他数据的PID(如数字广播,数据通讯等使用的PID)

只要我们处理了PMT，那么我们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分别是什么。这样如果我们要选择其中一个Video和Audio收看，那么只需要把要收看的节目的Video PID和Audio PID保存起来，在处理Packet的时候进行过滤即可实现。

typedef struct TS\_PMT\_Stream    
{    
 unsigned stream\_type                       : 8; //指示特定PID的节目元素包的类型。该处PID由elementary PID指定    
 unsigned elementary\_PID                    : 13; //该域指示TS包的PID值。这些TS包含有相关的节目元素    
 unsigned ES\_info\_length                    : 12; //前两位bit为00。该域指示跟随其后的描述相关节目元素的byte数    
 unsigned descriptor;    
}TS\_PMT\_Stream;

PMT表的结构体定义:

//PMT 表结构体  
typedef struct TS\_PMT  
{  
    unsigned table\_id                        : 8; //固定为0x02, 表示PMT表  
     unsigned section\_syntax\_indicator        : 1; //固定为0x01  
    unsigned zero                            : 1; //0x01  
    unsigned reserved\_1                      : 2; //0x03  
    unsigned section\_length                  : 12;//首先两位bit置为00，它指示段的byte数，由段长度域开始，包含CRC。  
     unsigned program\_number                    : 16;// 指出该节目对应于可应用的Program map PID  
    unsigned reserved\_2                        : 2; //0x03  
    unsigned version\_number                    : 5; //指出TS流中Program map section的版本号  
     unsigned current\_next\_indicator            : 1; //当该位置1时，当前传送的Program map section可用；  
                                                     //当该位置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。  
     unsigned section\_number                    : 8; //固定为0x00  
    unsigned last\_section\_number            : 8; //固定为0x00  
    unsigned reserved\_3                        : 3; //0x07  
    unsigned PCR\_PID                        : 13; //指明TS包的PID值，该TS包含有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。  
     unsigned reserved\_4                        : 4; //预留为0x0F  
    unsigned program\_info\_length            : 12; //前两位bit为00。该域指出跟随其后对节目信息的描述的byte数。  
      
     std::vector<TS\_PMT\_Stream> PMT\_Stream;  //每个元素包含8位, 指示特定PID的节目元素包的类型。该处PID由elementary PID指定  
     unsigned reserved\_5                        : 3; //0x07  
    unsigned reserved\_6                        : 4; //0x0F  
    unsigned CRC\_32                            : 32;   
} TS\_PMT;

PMT表的解析:

HRESULT CTS\_Stream\_Parse::adjust\_PMT\_table ( TS\_PMT \* packet, unsigned char \* buffer )  
{   
    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  
    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;  
  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;  
  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 );  
  TS\_Stream\_type.push\_back( pmt\_stream );  
    }  
 return 0;  
}

## **二、总结：**

今天的总结就分享到这里，我们下期见，下期分享flv封装格式的基本概念；