【PSI/SI学习系列】1.从TS流到PAT和PMT

前言

欢迎到我的网站阅读：http://www.onelib.biz/blog/stb

一 从TS流开始

最近开始学习数字电视机顶盒的开发，从MPEG-2到DVB，看着看着突然就出现了一大堆表格，什么PAT、PMT、CAT.....如此多的表该怎样深入了解呢?

我们知道，数字电视机顶盒接收到的是一段段的码流，我们称之为TS（Transport Stream，传输流），每个TS流都携带一些信息，如Video、Audio以及我们需要学习的PAT、PMT等信息。因此，我们首先需要了解TS流是什么，以及TS流是怎样形成、有着怎样的结构。

（一） TS流、PS流、PES流和ES流都是什么?

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：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。

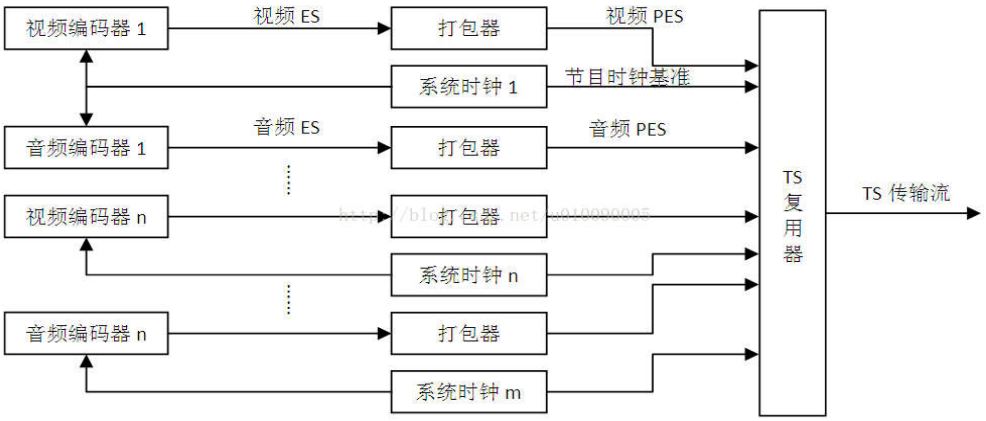
PS流（Program Stream）：节目流，将具有共同时间基准的一个或多个PES组合（复合）而成的单一数据流（用于播放或编辑系统，如m2p）。

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）。

*NOTE:TS流和PS流的区别：TS流的包结构是长度是固定的；PS流的包结构是可变长度的。这导致了TS流的**抵抗传输误码**的能力强于PS流（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，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，传输误码较高时，一般采用TS码流；而在信道环境较好，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码流基本上都采用了TS码流的包格。

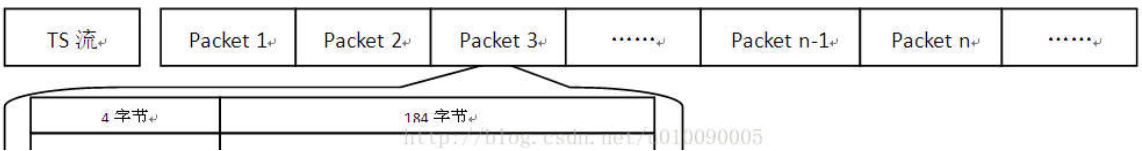
（二） TS流是如何产生的?

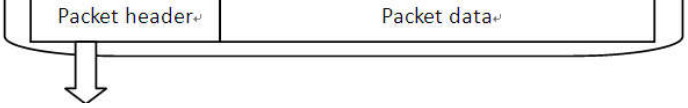


从上图可以看出，视频ES和音频ES通过打包器和共同或独立的系统时间基准形成一个个PES，通过TS复用器复用形成的传输流。注意这里的TS流是位流格式（分析Packet的时候会解释），也即是说TS流是可以按位读取的。

（三） TS流的格式是怎样的?

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）。整个TS流组成形式如下：





Packet Header（包头）信息说明			
1	sync_byte	8bits	同步字节
2	transport_error_indicator	1bit	错误指示信息（1：该包至少有1bits传输错误）
3	payload_unit_start_indicator	1bit	负载单元开始标志（packet不满188字节时需填充）
4	transport_priority	1bit	传输优先级标志（1：优先级高）
5	**PID**	**13bits**	**Packet ID号码，唯一的号码对应不同的包**
6	transport_scrambling_control	2bits	加密标志（00：未加密；其他表示已加密）
7	adaptation_field_control	2bits	附加区域控制
8	continuity_counter	4bits	包递增计数器

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。

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

下面以一个TS流的其中一个Packet中的Packet Header为例进行说明：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	...
Packet（十六进制）	4				7				0				7				e				5				1				2				...
Packet（二进制）	0	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	0	0	1	0	...
Packet Header信息	1 sync_byte=0x47								2	3	4	5 PID=0x07e5													6		7		8				...

sync_byte=01000111, 就是0x47,这是DVB TS规定的同步字节,固定是0x47.

transport_error_indicator=0, 表示当前包没有发生传输错误.

payload_unit_start_indicator=0, 含义参考ISO13818-1标准文档

transport_priority=0, 表示当前包是低优先级.

PID=00111 11100101即0x07e5, Video PID

transport_scrambling_control=00, 表示节目没有加密

adaptation_field_control=01 即0x01,具体含义请参考ISO13818-1

continuity_counte=0010　　即0x02,表示当前传送的相同类型的包是第3个

TS流的基本内容就是这些了。

回顾一下，TS流是一种位流（当然就是数字的），它是由ES流分割成PES后复用而成的；它经过网络传输被机顶盒接收到；数字电视机顶盒接收到TS流后将解析TS流。

TS流是由一个个Packet（包）构成的，每个包都是由Packet Header（包头）和Packet Data（包数据）组成的。其中Packet Header指示了该Packet是什么属性的，并给出了该Packet Data的数据的唯一网络标识符PID。

到这里，我们对TS流已经有了一定的了解，下面将从TS流转向PAT表和PMT表的学习。

二 从TS流到PAT、PMT

说完了TS流的基本概念，就该开始对TS流进行更深入的研究了。首先需要想一想：TS流的本质是什么？它的确是一段码流，并且是一段由数据包（Packet）组成的码流。那么这些数据包究竟是怎样的呢？它和我们收看的电视节目之间又有什么区别？这些都是这部分需要了解的内容。

在上一节中，我们可以看到**PID**这个被标红的字段频繁地出现。**PID**是当前TS流的Packet区别于其他Packet类型的唯一识别符，通过读取每个包的Packet Header，我们可以知道这个Packet的数据属于何种类型。上一节列出了几项固定的**PID**值，它们用于识别存储了特殊信息的Packet。下面要谈的PAT表的**PID**值就是固定的0x0000。

（一） PAT表（Program Association Table，节目关联表）

由于下面的内容比较繁杂，这里先给出大纲，方便查阅：

1. PAT表的描述（表格+分析）
2. PAT表的定义（代码+分析）
3. PAT表的结构（代码+分析）
4. PAT表的解析（代码+分析）
5. 通过TS流获取PAT表（代码+分析）

下面，开始正式的分析！

1. PAT表的描述（表格+分析）

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表携带以下信息:

TS流ID	transport_stream_id	该ID标志唯一的流ID
节目频道号	program_number	该号码标志TS流中的一个频道，该频道可以包含很多的节目(即可以包含多个Video PID和Audio PID)
PMT的PID	program_map_PID	表示本频道使用哪个PID做为PMT的PID,因为可以有很多的频道,因此DVB规定PMT的PID可以由用户自己定义

2. PAT表的定义（代码+分析）

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：

```cpp
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
```

3. PAT表的结构（代码+分析）

再将PAT表的结构体给出：

```cpp
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，最大值1021
    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,节目映射表的可选内容
    unsigned CRC_32                     : 32;  //CRC32校验码
} TS_PAT;
```

4. PAT表的解析（代码+分析）

接下来给出的是PAT表的解析代码：

```cpp
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; //记录该节目的网络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 );//向全局节目数组中插入节目
        }
    }
    return 0;
}
```

从for()开始，就是描述了当前流中的频道数目(N)，每一个频道对应的PMT PID是什么。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到PMT的 PID。

5. 通过一段TS流中一个Packet分析PAT表（表格+分析）

这里我们分析一段TS流其中一个Packet的Packet Data部分：
首先给出一个数据包，其数据如下：

Packet Header	Packet Data
0x47 0x40 0x00 0x10	0000 b0 11 00 01 c1 00 00 00 00 e0 1f 00 01 e1 00 24 ac48 84 ff ff..... ff ff

分析Packet Header如下表所示：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	...
Packet（十六进制）	4				7				4				0				0				0				1				0				...
Packet（二进制）	0	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	...
Packet Header Bits	1 sync_byte=0x47								2	3	4	5 PID=0x0000													6		7		8				...

根据包头数据格式，我们可以知晓整个数据包的属性，列表如下：

sync_byte	0x47	固定同步字节
transport_error_indicator	“0”	没有传输错误

payload_unit_start_indicator	“1”	在前4个字节后会有一个调整字节。所以实际数据应该为去除第一个字节后的数据。即上面数据中红色部分不属于有效数据包。
transport_priority	“0”	传输优先级低
PID	0x0000	PID=0x0000说明数据包是PAT表信息
transport_scrambling_control	“00”	未加密
adaptation_field_control	“01”	附加区域控制
continuity_counte	“0000”	包递增计数器

如上表所示，我们可以知道，首先Packet的Packet Data是PAT信息表，因为其PID为0x0000，并且在包头后需要除去一个字节才是有效数据（payload_unit_start_indicator="1"）。这样，Packet Data就应该是“00 b0 11 00 01 c1 00 00 00 00 e0 1f 00 01 e1 00 24 ac48 84 ff ff ..... ff ff”。

Packet Data分析																				
第n个字节	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Packet Data(除去开头的0x00)	00	b0	11	00	01	c1	00	00	00	00	e0	1f	00	01	e1	00	24	ac	48	84	...

字段名	位	具体值	次序	说明
table_id	8	0000	第1个字节 0000 0000B(0x00)	PAT的table_id只能是0x00
section_syntax_indicator	1	1	第2、3个字节 1011 0000 0001 0001B(0xb0 11)	段语法标志位，固定为1
zero	1	0		
reserved	2	11		
section_length	12	0000 0001 0001B=0x011=17		段长度为17字节
transport_stream_id	16	0x0001	第4、5个字节 0x00 0x01	
reserved	2	11	第6个字节 1100 0001B(0xc1)	
version_number	5	00000		一旦PAT有变化，版本号加1
current_next_indicator	1	1		当前传送的PAT表可以使用，若为0则要等待下一个表
section_number	8	0x00	第7个字节0x00	
last_section_number	8	0x00	第8个字节 0x00	
开始循环				
program_number	16	0x0000-第一次	2个字节(0x00 00)	**节目号**
reserved	3	111	2个字节 1110 0000 0001 1111B(0xe0 1f)	
network_id(节目号为0时) program_map_PID(节目号为其他时)	13	0 0000 0001 1111B=31 -第一次		**节目号**为0x0000时指定为network_PID，即PID=0x001f，即31 **节目号**为0x0000时指定为network_PID，即PID=0x100，即256
结束循环				
CRC_32	32	--	4个字节	

由以上几个表可以分析出PAT表和PMT表有着内在的联系。也就是之前提到的。PAT表描述了当前流的NIT（Network Information Table，网络信息表）中的PID、当前流中有多少不同类型的PMT表及每个PMT表对应的频道号。而PAT表和PMT表到底有什么深层次的联系呢？在讨论完了PMT表和SDT表后再做讨论吧。

6. 过滤PAT表信息的伪代码（代码）

[cpp]

```cpp
int Video_PID=0x07e5,Audio_PID=0x07e6;
void Process_Packet(unsigned char*buff)
{ int I; int PID=GETPID(buff);
  if(PID==0x0000) { Process_PAT(buff+4); }  // 如果[illegible]的函数
  else{                                      // 这里[illegible]节)
    ……
  }
}
```

（二） PMT表（Program Map Table，节目映射表）（Service Descriptor Table）

1. PMT表的描述

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：

(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。

2. PMT表的定义（代码）

[cpp]

```cpp
<span style="font-size:14px;">//PMT [illegible]</span>
```

[cpp]

```cpp
<span style="font-size:14px;">typedef struct TS_PMT_Stream
{
 unsigned stream_type                       : 8; //指[illegible]定
 unsigned elementary_PID                    : 13; //该域[illegible]
 unsigned ES_info_length                    : 12; //前两位[illegible]
 unsigned descriptor;
}TS_PMT_Stream;
```

```
</span>
```

3. PMT表的结构体定义（代码）

```cpp
//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;//首先两位置0，[illegible]
CRC
     unsigned program_number                    : 16;// 指出该节目对应于可应用的Pro[illegible]
    unsigned reserved_2                        : 2; //0x03
    unsigned version_number                    : 5; //指出TS流中Program map section的版本号[illegible]
     unsigned current_next_indicator            : 1; //当该位置1时，当前传送的Program map section可用；[illegible]
                                                     //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。[illegible]
     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，[illegible]
            //该节目[illegible]节目。
            //如果对于私有数据流的节目定义与PCR无关，[illegible]
     unsigned reserved_4                        : 4; //预留为0x0F
    unsigned program_info_length             : 12; //前两位bit为00。[illegible]

     std::vector<TS_PMT_Stream> PMT_Stream;  //每个元素包含8位, [illegible]
elementary PID指定
     unsigned reserved_5                        : 3; //0x07
    unsigned reserved_6                        : 4; //0x0F
    unsigned CRC_32                            : 32;
} TS_PMT;
```

4. PMT表的解析（代码）

```cpp
//PMT 表的解析
```

```cpp
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;
 }
```

5. 通过一段TS流中一个Packet分析PMT表（表格+分析）

老样子，还是通过分析一段TS流的数据包Packet来学习PMT表。

下面给出了一段TS流数据中的一个Packet（十六进制数）

Packet Header	Packet Data
0x47 0x43 0xe8 0x12	00 02 b0 12 00 01 c1 00 00 e3 e9 f0 00 1b e3 e9 f0 00 f0 af b4 4f ff ff..... ff ff

首先解析Packet Header，分析如下：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	...
Packet（十六进制）	4				7				4				3				e				8				1				2				...
Packet（二进制）	0	1	0	0	0	1	1	1	0	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	...
Packet Header Bits	1 sync_byte=0x47								2	3	4	5 PID=0x03e8													6		7		8				...

PID=0x03e8为其PID

下面是详细的解析表

Packet Header分析			
	Packet Header：0x47 0x40 0x00 0x10		
1	sync_byte	0x47	固定同步字节
2	transport_error_indicator	“0”	没有传输错误
3	payload_unit_start_indicator	“1”	在前4个字节后会有一个调整字节。所以实际数据应该为去除第一个字节后的数据。
4	transport_priority	“0”	传输优先级低
5	PID	0x03e8	PID=0x03e8说明数据包是PMT表信息
6	transport_scrambling_control	“00”	未加密
7	adaptation_field_control	“01”	附加区域控制
8	continuity_counte	“0010”	包递增计数器

因为payload_unit_start_indicator=‘1’，在解析数据包的时候需要去除Packet Data的第一个字节。下面是对Packet Data的详细解析：

PMT表的表的表的a分析Data																					
第n个字节	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Packet Data	02	b0	12	00	01	c1	00	00	e3	e9	f0	00	1b	e3	e9	f0	00	f0	1b	e3	...

字段名	位数	具体值	次序	说明
table_id	8	0x02	第1个字节	
section_syntax_indicator	1	1B	第2、3个字节 1011 0000 0001 0010B=0xb012	段语法标志
zero	1	0B		
reserved	2	11B=0x03		
section_length	12	0000 0001 0010B=0x12		段长度,从program_number开始,到CRC_32(含)的字节总数
program_number	16	0x0001	第4、5个字节 0x00 01	频道号码,表示当前的PMT关联到的频道

reserved	2	11B=0x03	第6个字节 1100 0001B=0xc1	
version_number	5	00000B=0x00		版本号码,如果PMT内容有更新,则它会递增1通知解复用程序需要重新接收节目信息
current_next_indicator	1	1B=0x01		当前未来标志符
section_number	8	0x00	第7个字节0x00	当前段号码
last_section_number	8	0x00	第8个字节 0x00	最后段号码,含义和PAT中的对应字段相同
reserved	3	111B=0x07	第9、10个字节 1110 0011 1110 1001B=0xe3e9	
PCR_PID	13	000111110B=0x3e9		PCR(节目参考时钟)所在TS分组的PID
reserved	4	1111B=0x0f	第11、12个字节 1111 0000 0000 0000=0xf000	
program_info_length	12	000000000000B=0x000		节目信息长度(之后的是N个描述符结构,一般可以忽略掉,这个字段就代表描述符总的长度,单位是Bytes)紧接着就是频道内部包含的节目类型和对应的PID号码了
stream_type	8	0x1b	第13个字节 0x1b	流类型,标志是Video还是Audio还是其他数据
reserved	3	111B=0x07	第14、15个字节 1110 0011 1110 1001B=0xe3e9	
elementary_PID	13	000111110 1001=0x3e9		该节目中包括的视频流,音频流等对应的TS分组的PID
reserved	4	1111B=0x0f	第16、17个字节 1111 0000 0000 0000B=0xf000	
ES_info_length	12	0000 0000 0000=0x000		
CRC	32	——	——	

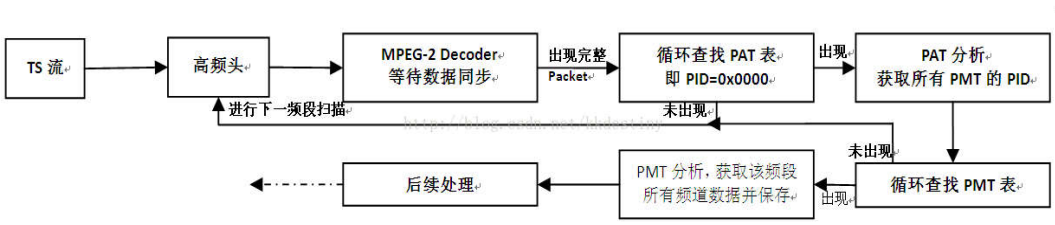
（三） 解复用模型（代码）

```cpp
int Video_PID=0x07e5,Audio_PID=0x07e6;
void Process_Packet(unsigned char*buff)
{
  int i; int PID=GETPID(buff);
  if(PID==0x0000) { Process_PAT(buff+4); }
  else if(PID==Video_PID) { SaveToVideoBuffer(buff+4); }
  else if(PID==Audio_PID) { SaveToAudioBuffer(buff+4); }
  else{
  for( i=0;i<64;i++)
  { if(PID==pmt[i].pmt_pid) { Process_PMT(buff+4); Break; }
} } }
```

解复用的意义在于，由于TS流是一种复用的码流，里面混杂了多种类型的包；解复用TS流可以将类型相同的Packet存入相同缓存，分别处理。这样就可以将Video、Audio或者其他业务信息的数据区分开来。

（四） DVB搜台原理以及SDT表（Service Descriptor Table，业务描述表）

机顶盒先调整高频头到一个固定的频率(如498MHZ)，如果此频率有数字信号，则COFDM芯片(如MT352)会自动把TS流数据传送给MPEG-2 decoder 。 MPEG-2 decoder先进行数据的同步，也就是等待完整的Packet的到来.然后循环查找是否出现PID == 0x0000 的Packet，如果出现了，则马上进入分析PAT的处理，获取了所有的[illegible]明该频段没有信号，进入下一个频率扫描。



在解析TS流的时候，首先寻找PAT表，根据PAT获取所有PMT表的PID；再寻找PMT表，获取该频段所有节目数据并保存。这样，只需要知道节目的PID就可以根据[illegible]且容易输错，所以需要有一张表将节目名称和该节目的PID对应起来，DVB设计了SDT表来解决这个问题。 该表格标志一个节目的名称，并且能和PMT中的PID联系起来，这样用户就可以通过直接选择节目名称来选择节目了。

SDT可以提供的信息包括:

(1) 该节目是否在播放中

(2) 该节目是否被加密

(3) 该节目的名称

三、 从PAT开始，走向更远

在本章的学习中，我们发现了一个特点：所有的TS流的解析都是从寻找PAT表开始的，只有找到了PAT表，我们才能继续下一步的解析。因此，在进行了TS流、PAT表和PMT表的初步知识储备后，在接下来的学习中将从PAT表开始，学习更多的PSI/SI相关的表，将走得更远。

-本章结束-

声明: 本篇文章的部分代码来自：
1. http://blog.csdn.net/beyondzd2000/article/details/8007325
2. http://blog.csdn.net/a31898534/article/details/4399374
同时还参考了一些前辈的资料，如：
1. http://blog.csdn.net/a31898534/article/details/4399374
2. http://blog.sina.com.cn/s/blog_4ae178ba01018o7g.html
感谢各位前辈的努力，才有了我们这些后来人的轻松！

经验水平所限，若有错漏之处，期待大家的批评指正！

==========

博主信息

==========

Destiny
QQ: 1139904786
邮箱/Email:1139904786@qq.com

==========