关于 TS 流的解析

TS 即是"Transport Stream"的缩写。他是分包发送的，每一个包长为 188 字节。在 TS 流里可以填入很多类型的数据，如视频、音频、自定义信息等。他的包的结构为，包头为 4 个字节，负载为 184 个字节（这 184 个字节不一定都是有效数据，有一些可能为填充数据）。

工作形式：

因为在 TS 流里可以填入很多种东西，所以有必要有一种机制来确定怎么来标识这些数据。制定 TS 流标准的机构就规定了一些数据结构来定义。比如: PSI（Program Specific Information）表，所以解析起来就像这样：先接收一个负载里为 PAT 的数据包，在整个数据包里找到一个 PMT 包的 ID。然后再接收一个含有 PMT 的数据包，在这个数据包里找到有关填入数据类型的 ID。之后就在接收到的 TS 包里找含有这个 ID 的负载内容，这个内容就是填入的信息。根据填入的数据类型的 ID 的不同，在 TS 流复合多种信息是可行的。关键就是找到标识的 ID 号。

现在以一个例子来说明具体的操作：

在开始之前先给出一片实际 TS 流例子：

```
0000f32ch: 47 40 00 17 00 00 B0 0D 00 01 C1 00 00 00 01 E0 ; G@....?..?...?
0000f33ch: 20 A2 C3 29 41 FF FF FF FF FF FF FF FF FF FF FF ;  ?9.)A
0000f34ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f35ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f36ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f37ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f38ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f39ch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f3ach: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f3bch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f3cch: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ;
0000f3dch: FF FF FF FF FF FF FF FF FF FF FF FF 47 40 20 17 ;                    G@ .
0000f3ech: 00 02 B0 1B 00 01 C1 00 00 E0 21 F0 00 1B E0 21 ; ..?..?.??.?
0000f3fch: F0 04 2A 02 7E 1F 03 E0 22 F0 00 5D 16 BD 48    ; ?*.~..??].紿
```

具体的分析就以这个例子来分析。

```
// Adjust TS packet header
void adjust_TS_packet_header(TS_packet_header* pheader)
{
    unsigned char buf[4];
    memcpy(buf, pheader, 4);
    pheader->transport_error_indicator      = buf[1] >> 7;
    pheader->payload_unit_start_indicator   = buf[1] >> 6 & 0x01;
    pheader->transport_priority             = buf[1] >> 5 & 0x01;
    pheader->PID                            = (buf[1] & 0x1F) << 8 | buf[2];
    pheader->transport_scrambling_control   = buf[3] >> 6;
    pheader->adaption_field_control         = buf[3] >> 4 & 0x03;
    pheader->continuity_counter             = buf[3] & 0x03;
}
```

这是一个调整 TS 流数据包头的函数，这里牵扯到位段调整的问题。现在看一下 TS 流数据包头的结构的定义：

```
// Transport packet header
typedef struct TS_packet_header
{
    unsigned sync_byte                  : 8;
    unsigned transport_error_indicator      : 1;
    unsigned payload_unit_start_indicator   : 1;
    unsigned transport_priority             : 1;
    unsigned PID                        : 13;
    unsigned transport_scrambling_control   : 2;
    unsigned adaption_field_control         : 2;
    unsigned continuity_counter             : 4;
} TS_packet_header;
```

下面我们来分析，在 ISO/IEC 13818-1 里有说明，PAT(Program Association Table)的 PID 值为 0x00，TS 包的标识(即 sync_byte)为 0x47，并且为了确保这个 TS 包里的数据有效，所以我们一开始查找 47 40 00 这三组 16 进制数，为什么这样？具体的奥秘在 TS 包的结构上，前面已经说了 sync_byte 固定为 0x47。现在往下看 transport_error_indicator、payload_unit_start_indicator、transport_priority 和 PID 这四个元素，PID 为 0x00，这是 PAT 的标识。transport_error_indicator 为 0，transport_priority 为 0。把他们看成是两组 8 位 16 进制数就是：40 00。现在看看我们的 TS 流片断例子，看来正好是 47 40 00 开头的，一个 TS 流的头部占据了 4 个字节。剩下的负载部分的内容由 PID 来决定，例子看来就是一个 PAT 表。在这里有个地方需要注意一下，payload_unit_start_indicator 为 1 时，在前 4 个字节之后会有一个调整字节，它的数值决定了负载内容的具体开始位置。现在看例子中的数据 47 40 00 17 00 第五个字节是 00，说明紧跟着 00 之后就是具体的负载内容。

下面给出 PAT 表的结构体：

```
// PAT table
// Programm Association Table
typedef struct TS_PAT
{
    unsigned table_id                   : 8;
    unsigned section_syntax_indicator       : 1;
    unsigned zero                       : 1;
    unsigned reserved_1                     : 2;
    unsigned section_length                 : 12;
    unsigned transport_stream_id            : 16;
    unsigned reserved_2                     : 2;
    unsigned version_number                 : 5;
    unsigned current_next_indicator         : 1;
    unsigned section_number                 : 8;
    unsigned last_section_number            : 8;
    unsigned program_number                 : 16;
    unsigned reserved_3                     : 3;
    unsigned network_PID                    : 13;
    unsigned program_map_PID                : 13;
    unsigned CRC_32                         : 32;
} TS_PAT;
```

再给出 PAT 表字段调整函数：

```
// Adjust PAT table
void adjust_PAT_table ( TS_PAT * packet, char * buffer )
{
    int n = 0, i = 0;
    int len = 0;
    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];
    // Get CRC_32
    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);
    // Parse network_PID or program_map_PID
    for ( n = 0; n < packet->section_length - 4; n ++ )
    {
        packet->program_number          = buffer[8] << 8 | buffer[9];
        packet->reserved_3              = buffer[10] >> 5;
        if ( packet->program_number == 0x0 )
            packet->network_PID = (buffer[10] << 3) << 5 | buffer[11];
        else
        {
            packet->program_map_PID = (buffer[10] << 3) << 5 | buffer[11];
        }
        n += 5;
    }
}
```

通过上面的分析，例子中的数据 00 B0 0D 00 01 C1 00 00 00 01 E0 20 A2 C3 29 41 就是具体的 PAT 表的内容，然后根据 PAT 结构体来具体分析 PAT 表。但是我们需要注意的是在 PAT 表里有 program_number、network_PID 的元素不只有一个，这两个元素是通过循环来确定的。循环的次数通过 section_length 元素的确定。在这个例子中 program_map_PID 为 20，所以下面来 PMT 分析时，就是查找 47 40 20 的开头的 TS 包。

下面来分析 PMT 表，先给出 PMT(Program Map Table)的结构体：

```
// PMT table
```

```
// Program Map Table
typedef struct TS_PMT
{
    unsigned table_id                   : 8;
    unsigned section_syntax_indicator       : 1;
    unsigned zero                       : 1;
    unsigned reserved_1                 : 2;
    unsigned section_length             : 12;
    unsigned program_number               : 16;
    unsigned reserved_2                 : 2;
    unsigned version_number               : 5;
    unsigned current_next_indicator         : 1;
    unsigned section_number             : 8;
    unsigned last_section_number        : 8;
    unsigned reserved_3                 : 3;
    unsigned PCR_PID                    : 13;
    unsigned reserved_4                 : 4;
    unsigned program_info_length          : 12;

    unsigned stream_type                : 8;
    unsigned reserved_5                 : 3;
    unsigned elementary_PID               : 13;
    unsigned reserved_6                 : 4;
    unsigned ES_info_length             : 12;
    unsigned CRC_32                     : 32;
} TS_PMT;
```

在给出调整字段函数：

```
// Adjust PMT table
void adjust_PMT_table ( TS_PMT * packet, char * buffer )
{
    int pos = 12, len = 0;
    int i = 0;
    packet->table_id                    = buffer[0];
    packet->section_syntax_indicator        = buffer[1] >> 7;
    packet->zero                        = buffer[1] >> 6;
    packet->reserved_1                  = buffer[1] >> 4;
    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;
    packet->reserved_4                      = buffer[10] >> 4;
    packet->program_info_length                 = (buffer[10] & 0x0F) << 8 | buffer[11];
    // Get CRC_32
    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);
    // 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; )
    {
       packet->stream_type                    = buffer[pos];
       packet->reserved_5                     = buffer[pos+1] >> 5;
       packet->elementary_PID                  = ((buffer[pos+1] << 8) | buffer[pos+2]) &
0x1FFF;
       packet->reserved_6                     = buffer[pos+3] >> 4;
        packet->ES_info_length                    =  (buffer[pos+3]  &  0x0F)  <<  8  |
buffer[pos+4];
       // Store in es
       es[i].type = packet->stream_type;
       es[i].pid = packet->elementary_PID;
       if ( packet->ES_info_length != 0 )
       {
          pos = pos+5;
          pos += packet->ES_info_length;
       }
       else
       {
          pos += 5;
       }
       i++;
    }
}
```

TS 流可以复合很多的节目的视频和音频，但是解码器是怎么来区分的呢？答案就在 PMT 表里，如其名节目映射表。他就是来解决这个问题的。现在看 PMT 结构体里的 stream_type、elementary_PID 这两个元素，前一个用来确定后一个作为标识 PID 的内容具体是什么，音频或视频等。还有要注意他们不只有一个，所以他们是通过循环读取来确保所有的值都被读取了，当然循环也是有规定的(具体看调整函数上)。从例子上来看，我们在倒数第三行找到了上面分析来的 PMT 表的 PID 为 0x20 的 TS 包。然后就可以把数据是用调整函数填入结构中。然后得到具体节目的 PID 为视频 0x21，音频 0x22。

PS.　文章里的 PID 是用来判断具体 TS 包是什么包的。分析每个包得到的 PID 值，都可以复合在 TS 头

部结构体的 PID 里。