16-09 TS协议补充

音视频高级开发课程：https://ke.qq.com/course/468797

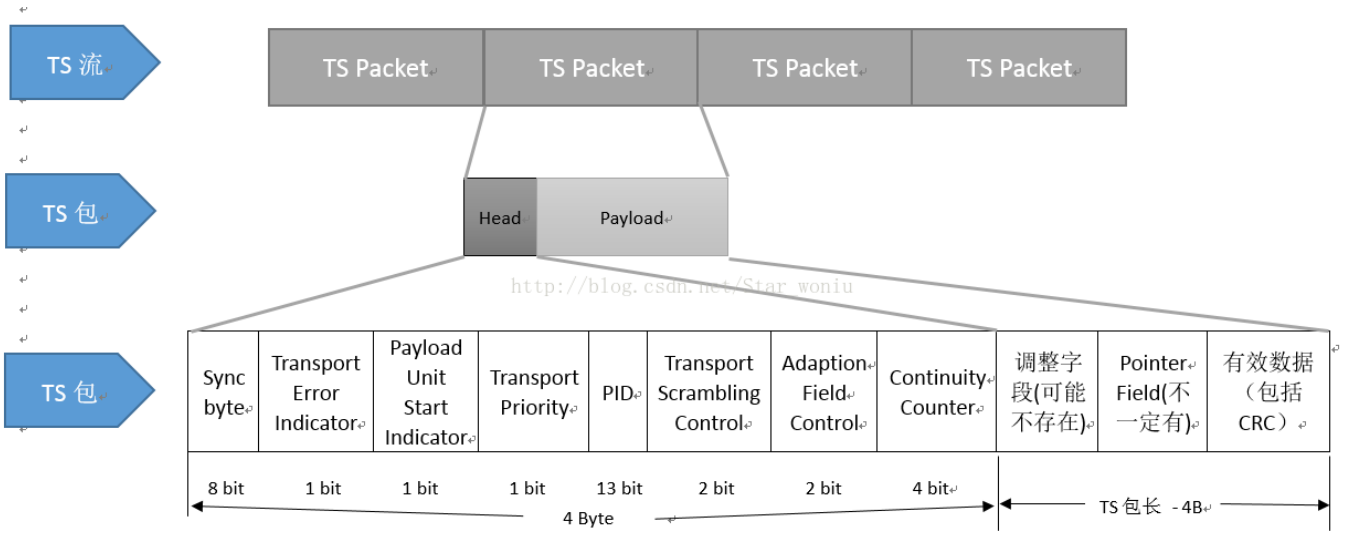
1. TS

1.1 TS流与其他流的关系

ES(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(Packetized Elementary Stream):分组的基本码流，**将基本码流ES流根据需要分成长度不等的数据包，并加上包头就形成了打包的基本码流PES流。是用来传输ES的一种数据结构。**
TS(Transport Stream):传输流，**是由固定长度的包组成，含有独立时间基准的一个或多个节目，适用于误码较多的环境，并且从流的任意一段开始都可以独立解码。**在MPEG-2系统中，由视频，音频的ES流和辅助数据复接生成的用于实际传输的标准信息流称为MPEG-2传送流。**个人理解，TS流是原始的PES流（音视频等）中按照一定的频率插入PSI/SI和一些标识符（辅助数据）信息，然后按固定长度打包形成的传输流。**值得注意的是，PSI/SI信息在TS流中并不是只发送一次，而是按照一定的频率插入码流，是重复发送的。
PS(Program Stream):节目流，PS流与TS流的区别在于，PS流的包结构是可变长度的，而TS流的包结构是固定长度的。

1.2 TS包

TS包的长度：188 B或204 B，204 B长度是在188B后面增加了16 B的CRC校验数据。



sync_byte: 1B，固定值0x47，TS包的标识符，正常的TS包在0x47的包头标识符往后188/204B之后仍然是0x47【下一个TS包的标识符】

transport_error_Indicator: 1bit，当其为1时，表示该TS包中至少有一个不可纠正的错误位，只有在错误纠正之后，该位才能重新置0【实际获取TS包之后，该位为1的包丢弃】

payload_unit_start_indicator: 1bit，对于PSI数据包，该位为1时，表示该TS包是某个Section的第一个包，并且该包含有pointer_field，该变量的值意义在于，除了调整字段之外，往后pointer_field个字节开始，才是有效数据。对于空包来说，该值为0。

transport_priority: 1bit，表示传输优先级，对于相同PID的TS包，该字段置1的TS包拥有更高的优先级。

PID：13bit，PID可以标识存储于TS包中有效净荷的数据的类型。PID用于TS包阶段用于鉴别各种PSI/SI信息表、电视节目，区分音视频的PES包等，是辨别码流信息性质的关键。

transport_scrambling_control:2bit，用来指示传送流包Payload的加扰方式。【传送流包首部包括调整字段，则不应被加扰；空包也不加扰。】

Transport_scrambling_control	描述
00	未加扰
01	用户定义
10	用户定义
11	用户定义

adaption_field_control：2bit，表示传送流包首部是否跟随调整字段/Payload【如果全部是调整字段则不含payload】

adaption_field_control	描述
00	为ISO/IEC未来使用保留
01	没有调整字段
10	没有payload，全部是调整字段

11	0–182B的调整字段后为有效净荷

continuity_counter: 4bit，随着具有相同PID的TS包增加而增加，当它达到最大（31）时，又恢复为0，如果adaption_field_control = 00/10，该连续计数器不增加，因为不含payload。

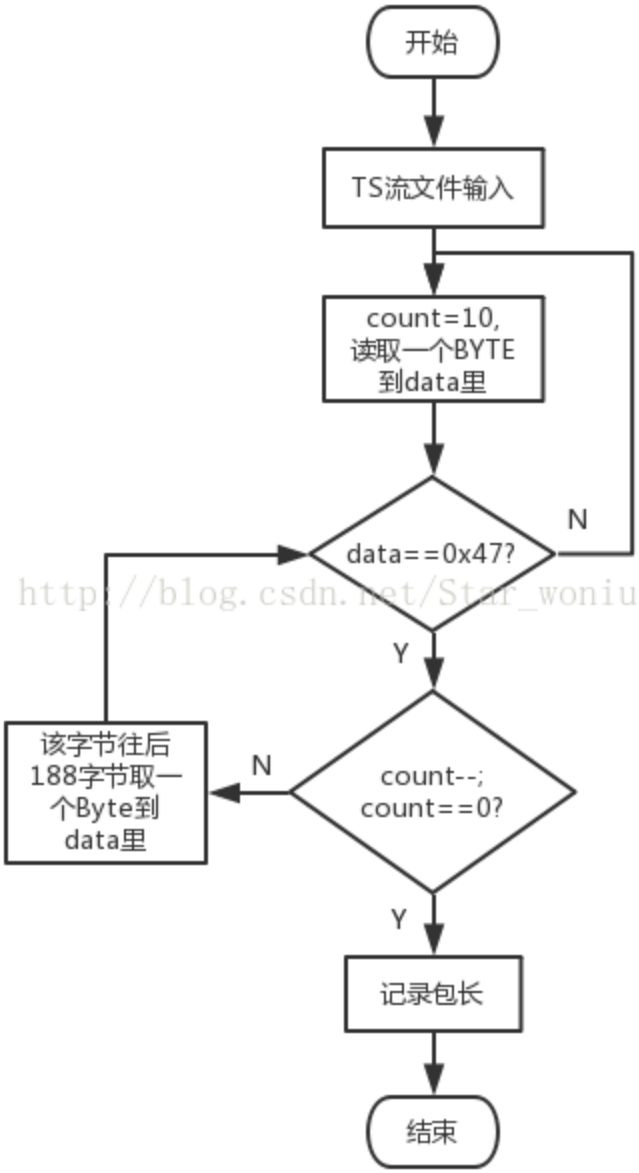
1.3解析TS包

1.3.1获取包长

TS包的包长有两种——188B或者204B，在解析TS包之前，必须要先判断TS包包长，以便后续进行分析。

本人设计获取包长的方法比较笨拙，在拿到第一个0x47数据之后，让文件指针往后188B，如果是0x47，就让指针继续往下，如此循环10次（可以更多），结果仍旧是0x47，就判断包长为188 B，否则用相同的方法判断204B，通过则包长为204，都不通过就对该文件继续往下搜索，用相同的方法判断包长。

具体流程如下图（以188 B为例）：



经过分析码流发现，大部分的TS包都是以188 B为指定长度的。

1.3.2 解析TS包头

在获取包长之后，就要对包头信息进行解析并获取有效数据，需要定义一个结构体存储数据：

```
1. /*TS包包头的结构体*/
2. typedef struct CSTSPacketHeader_S
3. {
4.  BYTE ucSyncByte;                                    //TS包的标识符
5.  BYTE ucTransport_error_indicator;                   //传输错误指示器，当值为时，表示该包有误
6.  BYTE ucPayload_unit_start_indicator;                //有效净荷开始标记
```

```
    位，当值为时，表示该包是某个section的开头，具有pointer_field 字段
7   7.  BYTE ucTransport_priority;              //传输的优先级
8   8.  WORD wPID;                              //TS包的ID，用于区分不同
    的section
9   9.  BYTE ucTransport_scrambling_control;        //指示ts传送流包有效
    净荷的加扰方式
10  10.     BYTE ucAdaptation_field_control;            //指示是否有调整
    字段和有效净荷
11  11.     BYTE ucContinuity_counter;              //随着相同PID TS包
    的增加而增加
12  12. }TSPacketHeader;
```

注：如果想要节省存储空间，可以使用位域的方式定义结构体。BYTE——unsigned char，Word——unsignedshort int

假定包长为188，我们每获取一个TS包，就装进一个长度为188的BYTE型数组里，前4个BYTE就是包头的数据了，获取数据可以逻辑与，左右移，逻辑或的方法进行，具体例子如下：

pstTSHeader->wPID = ((pucTSBuffer[1]& 0x1f) << 8) | pucTSBuffer[2];

1.3.3 判断TS包的有效性

在一个码流中，并不全部都是有效的TS包，需要将一些无效TS包剔除

无效TS包的情况分为五种：

（1） 该TS包往后188B不是0x47的包头标识符（TS包都是连续发送，如果出现包不连续的地方，说明该包数据传送时出错）；

（2） TS包存在错误，即transport_error_Indicator的值为1；

（3） TS包全是调整字段（空包），即adaption_field_control的值为10(二进制)；

（4） TS包的调整字段属于保留的情况，即adaption_field_control的值为00（二进制）；

（5） TS包被加扰，即transport_scrambling_control不为00，(如果有做解扰可以去掉这种情况）；

对于这五种情况的TS包我们一律丢弃，直接获取下一个TS包。

1.3.4 确定payload的起始位置

TS包中，Payload的起始位置并不是固定的，会受到调整字段和pointer_field的影响，解析获取包头信息之后就可以确定payload的起始位置payloadPosition了。

首先要判断是不是有调整字段，如果有payloadPosition = 5 + 调整字段长度。

如果没有 payloadPosition= 4。

其次要判断是不是有pointer_field，payload_unit_start_indicator= 1 则有，此时payloadPosition+= 1 + pointer_field;

注：1.这里算出的是数组的下标，从payloadPosition（包括payloadPosition下标）开始都是属于有效数据。
2.TS流里所有的长度都是从长度数据的下一个Byte开始算，比如section_length是5，就是从section_length的下一个Byte开始算，有5个字节的长度，所以第一种情况加的时候要加上调整字段长度本身的1个字节，还有包头4个字节，一共是5个。

2. Section

2.1 Section的概念

一个TS数据包的最大净荷为184个字节，当一个PSI/SI表的字节长度大于184字节时，就要对这个表进行分割，形成段（section）来传送。分段机制主要是将一个数据表分割成多个数据段。在PSI/SI表到TS包的转换过程中，段起到了中介的作用。由于一个数据包只有188字节，而段的长度是可变的，EIT表的段限长4096字节，其余PSI/SI表的段限长为1024字节。因此，一个段要分成几部分插入到TS包的payload中。从TS码流中可以获取到TS包，TS包要组成Section，才能提取到想要的信息，所以首先要懂得怎么组section。
组Section之前要了解TS包在码流中发送的一些情况：
（1） TS包发送的时候PID是无序的，连续的TS包的PID可能都是不一样的；
（2） TS包发送的时候Section是相对有序的，也就是说，对于同一个PID的TS包，只有发完了一个Section，才会发送下一个Section，不然无法区分该TS包属于哪一个Section,并且对于这个Section，TS包是有序发送的，否则数据会被打乱；
（3） 某个Section的第一个TS包有PSI/SI表的一些表头信息（table_id，section_length等信息），我称之为SectionHeader,后面的TS包就没有，所以接收某个Section必须先拿到首包。

2.2 TS包组Section

TS包组section首先要找到该section的第一个TS包(下面简称为首包)，首包含有该section的长度，可以用来判断一个section是不是组完了。通过判断TS包包头中的Payload Unit Start Indicator，该值为1

的话，就说明这个TS包是首包，可以开始组一个section，首包含有Section的头部，结构类似下图。

program_association_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf

拿到首包之后，要获取section的长度，有效数据的第二个字节的后四位和第三个字节组成一个12bit的字段，该值就是section_length后面数据的长度，如果算上前面三个字节，整个section的长度就是section_length += 3。将section_length和TS包有效长度进行对比，

(1)如果section_length > TS包的有效数据，证明后面还有其他的TS包，将section_length减去TS包有效数据长度，获得剩余长度；

(2)如果是section_length <= TS包的有效数据，证明该section已经结束了。

如果一个section还没组完，那么就要获取后续的TS包，后续的TS包应该是和原来相同PID，并且TS包头中continuity_counter要比原来的大1（31的话要变成0），拿到包后要与剩余长度进行对比，重复上面的步骤。

2.3 组多个Section和判全判重

对于一些PSI/SI表来说，由于数据较多，有时候不止一个section，怎么针对这个表将所有的section组全?

从上图可以看到首包里有个信息是last_section_number，这个字段表明了当前子表最后一个Section_number，也就是说，当前子表最多有last_section_number+1 个section（section_number从0开始），在获取同一个子表的section时，可以使用链表的形式，将多个section链接起来。

判全和判重：每一个section的首包信息中都有一个version_number的字段，表明当前子表的version，这个字段一旦发生变化，就表明子表发生了变化，旧版本的section就要被抛弃，重新获取新版本的section，如果版本没有发生变化，那么每获取一个section，就要判断这个section的section_number是否之前获取过，我们可以建立一个标记数组，每获取一个section，就把以section_number为下标的标记数组的值置1，表明获取过该section，如果这个标记数组下标从0到last_section_number的值都为1，证明所有的section都被收全了，如果获取了一个新的section，而其标记数组值为1，证明这个section是重复的，此时应该将它丢弃。

参考：https://blog.csdn.net/rell336/article/details/38109621
https://blog.csdn.net/rongdeguoqian/article/details/18214627