


【H.264/AVC视频编解码技术详解】五. H.264的码流封装格式

 blog.csdn.net/shagoneal/article/details/52098426

《H.264/AVC视频编解码技术详解》视频教程已经在“CSDN学院”上线，视频中详述了H.264的背景、标准协议和实现，并通过一个实战工程的形式对H.264的标准进行解析和实现，欢迎观看！

“纸上得来终觉浅，绝知此事要躬行”，只有自己按照标准文档以代码的形式操作一遍，才能对视频压缩编码标准的思想和方法有足够深刻的理解和体会！

链接地址：[H.264/AVC视频编解码技术详解](#)

本节视频免费

H.264的语法元素进行编码后，生成的输出数据都封装为NAL Unit进行传递，多个NAL Unit的数据组合在一起形成总的输出码流。对于不同的应用场景，NAL规定了一种通用的格式适应不同的传输封装类型。通常NAL Unit的传输格式分两大类：**字节流格式**和**RTP包格式**；

- **字节流格式**：字节流格式在H.264标准的协议文档中在Annex B中规定，是大多数编码器实现的默认输出格式。字节流格式以连续的bit字节的形式传输码流，因此必须从码流中获取NAL Unit。方法是在码流中识别NAL Unit的识别码：0x00 00 00 01或0x 00 00 01。**这一系列以使用流格式的H.264码流为主。**
- **RTP包格式**：包格式方法将NAL Unit按照RTP数据包的格式封装。使用RTP包格式不需要额外的分割识别码，在RTP包的封装信息中有相应的数据长度信息。此种封装格式在标准协议文档中没有明确规定，但在JM Decoder中做了一定处理。通常可以在NAL Unit的起始位置用一个固定长度的长度码表示整个NAL Unit的长度。

流格式的H.264码流的结构如下图所示：

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream() && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}	https://blog.csdn.net/shaqoneal	

NAL Unit的字节流格式：

- **leading_zero_8bits**：在第一个NAL单元之前的前缀0字节；
- **zero_byte**：一个字节的0字符；
- **start_code_prefix_one_3bytes**：3个字符的起始前缀符，值为0x 00 00 01，与 zero_byte共同构成一个0x 00 00 00 01的前缀符；
- **nal_unit**：表示一个NAL Unit的比特位；
- **trailing_zero_8bits**：结束0字符；

对于字节流格式的H.264码流，从原始码流中读取NAL Unit的方法为检测两个起始码0x 00 00 01或0x 00 00 00 01之间的数据即可。

下面给出一个简单的提取NAL UNIT的参考程序：

```
// FindNALContent.cpp：定义控制台应用程序的入口点。
//

#include "stdafx.h"

typedef unsigned char uint8;
using namespace std;

static int find_nal_prefix(FILE **pFileIn, vector<uint8> &nalBytes)
{
    FILE *pFile = *pFileIn;
    /*
    00 00 00 01 x x x x x 00 00 00 01
    */
    uint8 prefix[3] = { 0 };
    uint8 fileByte;
    /*
    [0][1][2] = {0 0 0} -> [1][2][0] = {0 0 0} -> [2][0][1] = {0 0 0}
    getc() = 1 -> 0 0 0 1
    [0][1][2] = {0 0 1} -> [1][2][0] = {0 0 1} -> [2][0][1] = {0 0 1}
    */
}
```

```

nalBytes.clear();

int pos = 0, getPrefix = 0;
for (int idx = 0; idx < 3; idx++)
{
    prefix[idx] = getc(pFile);
    nalBytes.push_back(prefix[idx]);
}

while (!feof(pFile))
{
    if ((prefix[pos % 3] == 0) && (prefix[(pos + 1) % 3] == 0) && (prefix[(pos + 2) % 3] == 1))
    {
        //0x 00 00 01 found
        getPrefix = 1;
        nalBytes.pop_back();
        nalBytes.pop_back();
        nalBytes.pop_back();
        break;
    }
    else if ((prefix[pos % 3] == 0) && (prefix[(pos + 1) % 3] == 0) && (prefix[(pos + 2) % 3] == 0))
    {
        if (1 == getc(pFile))
        {
            //0x 00 00 00 01 found
            getPrefix = 2;
            nalBytes.pop_back();
            nalBytes.pop_back();
            nalBytes.pop_back();
            break;
        }
    }
    else
    {
        fileByte = getc(pFile);
        prefix[(pos++) % 3] = fileByte;
        nalBytes.push_back(fileByte);
    }
}

return getPrefix;
}

int _tmain(int argc, _TCHAR* argv[])
{
    FILE *pFile_in = NULL;
    _tfopen_s(&pFile_in, argv[1], _T("rb"));
    if (!pFile_in)
    {
        printf("Error: Opening input file failed.\n");
    }

    vector<uint8> nalBytes;
    find_nal_prefix(&pFile_in, nalBytes);
}

```

```

find_nal_prefix(&pFile_in, nalBytes);

for (int idx = 0; idx < nalBytes.size(); idx++)
{
    printf("%x ", nalBytes.at(idx));
}
printf("\n");

find_nal_prefix(&pFile_in, nalBytes);

for (int idx = 0; idx < nalBytes.size(); idx++)
{
    printf("%x ", nalBytes.at(idx));
}
printf("\n");

find_nal_prefix(&pFile_in, nalBytes);

for (int idx = 0; idx < nalBytes.size(); idx++)
{
    printf("%x ", nalBytes.at(idx));
}
printf("\n");

fclose(pFile_in);
return 0;
}

```

- 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
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68
- 69
- 70
- 71
- 72
- 73
- 74
- 75
- 76
- 77
- 78
- 79
- 80
- 81
- 82

- 83
- 84
- 85
- 86
- 87
- 88
- 89
- 90
- 91
- 92
- 93
- 94
- 95
- 96
- 97
- 98
- 99
- 100
- 101
- 102