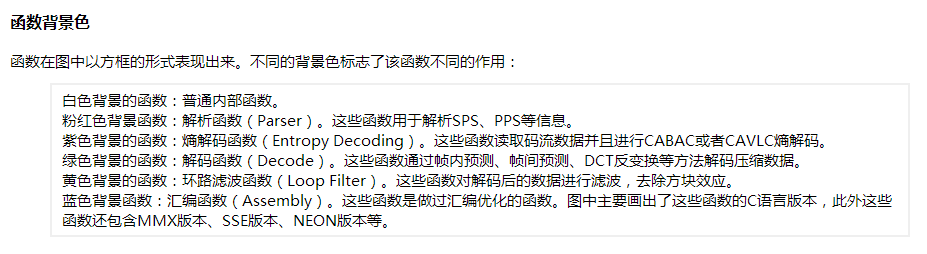
http://blog.csdn.net/lynchyo/article/details/41479519 编译

http://www.cnblogs.com/qq78292959/archive/2011/01/12/2076982.html 放在工程中编译

FFmpeg的H.264解码器源代码简单分析 <http://www.cnblogs.com/vijozsoft/p/5603298.html>



ffmpeg (转码时间戳pts dts问题)最简单的转码封装mp4文件 <http://blog.sina.com.cn/s/blog_8ae8fed10102wc84.html>

FFmpeg里有两种时间戳：DTS（Decoding Time Stamp）和PTS（Presentation Time Stamp）。 顾名思义，前者是解码的时间，后者是显示的时间。

需要先了解FFmpeg中的packet和frame的概念

FFmpeg中用AVPacket结构体来描述解码前或编码后的压缩包，用AVFrame结构体来描述解码后或编码前的信号帧。 对于视频来说，AVFrame就是视频的一帧图像。这帧图像什么时候显示给用户，就取决于它的PTS。DTS是AVPacket里的一个成员，表示这个压缩包应该什么时候被解码。 如果视频里各帧的编码是按输入顺序（也就是显示顺序）依次进行的，那么解码和显示时间应该是一致的。可事实上，在大多数编解码标准（如H.264或HEVC）中，编码顺序和输入顺序并不一致。 于是才会需要PTS和DTS这两种不同的时间戳。

视频的显示和存放原理

对于一个电影，帧是这样来显示的：I B B P。现在我们需要在显示B帧之前知道P帧中的信息。

因此，帧可能会按照这样的方式来存储：IPBB。

这就是为什么我们会有一个解码时间戳和一个显示时间戳的原因。解码时间戳告诉我们什么时候需要解码，显示时间戳告诉我们什么时候需要显示。

所以，在这种情况下，我们的流可以是这样的：

PTS: 1 4 2 3

DTS: 1 2 3 4

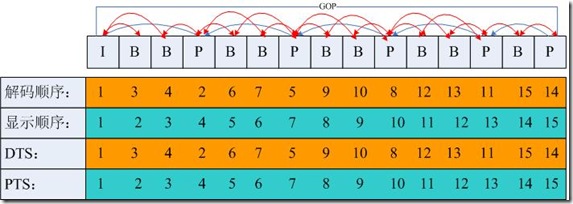
Stream: I P B B

通常PTS和DTS只有在流中有B帧的时候会不同。

DTS主要用于视频的解码,在解码阶段使用.PTS主要用于视频的同步和输出.在display的时候使用.在没有B frame的情况下.DTS和PTS的输出顺序是一样的.

**例子:**

下面给出一个GOP为15的例子,其解码的参照frame及其解码的顺序都在里面:

[](http://images.cnblogs.com/cnblogs_com/qingquan/201107/201107272227028458.jpg)(红色镜头 代表该帧被谁引用(指向引用它的帧))

如上图：I frame 的解码不依赖于任何的其它的帧.而p frame的解码则依赖于其前面的I frame或者P frame.B frame的解码则依赖于其前的最近的一个I frame或者P frame 及其后的最近的一个P frame.

<http://wenku.baidu.com/view/433e618933687e21ae45a945.html?from=search>

/\*\*

\* 分配一个image 根据宽w高h 和像素格式pix\_fmt 以及 填充参数pointers 和 linesizes .

\* 分配的 image buffer 需要通过av\_freep(&pointers[0]).来释放

\*

\* @param align 指定分配image buffer大小的对齐

\* @return 字节大小 image buffer的大小

\*/

int av\_image\_alloc(uint8\_t \*pointers[4], int linesizes[4],

int w, int h, enum AVPixelFormat pix\_fmt, int align);

SwsContext \*c 是sws\_getContext成功申请到的SwsContext结构指针；  
   const uint8\_t \*const srcSlice[]是输入图像各分量数据起始指针。  
          例如有3个分量的YUV420P格式，srcSlice[4] = {Y\_data, U\_data, V\_data, NULL};  
                 有1个分量的YUYV422格式  srcSlice[4] = {data, NULL, NULL, NULL};  
          因为有可能有ARGB四个通道，所以srcSlice, srcStride一般设为4维，用不到的就设为NULL/0；  
   const int srcStride[]是输入图像各分量一行数据量。stride = width\*Bpp      
         例如有3个分量的YUV420P格式，srcStride[4] = {width, width/2, width/2, 0};  
                有1个分量的YUYV422格式  srcStride[4] = {2\*width, 0, 0, 0};  
          对于RGB24 srcStride[4] = {3\*width, 0, 0, 0};  
   int srcSliceY 当前处理区域在图像中的起始纵坐标，  
   int srcSliceH 当前处理区域的高度。         
       srcSliceY 和 srcSliceH定义出图像中的一个条带，  
       这样做的目的可能是为了并行化，可以将整帧图像划分成N个条带，送给N个线程并行处理，加快速度。  
       如果不考虑这种并行性，srcSliceY = 0， srcSliceH = srcHeight。  
   uint8\_t \*const dst[]是输出图像各分量数据起始指针。它的定义原则和srcSlice一样。  
   const int dstStride[]是输出图像各分量一行数据量。它的定义原则和srcStride一样。  
   输出：在输出图像中做了多少行，跟srcSliceH相对应。  
   这个函数用起来很简单，注意srcSlice，srcStride。

调用ffmpeg的 libswscale.so实现图像格式转换及大小缩放

<http://www.bubuko.com/infodetail-479238.html>

测试每种格式有由多少个slice/plane组成 ，也就是知道颜色空间 对应的pointers[4], int linesizes[4] 应该填什么 比如NV12 pointers[0] [1] 都是存放数据的

static int test\_once = 0 ;

if( ++test\_once == 1 ){

unsigned char \*src\_data[4] = {0};

int src\_linesize[4] = {0};

unsigned char \*dst\_data[4] = {0};

int dst\_linesize[4] = {0};

int in\_buffer\_size = av\_image\_alloc(src\_data, src\_linesize, 640, 480, AV\_PIX\_FMT\_RGBA, 1);

int out\_buffer\_size = av\_image\_alloc(dst\_data, dst\_linesize, 640, 480, AV\_PIX\_FMT\_NV12, 1);

STREAMER\_TRACE(" in\_buffer\_size = %d ,out\_buffer\_size = %d\n "

"src %p %p %p %p %d %d %d %d \n"

"dst %p %p %p %p %d %d %d %d \n",

in\_buffer\_size , out\_buffer\_size ,

src\_data[0],src\_data[1],src\_data[2],src\_data[3],

src\_linesize[0],src\_linesize[1],src\_linesize[2],src\_linesize[3],

dst\_data[0],dst\_data[1],dst\_data[2],dst\_data[3],

dst\_linesize[0],dst\_linesize[1],dst\_linesize[2],dst\_linesize[3]

);

av\_freep(&src\_data[0]);

av\_freep(&dst\_data[0]);

//in\_buffer\_size = 1228800 ,out\_buffer\_size = 460800

//src 0xdec03000 0x0 0x0 0x0 2560 0 0 0

//dst 0xeff8a000 0xeffd5000 0x0 0x0 640 640 0 0

}