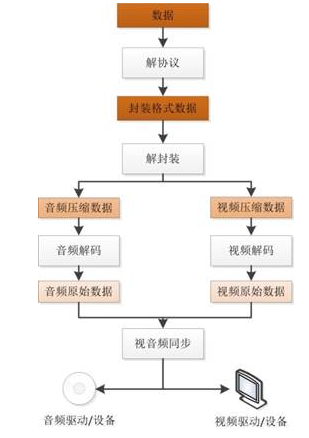
网页：http://blog.csdn.net/leixiaohua1020/article/details/18893769

音频文件都有不同的格式：avi，rmvb，MP4，flv，mkv等等。这里的格式代表封装格式。具体看不出使用了什么音频编码标准。有些封装格式支持的音频编码标准十分广泛，有的则非常少。

播放器原理：视屏播放器播放互联网上的视屏文件时，需要进过以下几个步骤：解协议，解封装，解码音频，视音频同步。而本地文件不需要解协议。

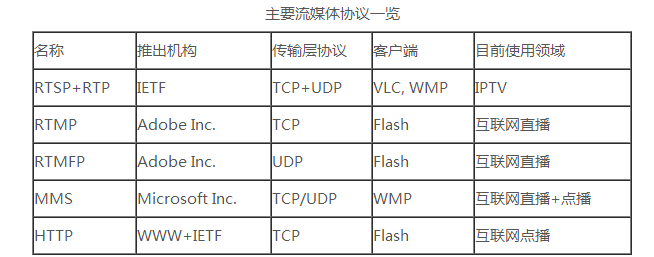
**解协议**的作用，就是将流媒体协议的数据，解析为标准的相应的封装格式数据。视音频在网络上传播的时候，常常采用各种流媒体协议，例如HTTP，RTMP，或是MMS等等。这些协议在传输视音频数据的同时，也会传输一些信令数据。这些信令数据包括对播放的控制（播放，暂停，停止），或者对网络状态的描述等。解协议的过程中会去除掉信令数据而只保留视音频数据。例如，采用RTMP协议传输的数据，经过解协议操作后，输出FLV格式的数据。

**解封装**的作用，就是将输入的封装格式的数据，分离成为音频流压缩编码数据和视频流压缩编码数据。封装格式种类很多，例如MP4，MKV，RMVB，TS，FLV，AVI等等，它的作用就是将已经压缩编码的视频数据和音频数据按照一定的格式放到一起。例如，FLV格式的数据，经过解封装操作后，输出H.264编码的视频码流和AAC编码的音频码流。

**解码**的作用，就是将视频/音频压缩编码数据，解码成为非压缩的视频/音频原始数据。音频的压缩编码标准包含AAC，MP3，AC-3等等，视频的压缩编码标准则包含H.264，MPEG2，VC-1等等。解码是整个系统中最重要也是最复杂的一个环节。通过解码，压缩编码的视频数据输出成为非压缩的颜色数据，例如YUV420P，RGB等等；压缩编码的音频数据输出成为非压缩的音频抽样数据，例如PCM数据。

**视音频同步**的作用，就是根据解封装模块处理过程中获取到的参数信息，同步解码出来的视频和音频数据，并将视频音频数据送至系统的显卡和声卡播放出来

流媒体协议：



RTSP+RTP经常用于IPTV领域。因为其采用UDP传输视音频，支持组播，效率较高。但其缺点是网络不好的情况下可能会丢包，影响视频观看质量。

因为互联网网络环境的不稳定性，RTSP+RTP较少用于互联网视音频传输。互联网视频服务通常采用TCP作为其流媒体的传输层协议，因而像RTMP，MMS，HTTP这类的协议广泛用于互联网视音频服务之中。这类协议不会发生丢包，因而保证了视频的质量，但是传输的效率会相对低一些。

封装格式：

封装格式的主要作用是把视频码流和音频码流按照一定的格式存储在一个文件中。



由表可见，除了AVI之外，其他封装格式都支持流媒体，即可以“边下边播”。有些格式更“万能”一些，支持的视音频编码标准多一些，比如MKV。而有些格式则支持的相对比较少，比如说RMVB。

**视频编码：**

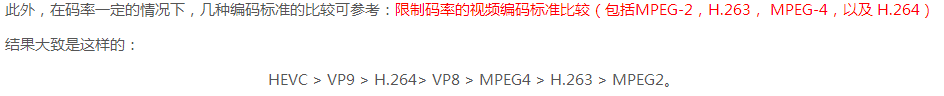
视频编码的主要作用是将视频像素数据（RGB，YUV等）压缩成为视频码流，从而降低视频的数据量。如果视频不经过压缩编码的话，体积通常是非常大的，一部电影可能就要上百G的空间。视频编码是视音频技术中最重要的技术之一。视频码流的数据量占了视音频总数据量的绝大部分。高效率的视频编码在同等的码率下，可以获得更高的视频质量。

****

### ****主流编码标准****

H.264仅仅是一个编码标准，而不是一个具体的编码器，H.264只是给编码器的实现提供参照用的。

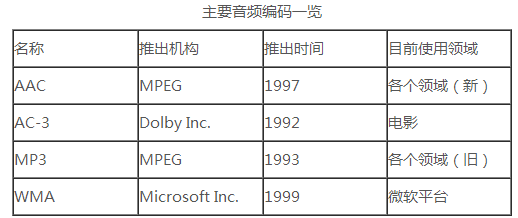
使用最多的编码器是x264了，性能强悍（超过了很多商业编码器），而且开源。其基本教程网上极多，不再赘述。



但是，HEVC、VP9正在研发阶段。

## ****音频编码****

音频编码的主要作用是将音频采样数据（PCM等）压缩成为音频码流，从而降低音频的数据量



YUV存储格式：

YUV是一种颜色编码方法，主要用于电视系统及模拟视屏领域，将亮度参量Y和颜色参量UV分开表示的像素编码格式。

YUV420P文件：如果视频帧的宽和高分别为w和h，那么一帧YUV420P像素数据一共占用w\*h\*3/2 Byte的数据。其中前w\*h Byte存储Y，接着的w\*h\*1/4 Byte存储U，最后w\*h\*1/4 Byte存储V。纯Y数据，分辨率为256x256。纯U数据，分辨率为128x128。纯V数据，分辨率为128x128。

YUV444P文件：如果视频帧的宽和高分别为w和h，那么一帧YUV444P像素数据一共占用w\*h\*3 Byte的数据。其中前w\*h Byte存储Y，接着的w\*h Byte存储U，最后w\*h Byte存储V。纯Y数据，分辨率为256x256。纯U数据，分辨率为256x256。纯V数据，分辨率为256x256。

把YUV420消除颜色：如果想把YUV格式像素数据变成灰度图像，只需要将U、V分量设置成128即可。这是因为U、V是图像中的经过偏置处理的色度分量。色度分量在偏置处理前的取值范围是-128至127，这时候的无色对应的是“0”值。经过偏置后色度分量取值变成了0至255，因而此时的无色对应的就是128了。

把YUV420图片的亮度降低一把：如果打算将图像的亮度减半，只要将图像的每个像素的Y值取出来分别进行除以2的工作就可以了。图像的每个Y值占用1 Byte，取值范围是0至255。

RGB24图像存储格式：YUV420P三个分量分开存储不同，RGB24格式的每个像素的三个分量是连续存储的。一帧宽高分别为w、h的RGB24图像一共占用w\*h\*3 Byte的存储空间。RGB24格式规定首先存储第一个像素的R、G、B，然后存储第二个像素的R、G、B…以此类推。类似于YUV420P的存储方式称为Planar方式，而类似于RGB24的存储方式称为Packed方式。上述调用函数的代码运行后，将会把一张分辨率为500x500的的RGB24格式的像素数据文件分离成为三个文件：

output\_r.y：R数据，分辨率为256x256。  
output\_g.y：G数据，分辨率为256x256。

output\_b.y：B数据，分辨率为256x256。

将RGB图像封装成MBP文件：需要在RGB数据外进行一些必要的封装。代码如下：

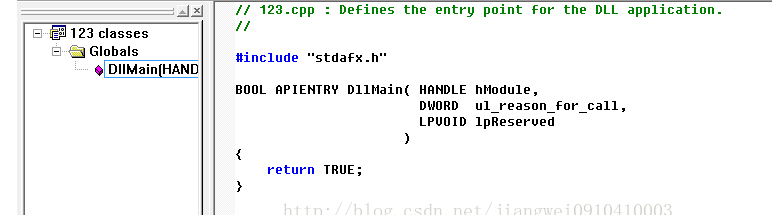
1. **int** simplest\_rgb24\_to\_bmp(**const** **char** \*rgb24path,**int** width,**int** height,**const** **char** \*bmppath){
2. **typedef** **struct**
3. {
4. **long** imageSize;
5. **long** blank;
6. **long** startPosition;
7. }BmpHead;
9. **typedef** **struct**
10. {
11. **long**  Length;
12. **long**  width;
13. **long**  height;
14. unsigned **short**  colorPlane;
15. unsigned **short**  bitColor;
16. **long**  zipFormat;
17. **long**  realSize;
18. **long**  xPels;
19. **long**  yPels;
20. **long**  colorUse;
21. **long**  colorImportant;
22. }InfoHead;
24. **int** i=0,j=0;
25. BmpHead m\_BMPHeader={0};
26. InfoHead  m\_BMPInfoHeader={0};
27. **char** bfType[2]={'B','M'};
28. **int** header\_size=**sizeof**(bfType)+**sizeof**(BmpHead)+**sizeof**(InfoHead);
29. unsigned **char** \*rgb24\_buffer=NULL;
30. **FILE** \*fp\_rgb24=NULL,\*fp\_bmp=NULL;
32. **if**((fp\_rgb24=fopen(rgb24path,"rb"))==NULL){
33. printf("Error: Cannot open input RGB24 file.\n");
34. **return** -1;
35. }
36. **if**((fp\_bmp=fopen(bmppath,"wb"))==NULL){
37. printf("Error: Cannot open output BMP file.\n");
38. **return** -1;
39. }
41. rgb24\_buffer=(unsigned **char** \*)malloc(width\*height\*3);
42. fread(rgb24\_buffer,1,width\*height\*3,fp\_rgb24);
44. m\_BMPHeader.imageSize=3\*width\*height+header\_size;
45. m\_BMPHeader.startPosition=header\_size;
47. m\_BMPInfoHeader.Length=**sizeof**(InfoHead);
48. m\_BMPInfoHeader.width=width;
49. //BMP storage pixel data in opposite direction of Y-axis (from bottom to top).
50. m\_BMPInfoHeader.height=-height;
51. m\_BMPInfoHeader.colorPlane=1;
52. m\_BMPInfoHeader.bitColor=24;
53. m\_BMPInfoHeader.realSize=3\*width\*height;
55. fwrite(bfType,1,**sizeof**(bfType),fp\_bmp);
56. fwrite(&m\_BMPHeader,1,**sizeof**(m\_BMPHeader),fp\_bmp);
57. fwrite(&m\_BMPInfoHeader,1,**sizeof**(m\_BMPInfoHeader),fp\_bmp);
59. //BMP save R1|G1|B1,R2|G2|B2 as B1|G1|R1,B2|G2|R2
60. //It saves pixel data in Little Endian
61. //So we change 'R' and 'B'
62. **for**(j =0;j<height;j++){
63. **for**(i=0;i<width;i++){
64. **char** temp=rgb24\_buffer[(j\*width+i)\*3+2];
65. rgb24\_buffer[(j\*width+i)\*3+2]=rgb24\_buffer[(j\*width+i)\*3+0];
66. rgb24\_buffer[(j\*width+i)\*3+0]=temp;
67. }
68. }
69. fwrite(rgb24\_buffer,3\*width\*height,1,fp\_bmp);
70. fclose(fp\_rgb24);
71. fclose(fp\_bmp);
72. free(rgb24\_buffer);
73. printf("Finish generate %s!\n",bmppath);
74. **return** 0;
75. **return** 0;
76. }

需要注意的是MBP文件在存储是并非R、G、B进行存储，而是B、G、R，所以在代码62#需要for循环进行交换。

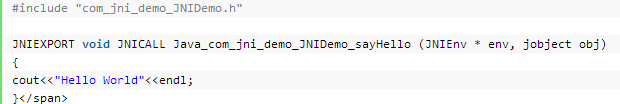
使用JNI:

<http://blog.csdn.net/jiangwei0910410003/article/details/17465085>

1. 使用javah生成头文件时，需要进入workspace/项目/src文件夹下，使用javah 包名.文件名。
2. 使用VC++6.0创建win32 Dynamic-Link Library，在下一个步骤中，选择”一个简单的DLL工程”



1. 将生成的头文件复制到VC++6.0的项目的工作路径下，并将里面的生成的方法复制到.cpp文件中，并实现该方法。



1. 需要将这两个文件也放入项目中，(可以放到vc目录下的include文件下D:\VC\VC98\Include)
2. 出现一个错误就是，提示：e:\c++\jnidemo\jnidemo.cpp(9) : fatal error C1010: unexpected end of file while looking for precompiled header directive，这个是预编译头文件读写错误，这时候还要在VC中进行设置:项目-》设置-》C/C++;在分类中选择预编译头文件，选择不使用预补偿页眉
3. 这样编译成功够将Debug下的.dll文件存放在一个path环境变量的文件夹下，这样可以让JVM能找到该dll文件



调用java类中的方法：

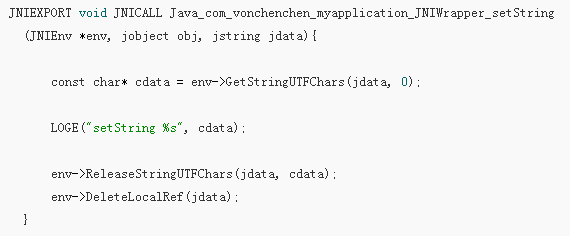


设置对象中的常数：



JNI中基本数据类型转换：

String 🡪const char \*:

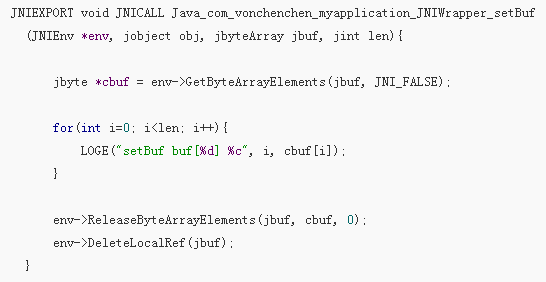


注意这样需要回收：

env->ReleaseStringUTFChars(jdata, cdata);

env->DeleteLocalRef(jdata);

java向native传递数组类型：



任然需要回收：

env->ReleaseByteArrayElements(jbyteArray, cbuf, 0);

env->DeleteLocalRef(jbyteArray);

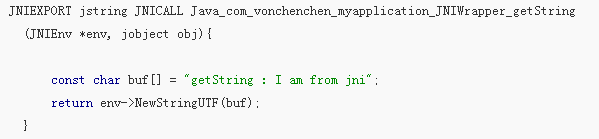
java向native传递自定义java对象

不需要进行转换，如果需要访问对象的方法、成员变量，则先获得id，在获得id对应的数据。

native向java传递数组类型



native向java传递字符串类型

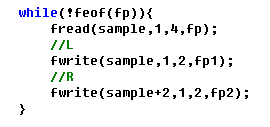


native向java传递java对象：



## PCM数据描述

PCM16LE双声道数据数据中，一个采样值有两个声道，左右声道分别占2个字节。



PCM16LE数据，“16”代表采样位数是16bit，所以一个声道的一个采样值占用2Byte。“LE”代表Little Endian，代表2 Byte采样值的存储方式为高位存在高地址中。

左右声道所占用2Byte可以为整数可以为负数，如果想将音量降低一半可以将其减少一半。这里使用C语言减少一般时需要使用short。注意不是Unsigned short。因为这里的数有正有负。

