MPEG 音频文件格式(包括 MP3 文件格式)详解

MP3 文件是由帧(frame)构成的，帧是 MP3 文件最小的组成单位。MP3 的全称应为 MPEG1 Layer-3 音频文件，MPEG(Moving Picture Experts Group)在汉语中译为活动图像专家组，特指活动影音压缩标准，MPEG 音频文件是 MPEG1 标准中的声音部分，也叫 MPEG 音频层，它根据压缩质量和编码复杂程度划分为三层，即 Layer-1、Layer2、Layer3，且分别对应 MP1、MP2、MP3 这三种声音文件，并根据不同的用途，使用不同层 次的编码。MPEG 音频编码的层次越高，编码器越复杂，压缩率也越高，MP1 和 MP2 的压缩率分别为 4：1 和 6：1-8：1，而 MP3 的压缩率则高达 10：1-12：1，也就是说，一分钟 CD 音质的音乐，未经压缩需要 10MB 的存储空间，而经过 MP3 压缩编码后只有 1MB 左右。不过 MP3 对音频信号采用的是有损压缩方式，为了降低声音失真度，MP3 采取了“感官编码技术”，即编码时先对音频文件进行频谱分析，然后用过滤器滤掉噪音电平，接着通过量化的方式将剩下的每一位打散排列，最后形成具有较高压缩比的 MP3 文件，并使压缩后的文件在回放时能够达到比较接近原音源的声音效果。

一、MPEG 音频压缩基础

在众多音频压缩方法中，这些方法在保持声音质量的同时尽量压缩数字音频使之占用更小的存储空间。MPEG 压缩是该领域中效果最好的一个。这种压缩是有损压缩，这意味着，当运用这一方法压缩时肯定会丢失一部分音频信息。但是，由于压缩方法的控制很难发现这种损失。使用几个非常复杂和苛刻的数学算法，使得只有原始音频中几乎听不到的部分损失掉。这就给重要的信息剩下了更多的空间。通过这种方法可以将音频压缩 12 倍（可以选择压缩率），效果显著。正是应为他的质量，MPEG 音频变得流行起来。
MPEG-1，MPEG-2 和 MPEG-4 都是人们熟悉的 MPEG 标准，MP3 只涉及到前两中，另外还有一个非官方标准 MPEG-2.5 用于扩展 MPEG-2/LSF 到更低的采样率。
MPEG-1 音频（ISO/IEC 11172-3）描述了具有如下属性的三层音频编码：
1 或 2 个声道
采样频率为 32kHz，44.1kHz 或 48kHz
位率从 32kbps 到 448kbps
每一层都有自己的优点。
MPEG-2 音频（ISO/IEC 13818-3）有两个 MPEG-1 的扩展，通常叫做 MPEG-2/LSF 和 MPEG-2/Multichannel
MPEG-2/LSF 有如下特点：
1 或 2 个声道
采样频率为 MPEG-1 的一半
波特率从 8kbps256kbps
MPEG-2/Mutichannel 有如下特点：
多达 5 个声道和 1 个 LFE-通道（低频增强 不是重低音）
同 MPEG-1 一样的采样频率
5.1 的最高波特率可能达到 1Mbps

二、MPEG Layer3 编/解码的基本原理

音乐 CD 具有 44.1KHz 16Bits 立体声的音频质量，一张 CD 可以存储 74 分钟的歌曲(大

约 15 首左右)。如何将这些歌曲无损或基本无损地进行压缩,以使在同样的媒体上存储更多的歌曲,一直困扰着软件业。当 MPEG 协会提出 MPEG Audio Layer1～Layer3 后，机会产生了。通过使用 MPEG1 Layer3 编码技术，制作者得以用大约 12：1 的压缩率记录 16KHz 带宽的有损音乐信号。不过,同 CD 原声区别不大。人的听力系统具有非常优越的性能，其动态范围超过 96dB。你既可以听到扣子掉在地上这样小的声音，也可以听到波音 747 的强大的轰鸣声。但当我们站在飞机场听着波音 747 的轰鸣时,你还能分辨出扣子掉在地上的声音吗？不可能。人的听力系统适应声音的动态变化，人们对这种适应及屏蔽特性音质研究后得出对声音压缩非常有用的理论。人们很早以前就知道利用这种特性来为磁带录音降低噪音了(当没有音乐时嘶嘶声很容易听到，而当音乐信号电平很高时嘶嘶声不容易听到)。当声音较强时产生屏蔽效应。在阈值曲线下的噪音或小信号声音无法被人耳听到。在较强信号出现时，允许通过更多的信号。在此时增加被量化过的小信号数据(使用无用的位来携带更多的信息)可以达到一定程度的压缩的目的。通常情况下,MP3 压缩器将原始声音通过 FFT(快速傅立叶变换)变化到频域，然后通过一定的算法算出何种频率声音可以携带更多的信息。而在还原时解码器所需要做的仅仅是将其从频域再变换回来。

三、整个 MP3 文件结构：

MP3 文件大体分为三部分：TAG_V2(ID3V2)，Frame, TAG_V1(ID3V1)

ID3V2	包含了作者，作曲，专辑等信息，长度不固定，扩展了ID3V1的信息量。
Frame . . . Frame	一系列的帧，个数由文件大小和帧长决定 每个FRAME的长度可能不固定，也可能固定，由位率bitrate决定 每个FRAME又分为帧头和数据实体两部分 帧头记录了mp3的位率，采样率，版本等信息，每个帧之间相互独立
ID3V1	包含了作者，作曲，专辑等信息，长度为128BYTE。

四、MPEG 音频帧格式

一个 MPEG 音频文件是许多的称为帧的较小部分组成的，通常，帧是独立的组成部分。每一帧都拥有自己的头和音频信息。没有文件头。所以，我们可以剪切 MPEG 文件的任何部分并且能够正常播放（当然要分割到帧的结束处尽管许多程序会处理错误头）。在 LayerIII 中就并不是 100%正确的。这是因为在 MPEG-1LayerIII 文件中的数据组织中，帧常常是互相关联的并且不能那样随便裁切。

当你想读取 MPEG 文件的信息时，通常只找到第一帧就足够了，读取它的头信息然后假设其它帧是相同的就可以。但这也不是所有情况。变比特率的 MPEG 文件使用使用所谓比特变换，也就是说每一帧的比特率依照具体内容变化。这种方法没有减少声音质量的帧将应用较低的波特率。这样就允许更好的压缩质量的同时又保证了高质量的音质。

帧头由每一帧的前 4 个字节（32 位）组成。帧头的前 11 比特（或前 12 个位，见下文关于

帧同步）总是固定的称作“帧同步”。因此，可以在整个文件中查找第一个帧同步（即：必须找到一个值为 255 的且其后跟着三到四个最高位置 1 的字节。）然后读取整个头检查值是否正确。关于头中每一个比特的具体含义应该验证那一个值的有效性可以操看下面的表格，如果存在被定义为保留，无效，损坏或不允许的值表明该头已经损坏。记住，光有这些是不够的，帧同步能在许多二进制文件里面的应用是很广的。而且，MPEG 文件可能在开头包含可能有错误同步信息的垃圾，所以我们必须检查两个或者更多一些帧来确定我们现在读取的文件是一个 MPEG 文件。
帧可能还有 CRC 校验。如果存在的话，CRC 校验紧跟在帧头之后，长为 16 比特。CRC 校验之后是音频数据。计算出帧长度，如果你需要读取其他头或者计算该帧的 CRC 值，可以使用它比较文件中读出来的帧。验证 MPEG 头的有效性这是一个非常好的方法。

1、帧头格式

下面是一个头内容图示，使用字符 A 到 M 表示不同的区域。在表格中你可以看到每一区域的详细内容。

AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM

<table>
<tr><th>符号i</th><th>长度(bits)</th><th>位置(bits)</th><th>描述</th></tr>
<tr><td>A</td><td>11</td><td>(31-21)</td><td>帧同步（所有位置1）</td></tr>
<tr><td>B</td><td>2</td><td>(20,19)</td><td>MPEG 音频版本ID
00 – MPEG 2.5
01 – 保留
10 – MPEG 2 （ISO/IEC 13818-3）
11 – MPEG 1 （ISO/IEC 11172-3）
注：MPEG 2.5不是官方标准。帧头第20个比特用来表示2.5版本。不支持该版本的应用程序一般认为该比特位置位为帧同步位，也就是说帧同步（A）的长度为12而不是这里规定的11，这样B也就变成了1位（第19个位）。推荐使用该表的方法因为这样允许你可以区分三个版本以获得最高兼容性。</td></tr>
<tr><td>C</td><td>2</td><td>(18,17)</td><td>Layer描述
00 - 保留
01 - Layer III
10 - Layer II
11 - Layer I</td></tr>
<tr><td>D</td><td>1</td><td>(16)</td><td>校验位
0 - 紧跟帧头后有16位即2个字节用作CRC校验
1 - 没有校验</td></tr>
<tr><td>E</td><td>4</td><td>(15,12)</td><td>位率索引
<table>
<tr><th rowspan="2">索引值</th><th colspan="3">MPEG 1</th><th colspan="2">MPEG 2, 2.5 (LSF)</th></tr>
<tr><th>Layer I</th><th>Layer II</th><th>Layer III</th><th>Layer I</th><th>Layer II & III</th></tr>
<tr><td>0000</td><td colspan="5">Free</td></tr>
<tr><td>0001</td><td>32</td><td>32</td><td>32</td><td>32</td><td>8</td></tr>
</table></td></tr>
</table>

0010	64	48	40	48	16
0011	96	56	48	56	24
0100	128	64	56	64	32
0101	160	80	64	80	40
0110	192	96	80	96	48
0111	224	112	96	112	56
1000	256	128	112	128	64
1001	288	160	128	144	80
1010	320	192	160	160	96
1011	352	224	192	176	112
1100	384	256	224	192	128
1101	416	320	256	224	144
1110	448	384	320	256	160
1111	Bad				

注：所有值单位为kbps，而且1kbit=1000bit而不是1024bit

Free表示空闲，如果固定比特率（这种文件不能变换比特率）和上表定义的不同，应该有应用程序决定。这种情况的实现应该只用于内部目的因为第三方应用程序是没有办法找出正确比特率的。但是这么做并不是很重要况且还浪费精力。Bad表示该值无效。
MPEG文件可以有VBR。表示文件的比特率可以变化。我已经知道了两种惯用方法：
比特率变换(bitrate switching)：每一帧都创建成不同的比特率。可以应用在任何层。LayerIII解码器必须支持该方法。LayerI和LayerII也可以支持。

比特池(bit reservoir)：比特率可以使从前面的帧中借来的（受限），以便腾出空间来容纳输入信号部分。然而这样就导致各帧之间不再相互独立，意味着不能随便分割文件。这种方法只有LayerIII支持。

LyaerII中有一些不被允许比特率组合和模式。下表是允许的组合。

bitrate	allowed modes
free	all
32	single channel
48	single channel
56	single channel
64	all
80	single channel

<table>
<tr><td></td><td></td><td></td><td>
<table>
<tr><td>96</td><td>all</td></tr>
<tr><td>112</td><td>all</td></tr>
<tr><td>128</td><td>all</td></tr>
<tr><td>160</td><td>all</td></tr>
<tr><td>192</td><td>all</td></tr>
<tr><td>224</td><td>stereo, intensity stereo, dual channel</td></tr>
<tr><td>256</td><td>stereo, intensity stereo, dual channel</td></tr>
<tr><td>320</td><td>stereo, intensity stereo, dual channel</td></tr>
<tr><td>384</td><td>stereo, intensity stereo, dual channel</td></tr>
</table>
</td></tr>
<tr><td>F</td><td>2</td><td>(11,10)</td><td>采样频率（单位：Hz）
<table>
<tr><td>bits</td><td>MPEG1</td><td>MPEG2</td><td>MPEG2.5</td></tr>
<tr><td>00</td><td>44100</td><td>22050</td><td>11025</td></tr>
<tr><td>01</td><td>48000</td><td>24000</td><td>12000</td></tr>
<tr><td>10</td><td>32000</td><td>16000</td><td>8000</td></tr>
<tr><td>11</td><td colspan="3">保留</td></tr>
</table>
</td></tr>
<tr><td>G</td><td>1</td><td>(9)</td><td>填充位
0 – 没有填充
1 – 填充了一个额外的空位
填充用来达到正确的比特率。例如：128k 44.1kHz LayerII使用了很多418bit或417bit长的帧来达到正确的128k比特率。LyaerI的空位有32bit长，LayerII和LayerIII的空位有8bit长。</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>H</td><td>1</td><td>(8)</td><td>私有bit，可以用来做特殊应用。例如可以用来触发应用程序的特殊事件。</td></tr>
<tr><td>I</td><td>2</td><td>(7,6)</td><td>声道
00 立体声
01 联合立体声（立体声）
10 双声道（立体声）
11 单声道（单声）

注：双声道文件由二个独立的单声道组成。 每一个声道使用整个文件一半的位率。大多数的解码器把它当作立体声来输出，但是它并不总是这种情况。按我的理解就是是两个声道的信息是完全相同的，并不能把它当作立体声看待。</td></tr>
<tr><td>J</td><td>2</td><td>(5,4)</td><td>扩展模式（仅在联合立体声时有效）
扩展模式用来连接对立体声效果无用的信息，来减少所需的资源。这两个位在联合立体声模式下有编码器动态指定。
完整的MPEG文件的频率序列分成有32个子带。在LayerI和LayerII中这两个位确定强度立体声应用的频带。
LayerIII中这两个位确定应用了哪一种联合立体声（M/S stereo或者Intensity stereo）频带由解压算法决定。</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td>
<table>
<tr><td rowspan="2">值</td><td rowspan="2">Layer I & II</td><td colspan="2">Layer III</td></tr>
<tr><td>M/S stereo</td><td>Intensity stereo</td></tr>
<tr><td>00</td><td>bands 4 to 31</td><td>off</td><td>off</td></tr>
<tr><td>01</td><td>bands 8 to 31</td><td>off</td><td>on</td></tr>
<tr><td>10</td><td>bands 12 to 31</td><td>on</td><td>off</td></tr>
<tr><td>11</td><td>bands 16 to 31</td><td>on</td><td>on</td></tr>
</table>
</td></tr>
<tr><td>K</td><td>1</td><td>(3)</td><td>版权
0无版权
1有版权</td></tr>
<tr><td>L</td><td>1</td><td>(2)</td><td>原创
0 原创拷贝
1 原创</td></tr>
<tr><td>M</td><td>2</td><td>(1,0)</td><td>强调
00 - 无
01 - 50/15 ms
10 - 保留
11 - CCIT J.17</td></tr>
</table>

关于读取帧头我使用了下面的方法
定义一个结构体

```
typedef struct frameHeader
{
unsigned int sync1:8; //同步信息 1

unsigned int error_protection:1; //CRC 校验
unsigned int layer:2; //层
unsigned int version:2; //版本
unsigned int sync2:3; //同步信息 2

unsigned int extension:1; //版权
unsigned int padding:1; //填充空白字
unsigned int sample_rate_index:2; //采样率索引
unsigned int bit_rate_index:4; //位率索引

unsigned int emphasis:2; //强调方式
unsigned int original:1; //原始媒体
unsigned int copyright:1; //版权标志
unsigned int mode_extension:2; //扩展模式，仅用于联合立体声
unsigned int channel_mode:2; //声道模式

}FHEADER, *pFHEADER;
```

请注意我的同步信息分成了两个部分，而且其他的位的顺序也和上表列出的有所差别，这个主要是因为 c 语言在存取数据时总是从低位开始，而这个帧头是需要从高位来读取的。
读取方式如下
FHEADER header;
fread(&header, sizeof(FHEADER), 1, streams);//这里假设文件已打开，读取位置已经指向帧头所在的位置

这样一次就可以读入帧头的所有信息了。

2、如何计算帧长度

我们首先区分两个术语：帧大小和帧长度。帧大小即每帧采样数表示一帧中采样的个数，这是恒定值。其值入下表所示

	MPEG 1	MPEG 2 (LSF)	MPEG 2.5 (LSF)
Layer I	384	384	384
Layer II	1152	1152	1152
Layer III	1152	576	576

帧长度是压缩时每一帧的长度，包括帧头。它将填充的空位也计算在内。LayerI 的一个空位长 4 字节，LayerII 和 LayerIII 的空位是 1 字节。当读取 MPEG 文件时必须计算该值以便找到相邻的帧。
注意：因为有填充和比特率变换，帧长度可能变化。
从头中读取比特率，采样频率和填充，
LyaerI 使用公式：
帧长度（字节） = ((每帧采样数 / 8 * 比特率) / 采样频率) + 填充 * 4
LyerII 和 LyaerIII 使用公式：
帧长度（字节）= ((每帧采样数 / 8 * 比特率) / 采样频率) + 填充
例：
LayerIII 比特率 128000，采样频率 44100，填充 0
=〉帧大小 417 字节

3、每帧的持续时间

之前看了一些文章都说 mp3 的一帧的持续时间是 26ms，结果在实际程序的编写中发现无法正确按时间定位到帧，然后又查了一些文章才知道，所谓 26ms 一帧只是针对 MPEG1 Layer III 而且采样率为 44.1KHz 来说是对的，但 mp3 文件并不都是如此，其实这个时间也是可以通过计算来获得，下面给出计算公式

每帧持续时间(毫秒) = 每帧采样数 / 采样频率 * 1000

这样通过计算可知 MPEG1 Layer III 采样率为 44.1KHz 的一帧持续时间为 26.12...不是整数，不过我们权且认为它就是 26 毫秒吧。
如果是 MPEG2 Layer III 采样率为 16KHz 的话那一帧要持续 36 毫秒，这个相差还是蛮大的，所以还是应该通过计算来获的，当然可以按 MPEG 版本，层数和采样率来建一个表，这样直接查表就可以知道时间了。

4、CRC 校验

如果帧头的校验位为 0，则帧头后就有一个 16 位的 CRC 值，这个值是 big-endian 的值，把这个值和该帧通过计算得出的 CRC 值进行比较就可以得知该帧是否有效。
关于 CRC 校验下面给出我找到的英文原文，我的英文水平不高，翻译的不行。

If the protection bit in the header is not set, the frame contains a 16 bit CRC (Cyclic Redundancy Checksum). This checksum directly follows the frame header and is a big-endian WORD. To verify this checksum you have to calculate it for the frame and compare the calculated CRC with the stored CRC. If they aren't equal probably a transfer error has appeared. It is also helpful to check the CRC to verify that you really found the beginning of a frame, because the sync bits do in same cases also occur within the data section of a frame.

The CRC is calculated by applying the CRC-16 algorithm (with the generator polynom 0x8005) to a part of the frame. The following data is considered for the CRC: the last two bytes of the header and a number of bits from the audio data which follows the checksum after the header. The checksum itself must be skipped for CRC calculation. Unfortunately there is no easy way to compute the number of frames which are necessary for the checksum calculation in Layer II. Therefore I left it out in the code. You would need other information apart from the header to calculate the necessary bits. However it is possible to compute the number of protected bits in Layer I and Layer III only with the information from the header.

For Layer III, you consider the complete side information for the CRC calculation. The side information follows the header or the CRC in Layer III files. It contains information about the general decoding of the frame, but doesn't contain the actual encoded audio samples. The following table shows the size of the side information for all Layer III files.

	MPEG 1	MPEG 2/2.5 (LSF)
Stereo, Joint Stereo, Dual Channel	32	17
Mono	17	9

For Layer I files, you must consider the mode extension from the header. Then you can calculate the number of bits which are necessary for CRC calculation by applying the following formula:
4 * (number of channels * bound of intensity stereo + (32 - bound of intensity stereo));

This can be read as two times the number of stereo subbands plus the number of mono subbands and the result multiplied with 4. For simple mono frames, this equals 128, because the number of channels is one and the bound of intensity stereo is 32, meaning that there is no intensity stereo. For stereo frames this is 256. For more information have a look at the CRC code in the class CMPAFrame.

5、帧数据

在帧头后边是 Side Info(姑且称之为通道信息)。对标准的立体声 MP3 文件来说其长度为 32 字节。通道信息后面是 Scale factor(增益因子)信息。当解码器在读到上述信息后，就可以进行解码了。当 MP3 文件被打开后，播放器首先试图对帧进行同步，然后分别读取通道信息及增益因子等数据，再进行霍夫曼解码，至此我们已经获得解压后的数据。但这些数据仍然不能进行播放，它们还处于频域，要想听到歌曲还要将它由频域通过特定的手段转换到时域。接下来的处理分别为立体化处理；抗锯齿处理；IMDCT 变换；IDCT 变换及窗口化滑动处理。

我们知道，对于 mp3 来说现在有两种编码方式，一种是 CBR，也就是固定位率，固定位率的帧的大小在整个文件中都是是固定的（公式如上所述），只要知道文件总长度，和从第一帧帧头读出的信息，就都可以通过计算得出这个 mp3 文件的信息，比如总的帧数，总的播放时间等等，要定位到某一帧或某个时间点也很方便，这种编码方式不需要文件头，第一帧开始就是音频数据。另一种是 VBR，就是可变位率，VBR 是 XING 公司推出的算法，所以在 MP3 的 FRAME 里会有“Xing"这个关键字（也有用"Info"来标识的，现在很多流行的小软件也可以进行 VBR 压缩，它们是否遵守这个约定，那就不得而知了），它存放在 MP3 文件中的第一个有效帧的数据区里，它标识了这个 MP3 文件是 VBR 的。同时第一个帧里存放了 MP3 文件的帧的总个数，这就很容易获得了播放总时间，同时还有 100 个字节存放了播放总时间的 100 个时间分段的帧索引，假设 4 分钟的 MP3 歌曲，240S，分成 100 段，每两个相邻 INDEX 的时间差就是 2.4S，所以通过这个 INDEX，只要前后处理少数的 FRAME，就能快速找出我们需要快进的帧头。其实这第一帧就相当于文件头了。不过现在有些编码器在编码 CBR 文件时也像 VBR 那样将信息记入第一帧，比如著名的 lame，它使用"Info"来做 CBR 的标记。

6、VBR 头

这里列出 VBR 的第一帧存储文件信息的头的格式。有两种格式，一种是常见的 XING Header（头部包含字符‘Xing’），另一种是 VBRI Header（头部包含字符‘VBRI’）鉴于 VBRI Header 不常见，下面只说 XING Header，关于 VBRI Header 请看 http://www.codeproject.com/audio/MPEGAudioInfo.asp。

XING Header 的起始位置，相对于第一帧帧头的位置，单位是字节

36-39 "Xing" 文件为 MPEG1 并且不是单声道(大多数 VBR 的 mp3 文件都是如此)
21-24 "Xing" 文件为 MPEG1 并且是单声道
21-24 "Xing" 文件为 MPEG2 并且不是单声道
13-16 "Xing" 文件为 MPEG2 并且是单声道

XING Header 格式

位置（从'Xing'标记开始）	长度	含义	举例
0	4	VBR头标记，4个字节的ASCII字符，内容为 'Xing' 或者 'Info'	'Xing'
4	4	指示VBR头具体内容的标记，组合方式为逻辑或．区域是强制的． 0x0001 - 总帧数存储区域设置为存在，不包括第一帧 0x0002 - 文件长度存储区域设置为存在，不包括标签 0x0004 - TOC 索引存储区域设置为存在 0x0008 - 质量指示存储区域设置为存在	0x0007 (意味总帧数，文件长度，TOC的存储区有效)
8	4	存储总帧数的Big-Endian值	7344
8 or 12	4	存储文件长度Big-Endian值，单位为字节	45000
8, 12 or 16	100	100字节的 TOC 索引，用于快速定位 对于这个区域的存储内容，我认为可有可无，因为用1个字节来索引一个几兆文件的一帧是不可能做到准确定位的，就我所见基本上所有的VBR的mp3文件的TOC都几乎是相同的，就是把256平均分成100份然后填进去，其实和正确的值差不到哪里去，如果懒的话这么做也成吧，反正也是不准确的定位。 TOC索引的计算方式如下 (TOC[i] / 256) * 文件长度 比如文件持续240秒，我需要跳到60秒，文件长度为5000000字节 计算如下 TOC[(60/240)*100] = TOC[25] 然后相对于文件中的位置大约是在 (TOC[25]/256) * 5000000	
		如果要自己重建的话，基本是把这个步骤反过来做就可以了。要求准确的话，就需要根据时间点找到正确帧的位置然后再计算，我定位帧的做法都是从第一帧开始搜索，这样偏差我认为不会超过1帧，也比较准确，不过计算出来的TOC的值还是和偷懒的做法大同小异。	
8, 12, 16, 108, 112 or 116	4	质量指示器，为0(最好)-100(最差)的Big-Endian值	0

这样算来，XING Header 包括帧头一共最多只需要 156 个字节就够了。当然也可以在 XING Header 后面存储编码器的信息，比如 lame 在其后就是存储其版本，这需要给第一帧留足够的空间才行。

至于 mp3 的信息用从 XING Header 读出的信息就可以计算
比如
总持续时间 = 总帧数 * 每帧采样数 / 采样率 （结果为秒）
平均位率 = 文件长度 / 总持续时间 * 8

五、MPEG 音频标签

MPEG 音频标签分为两种，一种是 ID3v1，存在文件尾部，长度 128 字节，另一种是 ID3v2，是对 ID3v1 的扩展，存在文件头部，长度不定。

1、ID3v1

ID3v1 标签用来描述 MPEG 音频文件。包含艺术家，标题，唱片集，发布年代和流派。另外还有额外的注释空间。位于音频文件的最后固定为 128 字节。可以读取该文件的最后这 128 字节获得标签。

结构如下

AAABBBBB BBBBBBBB BBBBBBBB BBBBBBBB
BCCCCCCC CCCCCCCC CCCCCCCC CCCCCCCD
DDDDDDDD DDDDDDDD DDDDDDDD DDDDDEEE
EFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFG

符号	长度 (bytes)	位置 (bytes)	描述
A	3	(0-2)	标签标志。如果存在标签并且正确的话，必须包含'TAG'。
B	30	(3-32)	标题
C	30	(33-62)	艺术家
D	30	(63-92)	唱片集
E	4	(93-96)	年代
F	30	(97-126)	注释
G	1	(127)	流派

该规格要求所有的空间必须以空字符(ASCII 0)填充。但是并不是所有的应用程序遵循该规则，比如 winamp 就用空格(ASCII 32)代替之。
在 ID3v1.1 结构中有些改变。注释部分的最后一个字节用来定义唱片集中的轨道号。如果不知道该信息时可以用空字符(ASCII 0)代替。
流派使用原码表示，为下列数字之一：

0	'Blues'	20	'Alternative'	40	'AlternRock'	60	'Top 40'
1	'Classic Rock'	21	'Ska'	41	'Bass'	61	'Christian Rap'
2	'Country'	22	'Death Metal'	42	'Soul'	62	'Pop/Funk'
3	'Dance'	23	'Pranks'	43	'Punk'	63	'Jungle'
4	'Disco'	24	'Soundtrack'	44	'Space'	64	'Native American'
5	'Funk'	25	'Euro-Techno'	45	'Meditative'	65	'Cabaret'
6	'Grunge'	26	'Ambient'	46	'Instrumental Pop'	66	'New Wave'
7	'Hip-Hop'	27	'Trip-Hop'	47	'Instrumental Rock'	67	'Psychadelic'
8	'Jazz'	28	'Vocal'	48	'Ethnic'	68	'Rave'
9	'Metal'	29	'Jazz+Funk'	49	'Gothic'	69	'Showtunes'
10	'New Age'	30	'Fusion'	50	'Darkwave'	70	'Trailer'
11	'Oldies'	31	'Trance'	51	'Techno-Industrial'	71	'Lo-Fi'
12	'Other'	32	'Classical'	52	'Electronic'	72	'Tribal'
13	'Pop'	33	'Instrumental'	53	'Pop-Folk'	73	'Acid Punk'
14	'R&B'	34	'Acid'	54	'Eurodance'	74	'Acid Jazz'
15	'Rap'	35	'House'	55	'Dream'	75	'Polka'
16	'Reggae'	36	'Game'	56	'Southern Rock'	76	'Retro'
17	'Rock'	37	'Sound Clip'	57	'Comedy'	77	'Musical'
18	'Techno'	38	'Gospel'	58	'Cult'	78	'Rock & Roll'
19	'Industrial'	39	'Noise'	59	'Gangsta'	79	'Hard Rock'

Winamp 扩充了这个表

80	'Folk'	92	'Progressive Rock'	104	'Chamber Music'	116	'Ballad'
81	'Folk-Rock'	93	'Psychedelic Rock'	105	'Sonata'	117	'Poweer Ballad'
82	'National Folk'	94	'Symphonic Rock'	106	'Symphony'	118	'Rhytmic Soul'
83	'Swing'	95	'Slow Rock'	107	'Booty Brass'	119	'Freestyle'
84	'Fast Fusion'	96	'Big Band'	108	'Primus'	120	'Duet'
85	'Bebob'	97	'Chorus'	109	'Porn Groove'	121	'Punk Rock'
86	'Latin'	98	'Easy Listening'	110	'Satire'	122	'Drum Solo'
87	'Revival'	99	'Acoustic'	111	'Slow Jam'	123	'A Capela'
88	'Celtic'	100	'Humour'	112	'Club'	124	'Euro-House'
89	'Bluegrass'	101	'Speech'	113	'Tango'	125	'Dance Hall'
90	'Avantgarde'	102	'Chanson'	114	'Samba'		
91	'Gothic Rock'	103	'Opera'	115	'Folklore'		

其他扩充

126	'Goa'	132	'BritPop'	138	'BlackMetal'	144	'TrashMetal'
127	'Drum&Bass'	133	'Negerpunk'	139	'Crossover'	145	'Anime'
128	'Club-House'	134	'PolskPunk'	140	'ContemporaryChristian'	146	'JPop'
129	'Hardcore'	135	'Beat'	141	'ChristianRock'	147	'Synthpop'
130	'Terror'	136	'ChristianGangstaRap'	142	'Merengue'		
131	'Indie'	137	'HeavyMetal'	143	'Salsa'		

其他任何的数值都认为是“unknown”

2、ID3V2

ID3V2 到现在一共有 4 个版本，但流行的播放软件一般只支持第 3 版，既 ID3v2.3。由于 ID3V1 记录在 MP3 文件的末尾，ID3V2 就只好记录在 MP3 文件的首部了(如果有一天发布 ID3V3，真不知道该记录在哪里)。也正是由于这个原因，对 ID3V2 的操作比 ID3V1 要慢。而且 ID3V2 结构比 ID3V1 的结构要复杂得多，但比前者全面且可以伸缩和扩展。
下面就介绍一下 ID3V2.3。
每个 ID3V2.3 的标签都一个标签头和若干个标签帧或一个扩展标签头组成。关于曲目的信

息如标题、作者等都存放在不同的标签帧中，扩展标签头和标签帧并不是必要的，但每个标签至少要有一个标签帧。标签头和标签帧一起顺序存放在 MP3 文件的首部。

（一）、标签头

在文件的首部顺序记录 10 个字节的 ID3V2.3 的头部。数据结构如下：

```
char Header[3]; /*必须为"ID3"否则认为标签不存在*/
char Ver; /*版本号 ID3V2.3 就记录 3*/
char Revision; /*副版本号此版本记录为 0*/
char Flag; /*存放标志的字节，这个版本只定义了三位，稍后详细解说*/
char Size[4]; /*标签大小，包括标签头的 10 个字节和所有的标签帧的大小*/
```

注:对这里我有疑惑，因为在实际寻找首帧的过程中，我发现有的 mp3 文件的标签大小是不包含标签头的，但有的又是包含的，可能是某些 mp3 编码器写标签的 BUG，所以为了兼容只好认为其是包含的，如果按大小找不到，再向后搜索，直到找到首帧为止。

（1）.标志字节

标志字节一般为 0，定义如下：

abc00000

a -- 表示是否使用 Unsynchronisation(这个单词不知道是什么意思，字典里也没有找到，一般不设置)

b -- 表示是否有扩展头部，一般没有(至少 Winamp 没有记录)，所以一般也不设置

c -- 表示是否为测试标签(99.99%的标签都不是测试用的啦，所以一般也不设置)

（2）.标签大小

一共四个字节，但每个字节只用 7 位，最高位不使用恒为 0。所以格式如下

0xxxxxxx 0xxxxxxx 0xxxxxxx 0xxxxxxx

计算大小时要将 0 去掉，得到一个 28 位的二进制数，就是标签大小(不懂为什么要这样做)，计算公式如

下：

```
int total_size;
total_size = (Size[0]&0x7F)*0x200000
+(Size[1]&0x7F)*0x4000
+(Size[2]&0x7F)*0x80
+(Size[3]&0x7F)
```

（二）、标签帧

每个标签帧都有一个 10 个字节的帧头和至少一个字节的不固定长度的内容组成。它们也是顺序存放在文件

中，和标签头和其他的标签帧也没有特殊的字符分隔。得到一个完整的帧的内容只有从帧头中的到内容大

小后才能读出，读取时要注意大小，不要将其他帧的内容或帧头读入。

帧头的定义如下：

```
char FrameID[4]; /*用四个字符标识一个帧，说明其内容，稍后有常用的标识对照表*/
```

char Size[4]; /*帧内容的大小，不包括帧头，不得小于 1*/
char Flags[2]; /*存放标志，只定义了 6 位，稍后详细解说*/

（1）.帧标识

用四个字符标识一个帧，说明一个帧的内容含义，常用的对照如下：
TIT2=标题 表示内容为这首歌的标题，下同
TPE1=作者
TALB=专集
TRCK=音轨 格式：N/M 其中 N 为专集中的第 N 首，M 为专集中共 M 首，N 和 M 为 ASCII 码表示的数字
TYER=年代 是用 ASCII 码表示的数字
TCON=类型 直接用字符串表示
COMM=备注 格式："eng\0 备注内容"，其中 eng 表示备注所使用的自然语言

（2）.大小

这个可没有标签头的算法那么麻烦，每个字节的 8 位全用，格式如下
xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
算法如下：
int FSize;
FSize = Size[0]*0x1000000
+Size[1]*0x10000
+Size[2]*0x100
+Size[3];

（3）.标志

只定义了 6 位，另外的 10 位为 0，但大部分的情况下 16 位都为 0 就可以了。格式如下：
abc00000 ijk00000
a -- 标签保护标志，设置时认为此帧作废
b -- 文件保护标志，设置时认为此帧作废
c -- 只读标志，设置时认为此帧不能修改(但我没有找到一个软件理会这个标志)
i -- 压缩标志，设置时一个字节存放两个 BCD 码表示数字
j -- 加密标志(没有见过哪个 MP3 文件的标签用了加密)
k -- 组标志，设置时说明此帧和其他的某帧是一组
值得一提的是 winamp 在保存和读取帧内容的时候会在内容前面加个'\0'，并把这个字节计算在帧内容的
大小中。

附：帧标识的含义
（4）. Declared ID3v2 frames

The following frames are declared in this draft.

AENC Audio encryption
APIC Attached picture
COMM Comments
COMR Commercial frame
ENCR Encryption method registration
EQUA Equalization
ETCO Event timing codes
GEOB General encapsulated object
GRID Group identification registration
IPLS Involved people list
LINK Linked information
MCDI Music CD identifier
MLLT MPEG location lookup table
OWNE Ownership frame
PRIV Private frame
PCNT Play counter
POPM Popularimeter
POSS Position synchronisation frame
RBUF Recommended buffer size
RVAD Relative volume adjustment
RVRB Reverb
SYLT Synchronized lyric/text
SYTC Synchronized tempo codes
TALB Album/Movie/Show title
TBPM BPM (beats per minute)
TCOM Composer
TCON Content type
TCOP Copyright message
TDAT Date
TDLY Playlist delay
TENC Encoded by
TEXT Lyricist/Text writer
TFLT File type
TIME Time
TIT1 Content group description
TIT2 Title/songname/content description
TIT3 Subtitle/Description refinement
TKEY Initial key
TLAN Language(s)
TLEN Length
TMED Media type
TOAL Original album/movie/show title
TOFN Original filename
TOLY Original lyricist(s)/text writer(s)

TOPE Original artist(s)/performer(s)
TORY Original release year
TOWN File owner/licensee
TPE1 Lead performer(s)/Soloist(s)
TPE2 Band/orchestra/accompaniment
TPE3 Conductor/performer refinement
TPE4 Interpreted, remixed, or otherwise modified by
TPOS Part of a set
TPUB Publisher
TRCK Track number/Position in set
TRDA Recording dates
TRSN Internet radio station name
TRSO Internet radio station owner
TSIZ Size
TSRC ISRC (international standard recording code)
TSSE Software/Hardware and settings used for encoding
TYER Year
TXXX User defined text information frame
UFID Unique file identifier
USER Terms of use
USLT Unsychronized lyric/text transcription
WCOM Commercial information
WCOP Copyright/Legal information
WOAF Official audio file webpage
WOAR Official artist/performer webpage
WOAS Official audio source webpage
WORS Official internet radio station homepage
WPAY Payment
WPUB Publishers official webpage
WXXX User defined URL link frame

五、MP3 文件实例剖析

在 VC++中打开一个名为 test.mp3 文件，其内容如下：

000000 FF FB 52 8C 00 00 01 49 09 C5 05 24 60 00 2A C1
000010 19 40 A6 00 00 05 96 41 34 18 20 80 08 26 48 29
000020 83 04 00 01 61 41 40 50 10 04 00 C1 21 41 50 64
……
0000D0 FE FF FB 52 8C 11 80 01 EE 90 65 6E 08 20 02 30
0000E0 32 0C CD C0 04 00 46 16 41 89 B8 01 00 08 36 48
0000F033 B7 00 00 01 02 FF FF FF F4 E1 2F FF FF FF FF
……
0001A0 DF FF FF FB 52 8C 12 00 01 FE 90 58 6E 09 A0 02
0001B0 33 B0 CA 85 E1 50 01 45 F6 19 61 BC 26 80 28 7C
0001C0 05 AC B4 20 28 94 FF FF FF FF FF FF FF FF FF FF

……

001390 7F FF FF FF FD 4E 00 54 41 47 54 45 53 54 00 00

0013A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

……

0013F000 00 00 00 04 19 14 03 00 00 00 00 00 00 00 00

001400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

001410 00 00 00 00 00 00 4E

该文件长度 1416H（5.142K），帧头为：FF FB 52 8C，转换成二进制为：

11111111 11111011

01010010 10001100

对照表 1 可知，test.mp3 帧头信息见表 5。

表 5 test.mp3 文件帧头信息

名称	位值	说　明
同步信息	11111111111	第1字节恒为FF，11位均为1。
版本	11	MPEG 1
层	01	Layer 3
CRC校验	1	不校验
位率	0101	64kbps
频率	00	44.1kHz
帧长调节	1	调整，帧长是210字节。
保留字	0	没有使用。
声道模式	10	双声道
扩充模式	00	未使用。
版权	1	合法
原版标志	1	原版
强调方式	00	未定义

第 1397H 开始的三个字节是 54 41 47，存放的是字符“TAG”，表示此文件有 ID3 V1.0 信息。

139AH 开始的 30 个字节存放歌名，前 4 个非 00 字节是 54 45 53 54，表示“TEST”；

13F4H 开始的 4 个字节是 04 19 14 03，存放年份“04/25/2003”；

最后 1 个字节是 4E，表示音乐类别，代号为 78，即“Rock&Roll”；

其它字节均为 00，未存储信息。

六、资料

www.id3.org

部分参考文章的网址

http://mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm

http://www.codeproject.com/audio/MPEGAudioInfo.asp

http://le-hacker.org/hacks/mpeg-drafts/11172-3.pdf (ISO/IEC 11172-3 我想这个应该有很多人找吧，不过这里面定义的帧同步位为 12 位，因为是老标准)

http://webstore.iec.ch/preview/info_isoiec13818-3%7Bed2.0%7Den.pdf（ISO/IEC 13818-3 网站似乎是收费的，不过直接可下，应该不会有人找我麻烦吧）