MP3 文件格式详解

1、MP3 简介

MP3 是 MPEG-1 Audio Layer 3 的简称，是当今比较流行的一种数字音频编码和有损压缩格式（有 Layer 3，也必然有 Layer1 和 Layer2，也就是 MP1 和 MP2，但不在本文讨论范围之内）。MP3 技术的应该可以用来大幅度的降低音频文件存储所需要的空间。它丢掉脉冲编码调制（PCM）音频数据中对人类听觉不重要得数据，从而达到了较高的压缩比（高达 12：1－10：1）。简单地说，MP3 在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，最后形成有较高压缩比的 MP3 文件，并使压缩后的文件在回放时也能够达到比较接近原音源的效果。

MP3 的音频质量取决于它的 Bitrate 和 Sampling frequency，以及编码器质量。MP3 的典型速度介于每秒 128 到 320kb 之间。采样频率也有 44.1，48 和 32 kHz 三种频率，比较常见的是采用 CD 采样频率——44.1kHz。常用的编码器是 LAME，它完全遵循 LGPL 的 MP3 编码器，有着良好的速度和音质。

2、MP3 文件格式

用一个二进制查看器（比如 Ultra-Edit）打开一个 MP3 文件，就能看到一大堆看似杂乱无序的数据。但只要用心了解就会知道，其实，这一切都是有规律可循的。

MP3 文件是由帧（frame）构成，帧是 MP3 文件的最小组成单位。每帧都包含帧头，并可以计算帧的长度。根据帧的性质不同，文件主要分为三个部分，ID3v2 标签帧，数据帧和 ID3v1 标签帧。并非每个 MP3 文件都有 ID3v2，但是数据帧和 ID3v1 帧是必须的。ID3v2 在文件头，以字符串“ID3”为标志，包含了演唱者，作曲，专辑等信息，长度不固定，扩展了 ID3V1 的信息量。ID3v1 在文件结尾，以字符串“TAG”为标记，其长度是固定的 128 个字节，包含了演唱者、歌名、专辑、年份等信息。

I、ID3V2

ID3V2 到现在一共有四个版本，但流行的播放软件一般只支持第三版，既 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 ID3V2.3 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。

标签头

长度为 10 个字节，位于文件首部，其数据结构如下：

```
char Header[3]; /* 字符串 "ID3" */
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，很少用到，可以忽略 */
char Size[4]; /* 标签大小，除了标签头的 10 个字节的标签帧的大小 */
```

标签大小为四个字节，但每个字节只用低 7 位，最高位不使用，恒为 0，其格式如下：
0xxxxxxx 0xxxxxxx 0xxxxxxx 0xxxxxxx

计算公式如下：

```
ID3V2_frame_size = (int)(Size[0] & 0x7F) << 21
                                | (int)(Size[1] & 0x7F) << 14
                                  | (int)(Size[2] & 0x7F) << 7
                                  | (int)(Size[3] & 0x7F) + 10;
```

标签帧

每个标签帧都有一个 10 字节的帧头和至少一个字节的不固定长度的内容组成。它们是顺序存放在文件中，由各自特定的标签头来标记帧的开始。其帧的结构如下：

```
char FrameID[4];     /*用四个字符标识一个帧，说明其内容 */

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，此处不再说明 */
```

常用帧标识：

TIT2：标题
TPE1：作者
TALB：专辑
TRCK： 音轨，格式：N/M，N 表示专辑中第几首，M 为专辑中歌曲总数
TYER：年份
TCON：类型
COMM：备注，格式：“eng\0 备注内容”，其中 eng 表示所使用的语言
帧大小为四个字节所表示的整数大小。

II、ID3V1

其数据结构如下：

```
char Header[3];       /* 标签头必须是"TAG"否则认为没有标签 */
char Title[30];       /* 标题 */
char Artist[30];     /* 作者 */
char Album[30];       /* 专集 */
char Year[4];           /* 出品年代 */
char Comment[28]; /* 备注 */
char reserve;           /* 保留 */
char track;;             /* 音轨 */
char Genre;               /* 类型 */
```

其实，关于最后 31 个字节还存在另外一个版本，就是 30 个字节的 Comment 和一个字节的 Genre.

有了上述的这些信息，我们就可以自己写代码，从 MP3 文件中抓取信息以及修改文件名了。但是，如果真的想写一个播放软件，还是需要读它的数据帧，并进行解码。

III、数据帧

数据帧往往有多个，至于有多少，由文件大小和帧大小来决定。每个帧都有一个四字节长的帧头，接下来可能有两个字节的 CRC 校验，其存在由帧头中的具体信息决定。接着就是帧的实体数据，也就是 MAIN_DATA 了。

A,帧头结构如下：

位置（BIT）	长度（BITS）	描述
31－19	12	Frame sync(0xFFF)
18/17	2	Layer, 00 - reserved, 01 - Layer III 10 - Layer II, 11 － Layer I
16	1	protection_bit, 0 意味着受 CRC 保护，帧头后面跟 16 位的 CRC。
15-12	4	bitrate_index, 比特率
11-10	2	sampling_frequency, 00 - 44.1KHz, 01 - 48KHz 10 - 32 KHz, 11 - 保留
9	1	padding_bit,1 意味着帧里包含 padding 位，仅当采样频率为 44.1KHz 时发生。
8	1	private_bit
7－6	2	mode, 00－stereo, 01-joint stereo(intensity stereo and/or ms_stereo) 11- dual_channel, 11 - single_channel
5-4	2	mode_extension,在 Layer III 中表示使用了哪一种 joint stereo 编码方式。 Intensity_stereo ms_stereo 00 off off 01 on off 10 off on 11 on on
3	1	copyright,1 表示受版权保护。
2	1	original，0 表示该 bitstream 是一个 copy，1 表示是 original.
1-0	2	emphasis，表示会使用哪一种 de-emphasis。 00 － no emphasis, 01 - 50/15 microsec. Emphasis 10 - reserved, 11 - CCITT J.17

1) 无论帧长是多少，每帧的播放时间都是 26ms

2) 数据帧大小：

FrameSize = 144 * Bitrate / SamplingRate + PaddingBit
当 144 * Bitrate / SamplingRate 不能被 8 整除，则加上相应的 paddingBit.

B,MAIN_DATA:

MP3 的 granule 包含 18 * 32 个 subband 采样。每个数据帧含有两个 granule 的数据，其内容结如下：

- main_data_end pointer
- side info for both granules (scfsi)
- side info granule 1
- side info granule 2

- scalefactors and Huffman code data granule 1
- scalefactors and Huffman code data granule 2

主要数据里包含了 scalefactors, Huffman encoded data 和 ancillary information。其内容不再详叙，可以参考 MP3 SPEC－ISO 11172-3 AUDIO PART。我们一般用的都是立体声，scfsi 的长度为 32 个字节。

这里要解释的一个概念就是位流－－bitstream。我们平常接触到的数据都是整数，最小的单位就是 byte 后者 char。虽然我们也会用一个字节里的不同位来表示不同的含义，但总的来说，我们在出来数据的时候还是把它当作一个个字节看待。但对 MP3 这种数据格式来说，这是行不通的。在解码时，它的数据输入就是一个个比特流。其中一个或几个比特会是你的采样数据或者信息编码。你需要从整个 MAIN_DATA 里提取你所需要的以 BIT 为单位的参数和输入信号，从而进行解码。所以我们需要一个子程序，getbit(n)，也就是从缓冲中提取所需要的位，并形成一个新的整数，作为我们的输出。

C,LAME 标签帧

可是，当你真的打开一个 MP3 文件的时候，你会发现，很奇怪，很多时候第一个数据帧的帧头后面的 32 个字节居然都为 0，这是为什么呢，这么奇怪的解码信息该如何解释？找到 MP3 INFO TAG REV SPECIFICATION 的网站，我才明白，原来第一帧并不是真正的数据帧，而是 LAME 编码的标志帧。

这里又要牵涉到两个概念：CBR 和 VBR。CBR 表示比特率不变，也就是每帧的长度是一致的，它以字符串“INFO”为标记。VBR 是 Variable BitRate 的简称，也就是每帧的比特率和帧的长度是变化的，它以字符串“Xing”为标记。同时，它还存放了 MP3 文件里帧的总个数，和 100 个字节的播放总时间分段的帧的 INDEX，还有其他一些参数，这被称为 Zone A，传统 Xing VBR 标签数据，共 120 个字节。

在二进制文本编辑器里我们还可看到一个字符串“LAME”，并且后面清楚地跟着版本号。这就是 20 个字节的 Zone B 初始 LAME 信息，表示该文件是用 LAME 编码技术。接下来一直到该帧结束就是 Zone C－LAME 标签。

3、案例

为了获取 MP3，写了一个爬虫程序，专门从一家音乐网站上搜索下载 mp3,一下子下载了有上千首。这这些 MP3 的歌曲名字都是使用 1，2，3，4，。。等数字命名，挑选起来十分不方便。虽然 MP3 播放器能够读出 MP3 文件信息的歌曲名，但歌曲文件本身的名字却不利于自己管理。于是就想写一个小程序实现 MP3 自动更名。查了一些资料，研究了一下 MP3 的文件结构。

研究 MP3 的结构，就不能不研究 ID3 标签。ID3 标签是 MP3 音乐档案中的歌曲附加讯息，它能够在 MP3 中附加曲子的演出者、作者以及其它类别资讯，方便众多乐曲的管理。缺少 ID3 标签并不会影响 MP3 的播放，但若没有的话，管理音乐文件也会相当的麻烦。如果你在网上 download MP3，里面多半已经写有预设的 ID3 讯息。ID3，一般是位于一个 mp3 文件的开头或末尾的若干字节内，附加了关于该 mp3 的歌手，标题，专辑名称，年代，风格等信息，该信息就被称为 ID3 信息，ID3 信息分为两个版本，v1 和 v2 版。

其中：v1 版的 ID3 在 mp3 文件的末尾 128 字节，以 TAG 三个字符开头，后面跟上歌曲信息。

v2 版一般位于 mp3 的开头，可以存储歌词，该专辑的图片等大容量的信息。

但 ID3 并不是 MP3 标签的 ISO 国际标准，ID3 的各种版本目前只是一个近乎事实上的标准，并没有人强迫播放器或者编码程序必须支持它。

ID3V1 大概有两个版本，由于 ID3V1.0 没有包括曲目序号的定义，所以 Michael Mutschler 在 1997 年进行了改进，引入了版本 1.1。通过占用备注字段的最后两个字节，用一个 00 字节作标记，另一个字节改为序号，可以让 ID3 支持曲目编号了。一个字节的空间让 ID3 V1.1 支持最高到 255 的曲目序号，考虑到一张唱片超过 256 个曲目的可能性极小，这个改进还是相当合理的。但 ID3V1 只有 128 个字节可以使用，如果要在 MP3 中储存更多的信息，比如歌词，专辑图片等，显然是无法达到的，于是产生了 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。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。

但我们只需要读出 MP3 的 TITLE，所以只要解析 IDV1 就够了，这里不对 IDV2 做过多说明，其实我也没有深入研究 IDV2。

ID3V1 的内容和每个标签占用的字节说明如下：

```
char Header[3];     /*标签头必须是"TAG"否则认为没有标签*/
char Title[30];     /*标题*/
char Artist[30];    /*作者*/
char Album[30];     /*专集*/
char Year[4];     /*出品年代*/
char Comment[30];    /*备注*/
char Genre;     /*类型*/
```

可以定义一个如下的结构来存储 MP3 信息：

```
typedef struct _MP3INFO //MP3 信息的结构
{
 char Identify[3]; //TAG 三个字母
 //这里可以用来鉴别是不是文件信息内容
 char Title[31];   //歌曲名，30 个字节
 char Artist[31];  //歌手名，30 个字节
 char Album[31];   //所属唱片，30 个字节
 char Year[5];   //年，4 个字节
 char Comment[29]; //注释，28 个字节
 unsigned char reserved;  //保留位，1 个字节
 unsigned char reserved2; //保留位，1 个字节
 unsigned char reserved3; //保留位，1 个字节
} MP3INFO;
```

代码可以简单如下：

```
 #include "stdlib.h"
#include "stdio.h"
#include "windows.h"
#define MAX 128
typedef struct _MP3INFO //MP3 信息的结构
{
 char Identify[3]; //TAG 三个字母
 //这里可以用来鉴别是不是文件信息内容
 char Title[31];   //歌曲名，30 个字节
 char Artist[31];  //歌手名，30 个字节
 char Album[31];   //所属唱片，30 个字节
 char Year[5];   //年，4 个字节
 char Comment[29]; //注释，28 个字节
 unsigned char reserved;  //保留位，1 个字节
 unsigned char reserved2; //保留位，1 个字节
 unsigned char reserved3; //保留位，1 个字节
} MP3INFO;
int main(int argc, char* argv[])
{
 FILE * fp;
 unsigned char mp3tag[128] = {0};
    MP3INFO mp3info;
```

```
 char oldname[MAX],newname[MAX],cmd[MAX];

 fp = fopen("G://mp3//Debug//5.mp3","rb");
 if (NULL==fp)
 {
  printf("open read file error!!");
  return 1;
 }
 fseek(fp,-128,SEEK_END);
 fread(&mp3tag,1,128,fp);
    if(!((mp3tag[0] == 'T'|| mp3tag[0] == 't')
  &&(mp3tag[1] == 'A'|| mp3tag[1] == 'a')
  &&(mp3tag[2] == 'G'|| mp3tag[0] == 'g')))
 {
     printf("mp3 file is error!!");
  fclose(fp) ;
  return 1;
 }
 memcpy((void *)mp3info.Identify,mp3tag,3); //获得 tag
 memcpy((void *)mp3info.Title,mp3tag+3,30); //获得歌名
 memcpy((void *)mp3info.Artist,mp3tag+33,30); //获得作者
 memcpy((void *)mp3info.Album,mp3tag+63,30); //获得唱片名
 memcpy((void *)mp3info.Year,mp3tag+93,4); //获得年
 memcpy((void *)mp3info.Comment,mp3tag+97,28); //获得注释
 memcpy((void *)&mp3info.reserved,mp3tag+125,1); //获得保留
 memcpy((void *)&mp3info.reserved2,mp3tag+126,1);
 memcpy((void *)&mp3info.reserved3,mp3tag+127,1);
 fclose(fp);
 if (strlen(mp3info.Title) == 0)
 {
  printf("title is null/n");
  return 1;
 }
    strcpy(oldname,"5.mp3");
 sprintf(newname,"%s.mp3",mp3info.Title);
 sprintf(cmd,"rename G://mp3//Debug//%s %s",oldname,newname);
 printf("%s/n", cmd);
 system(cmd);

 return 0;
}
```