ISSN 1009-3044
Computer Knowledge and Technology 电脑知识与技术
Vol.12, No.27, September 2016
E-mail:kfyj@dnzs.net.cn
http://www.dnzs.net.cn
Tel:+86-551-65690963 65690964

wav音频文件格式分析与数据获取

罗海涛

(广东外语外贸大学 思科信息学院,广东 广州 510420)

摘要:音频文件是把语音信号离散化的数字文件,wav格式的音频文件是常用的二进制音频格式,广泛应用于语音信号处理、语音识别、语音合成等领域;本文详细分析了wav音频文件格式,并用C语言编程,实现对该格式文件的访问,获取音频信息和数据。

关键字:wav;音频文件;音频信息;音频数据

中图分类号:TP37 文献标识码:A 文章编号:1009-3044(2016)27-0211-03

DOI:10.14004/j.cnki.ckt.2016.3935

1 概述

语言是人们之间进行通讯和交流必不可少的手段。语音由人的发音器官发出,语音信号是连续的模拟信号,在用计算机来处理时,需要进行数字化,包括采样、量化等过程,转换成离散的数字信号,保存在音频文件中。现在很多领域要求对语音信号中的音频数据进行进一步的加工和处理。例如,利用读出的音频信号数据,进行语音信号时域和频域分析、语音压缩、语音编码、解码、语音合成、语音识别、语音增强等,并通过波形观察比较不同编码效果。另外,利用多媒体语音系统我们还可以用语音数据和波形方便地进行噪声模拟分析,语音特征提取研究,以及语音识别和训练等应用方面的实验。又如:在人工智能领域,通过设计软件和硬件电路,用声音去控制计算机工作,还有机器人通过语音与人进行简单的对话交流等等。这些都要求我们对数字语音信号进行一些必要的加工处理。

wav文件格式是一种重要的数字音频文件格式,是目前应用很广泛的一种音频格式。相比于其他格式如MP3、MP4、RAM等压缩效率更高的音频文件格式,wav文件没有采用压缩技术,因而其文件要大很多,一般都在几兆字节,甚至更大。但也正因为没有采用压缩技术,wav文件中声音的采样数据很容易被读出来,便于做其他处理。例如:画出声音的信号波形、作出频谱,进行时域、频域分析,提取语音信号的特征参数用于语音识别等。现在的应用程序几乎都支持wav文件格式,也有专门软件可以完成从wav文件格式向其他文件格式的转换,或者把其他格式文件转换为wav格式,例如,微软公司的Adobe Audition。因此wav文件在目前仍然有着广泛的应用价值,有很多应用程序仍然采用wav文件格式。

本文在Visual C++环境下编程实现了wav音频文件的读取,读出其中的参数和音频数据,以便进一步用于特征参数提取、说话人识别等,并对TIMIT语音库的语音文件进行了读取试验验证。

2 wav文件格式分析

2.1 wav文件格式

由于wav格式的波形文件是二进制文件,用C语言编程对该文件读取其中的数据,需要先了解它的格式。

wav格式是微软公司开发的一种声音文件格式,也叫波形文件,是最早的数字音频文件格式,它具有RIFF(Resource Interchange File Format)格式。RIFF格式的wav文件由若干个Chunk(块)组成,按顺序为RIFF WAVE Chunk、Format Chunk、Fact Chunk(可选)和Data Chunk。每个块都有固定而且类似的格式,一般第1部分是块的ID,作为标识,4个字节大小,紧跟其后的是该块的大小,也是用4个字节表示,低字节表示低位,高字节表示高位;第3部分略有差异,以下分别详细说明。

RIFF WAVE块格式如表1所示:

表1 RIFF WAVE块格式

名称	长度(字节数)	内容
ID	4	"RIFF"
Size	4	
Type	4	"WAVE"

Wav文件最开始4个字节是ID部分,其内容为RIFF的ASCII码,紧跟的4个字节是文件大小(字节数)减去ID和Size所占字节数,共8个字节,即文件大小(字节数)减去8。然后是Type部分,其内容为WAVE的ASCII码,4个字节。

Format块要复杂得多,其格式如表2所示:

表2 Format块格式

名称	长度(字节数)	内容
ID	4	"fmt "
Size	4	16或18,18则最后有附加信息
FormatTag	2	编码方式,一般0x0001
Channels	2	声道数,1为单声道,2为双声道
SamplesPerSec	4	采样频率
AvgBytesPerSec	4	每秒所需字节数
BolckAlign	2	每个采样数据需要的字节数
BitsPerSample	2	每个采样数据需要的二进制位数
(附加,可选)	2	附加信息,由Size决定有无

其中ID部分同样占4个字节,其内容为"fmt "(注意最后有

收稿日期:2016-08-25

一个空格)的ASCII码。

fact块为可选的块,有的波形文件有这个块,有的没有,其格式如表3所示:

表3　fact块格式

名称	长度(字节数)	内容
ID	4	"fact"
Size	4	4
data	4	"WAVE"

有些wav文件是由某些软件转化来的,一般就包含该块。其ID部分为fact的ASCII码,紧跟的4个字节是存储的是4,然后是data部分,其内容为WAVE的ASCII码,4个字节。

最后一个块是Data块,其格式如表4所示:

表4　data块格式

名称	长度(字节数)	内容
ID	4	"data"
Size	4	
data		(具体数据)

其ID部分为data的ASCII码,紧跟的4个字节是音频数据个数,然后是data部分,存储的是具体的音频数据。

2.2 wav文件示例

我用debug调试工具把一个wav二进制文件调入内存,再用该工具的显示命令d,把其中部分内容显示出来,如图1所示。

```
-d
1396:0100  52 49 46 46 06 C7 03 00-57 41 56 45 66 6D 74 20   RIFF....WAVEfmt
1396:0110  10 00 00 00 01 00 01 00-44 AC 00 00 88 58 01 00   ........D....X..
1396:0120  02 00 10 00 64 61 74 61-E2 C6 03 00 00 00 00 00   ....data........
1396:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1396:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1396:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1396:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1396:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

图1　一个wav文件开头部分内容

图中,第1行的d是显示命令,第2行的"1396:0100"是内存地址,4个十六进制数据"52 49 46 46"依次是"RIFF"4个字母的ASCII码;紧跟的4个十六进制数据"06 C7 03 00"实际上是一个数值,低位在前,高位在后,因此是十六进制数0x3C706,即十进制数247558,该值等于整个wav文件大小减去8;再后面的4个十六进制数据"57 41 56 45"依次是"WAVE"4个字母的ASCII码;至此是第一个块,即RIFF WAVE块。

之后4个十六进制数据"66 6D 74 20"依次是"fmt "3个字母加最后一个空格的ASCII码,这表示Format块开始;之后(第3行1396:0110)4个十六进制数据"10 00 00 00"实际上是一个数值,低位在前,高位在后,因此是十六进制数0x10,即十进制数16,表示该块的"Size"分布,由表2得知,该块最后没有附加信息;之后2个十六进制数据"01 00"实际上是一个数值,低位在前,高位在后,因此是十六进制数0x1,即十进制数1,根据表2,这是编码方式;之后再2个十六进制数据"01 00"同样是一个数值,低位在前,高位在后,因此是十六进制数0x1,即十进制数1,根据表2,这是声道数目,1表示单声道;之后4个十六进制数据"44 AC 00 00"是一个数值,低位在前,高位在后,因此是十六进制数0xAC44,即十进制数44100,根据表2,这是采样频率,即每秒钟采集的样本个数,单位为Hz;之后4个十六进制数据"88 58 01 00"是一个数值,低位在前,高位在后,因此是十六进制数0x15888,即十进制数88200,根据表2,这是每秒所需字节数;前面采样频率为44100,即每秒有44100个样本数据,此处每秒需88200字节,显然,每个采样数据用2个字节来表示;之后2个十六进制数据"02 00"(第4行1396:0120)是一个数值,低位在前,高位在后,因此是十六进制数0x2,即十进制数2,根据表2,这是每个采样数据需要的字节数,这和前面的分析结论一致;之后2个十六进制数据"10 00"是一个数值,低位在前,高位在后,因此是十六进制数0x10,即十进制数16,根据表2,这是每个采样数据需要的二进制位数,16位即2个字节;Format块到此结束。

之后的4个十六进制数据"64 61 74 61"(第4行1396:0120)依次是"data"4个字母的ASCII码,此处是data块开始;之后4个十六进制数据"E2 C6 03 00"(第4行1396:0120)是一个数值,低位在前,高位在后,因此是十六进制数0x3C6E2,即十进制数247522,根据表4,这是采样数据大小,以字节个数为单位;再后面就是具体的采样数据,每个数据用2个字节表示,低位在前,高位在后。这个例子的wav文件没有fact块。

从图1中看出,前面的采样数据全为0,这是因为wav音频文件一开始是静音,没有声音。在进一步用debug工具的d命令继续往下显示,如图2所示。

```
-d
1396:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1396:0190  00 00 00 00 00 00 00 00-00 00 01 00 01 00 00 00   ................
1396:01A0  00 00 00 00 FF FF FF FF-FF FF 00 00 00 00 00 00   ................
1396:01B0  01 00 01 00 01 00 00 00-00 00 FF FF FF FF FF FF   ................
1396:01C0  FF FF FF FF 00 00 01 00-01 00 02 00 01 00 01 00   ................
1396:01D0  00 00 FF FF FE FF FE FF-FE FF FF FF 00 00 02 00   ................
1396:01E0  02 00 02 00 02 00 01 00-FF FF FE FF FD FF FD FF   ................
1396:01F0  FE FF FF FF 01 00 03 00-04 00 04 00 03 00 01 00   ................
```

图2　一个wav文件部分内容

从图2可以看出,音频文件有声音的部分,其采样数据就不是0。

3 编程实现

根据前面的分析和示例,我在Visual C++环境下用C语言编程实现了对wav音频文件的音频数据读取。

由于wav二进制文件结构较复杂,先定义一个头文件"wav.h",在头文件除了给出定义:

```
#ifndef _WAV_H_
#define _WAV_H_
```

更重要的是针对前面分析的四种块,分别定义四种结构体,第1个结构体对应RIFF WAVE块:

```
struct RIFF_HEADER
{unsigned char szRiffID[4];  //'R','I','F','F'
unsigned int dwRiffSize;  //4个字节显示RIFF块大小,
        //其值等于WAV文件字节数-4(RIFFID字节数)-4(此处字节数)
unsigned char szRiffFormat[4]; }; //'W','A','V','E'
```

考虑到Format块较复杂,定义了两个结构体来表示它:

```
struct FORMAT_Chunk
{unsigned short int wFormatTag;  //2字节显示编码方式,一般为0x0001
unsigned short int wChannels;  //2字节显示声道数目,1为单声道,2为双声道
```

unsigned int dwSamplesPerSec; //4字节显示采样频率

unsigned int dwAvgBytesPerSec; //4字节显示每秒所需字节数

unsigned short int wBlockAlign; //2字节显示数据块对齐方式(每个采样需要的字节数)

unsigned short int wBitsPerSample; //2字节显示每个采样需要的bit数

unsigned short int append_info; }; //附加信息

以及：

struct FMT_Chunk

{unsigned char szFmtID[4]; //'f','m','t',' '

unsigned int dwFmtSize;//4字节，=16，=18有附加信息在FORMAT_Chunk的append_info

struct FORMAT_Chunk wavFormat;};

第4个结构体对应fact块：

struct FACT_Chunk

{unsigned char szFactID[4]; //'f','a','c','t'

unsigned int dwFactSize; }; //4字节显示大小(数值为4)

第5个结构体对应data块：

struct DATA_Chunk

{unsigned char szDataID[4]; //'d','a','t','a'

unsigned int dwDataSize; }; //4字节显示音频数据块大小(字节数)，其后是音频数据。

在头文件定义了这些结构体后，就可以在C++文件(.cpp文件)中编程，读取wav文件中的音频数据。可以先定义一个文件指针：

FILE *wav_in;

以指向并打开音频文件；再分别定义四种结构体变量，文件位置指针等；再用fopen函数打开音频文件，用fread函数依次读取文件中的各个量，以及后面的音频数据。fread函数可以一次读取一个字节，也可以一次读取一个结构体的所有变量值，并自动移动文件指针，指向下一个需要读取的位置。

4 语

用C语言编程读取二进制音频文件内容，比较复杂，但代码效率高，得到音频数据后，便于后面用C语言进行进一步的处理，例如，分帧、提取语音特征参数或进行语音滤波等，这些工作在语音信号处理、语音识别中经常用到。

参考文献：

[1] Haitao Luo, Local Thresholding De- noise Speech Signal, Fifth International Conference on Digital Image Processing(ICDIP 2013), 8878-48.

[2] 李敏.音频文件格式WAVE的转换[J].电脑知识与技术,2005(8)73-75.

[3] 徐济仁.WAV文件格式实例分析[J].微型机与应用,2002(3):50-51.

[4] 舒广,丁晓明,RIFF WAVE文件的结构及应用[J].电声技术,200(1):49-51.