录音转换原理研究（wave格式数据）

最近公司用的VOX录音文件转换为wav文件的需求。经过一周的收集与研究，总结如下文档与大家分享。也请语音方面大神指点迷津。

# 前言

本文参考了网络其他大神的文章，语音转换方面资料还是蛮多的，但是绝大部分是C语言（毕竟这方面接近硬件方向）的同一套代码，不过在浏览了大量的文章后，也收获了对wave格式的一个整体理解和认识，最终用JAVA语音完成了录音的格式完美转换。

这里主要讨论了三个方面的问题：

a、文件格式之间如何转换（例如：ADPCM的vox转换为PCM的wav）；

b、文件编码之间如何转换（主要涉及到wave格式的调整，存储数据位数的变化等）

c、各个文件的编码算法是什么（这个主要涉及具体如何进行编码实现）

研究主要工具及方法：

工具：

GoldWave：录音编辑工具，方便于录音文件属性查看、调整及编辑等

UltraEdit：强大的编辑器，便于16进制查看、编辑录音文件

UltraCompare：强大的文件比较器，用于二进制对比两个录音文件

Windows 计算器：用来16进制与10进制等其他进制转换。这里主要是分析录音文件使用。

方法：

刚开始对于录音转换完全没有概念，于是从要转换的录音格式VOX（ADPCM）与wav（PCM）入手，先了解两个文件的格式及编码方法等。最后阅读代码并对应相应的算法说明，慢慢的理解了音频文件的编码。在动手编写代码实现时，总是发现转码后的录音文件无法直接被GoldWave识别，总是有部分问题。譬如文件长度不一致，文件内部错误等等情况。最终通过使用UltraEdit、UltraCompare对比和编辑后，发现具体问题所在，经过修改后转换正常。

总结一下，大家可以使用待转换的录音，通过GoldWave工具进行录音格式转换为目标录音格式，最后再对比自己代码生成的文件与GoldWave转换的文件差别。网络上录音转换算法差不多都是正确的，主要是转换出来的录音是否可以自动被播放设备识别的事了。

此外，wave文件格式转换前，可以根据源文件长度、音频格式以及目标文件的音频格式进行理论推导的目标文件的文件长度的。该部分需要在理解wave文件格式以及文件数据组织方式情况下进行，具体可参考本文的wave具体格式等段落。举例如下（可以同后面的各个块的组织格式一起阅读，方便理解具体参数含义）

源文件格式：PCM、16bit、8000Hz、单通道、文件长度为205584字节

目标文件格式：IMA-ADPCM、4bit、8000Hz、单通道

推导：

**源文件属性：**

**采样数据长度** = 205584 -44 //44: PCM格式head长度（字节）

**采样数据个数** = (205584 -44) / (16/8) // (16/8) : 16bit 换算为字节数

= (205584 -44) / 2

= 102770

**转换后的目标文件属性预测：**

**单纯采样数据长度** = (205584 -44) / 4 //4: 源文件的16bit[2字节]，转换为目标4bit[1/2字节]，即压缩比4:1

**采样数据个数** = 102770 // 采样个数是不变的

**Block块的大小**= 256 // 我使用的是256字节一个块（包含4byte的blockHead + 252byte的Data）详解blockHead块组织方式。网络上说可以是其他的大小；

**Block块包含采样个数** = Data\*2 + 1 // Data\*2：1Byte（字节）存储2个4bit（位）的采样个数；+ 1：blockHead包含一个未压缩的16bit的采样数据

=252\*2 +1

=505

**Block块个数** = 采样数据个数 / Block块包含采样个数 // 该方法应使用向上取整逻辑。即，最后不足505个采样的block块也要导出。

= 102770 / 505

= 203.5049504950495

= 204 // 向上取整，0.5049…的块数据也不能丢失

**最后一个块的长度** = (采样数据个数 – 203 \* 505 – 1) / 2 + 4 // 203 \* 505：前面完整块的采样个数；-1：head中未压缩的采样个数；/ 2：1个采样存储于4bit（1/2字节）；+ 4：blockHead的4byte

=(102770 -102515 -1)/2 +4

=254/2 + 4 // 主意，这里除以2同样需要向上取整的（不能丢失每一个采样数据）

=131

**采样数据组织为Block块后的长度** = 203 \* Block块的大小 + 最后一个块的长度

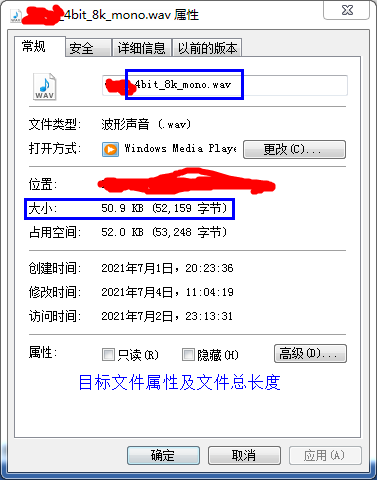
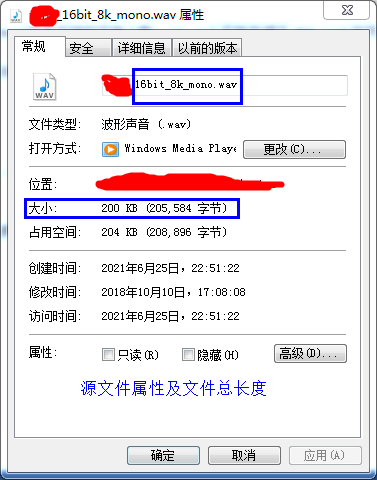
= 203 \* 256 + 131

= 52099

**文件长度(字节)**= 采样数据组织为Block块后的长度 + 60 // 60 ：IMA-ADPCM文件head长度（字节）

= 52099 + 60 // 2: 采样数据大小转换为字节单位，即 4bit 等于 1/2的字节

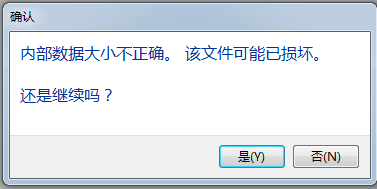
= 52159



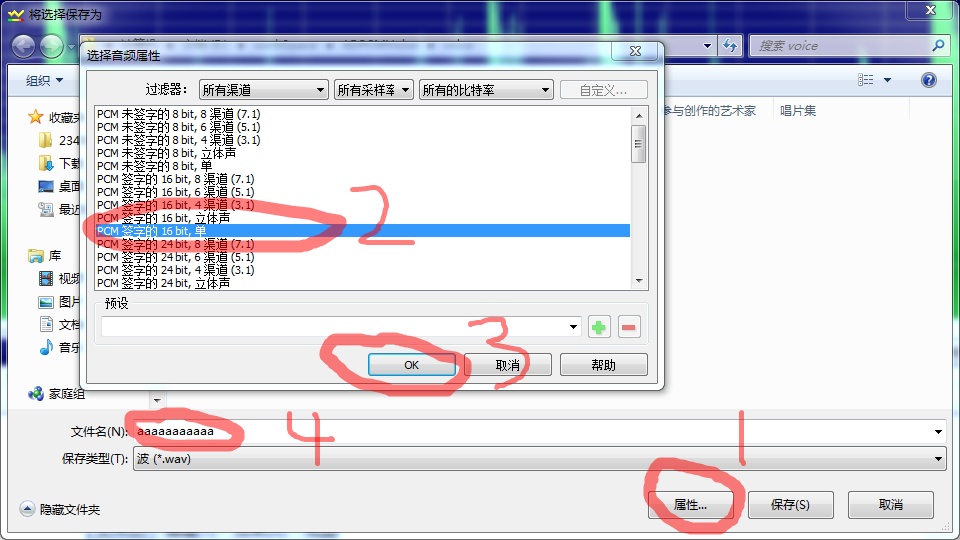
以上推导，是在完全了解wave的PCM、IMA-ADPCM格式的具体格式下进行的。有助于我们排查文件不能正常识别的具体原因。

工具使用举例如下，

在我转换完成的文件GoldWave总报”内部数据大小不正确。该文件可能已损坏”，“文件长度不一致”，“无法确认文件格式”等错误。



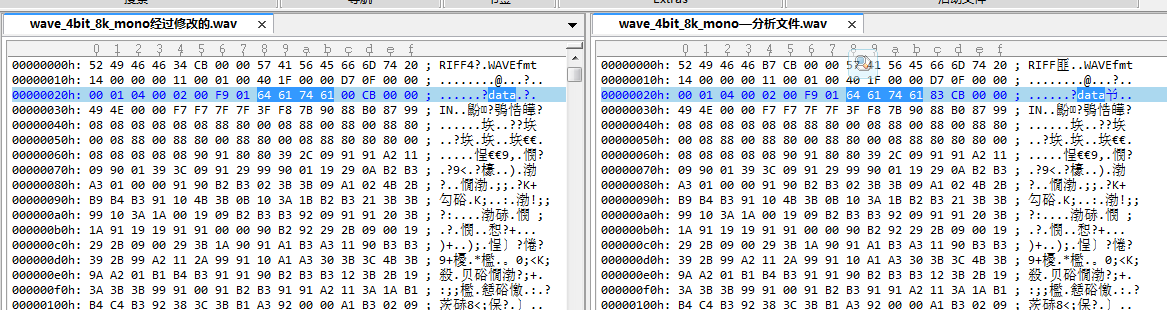
我点击是，并强行用ClodWave打开后，再使用“文件”》》“将选择保存为”， 然后现在我需要转换的目标格式（PCM 16bit 单声道）。这样就有了一个与代码生成的文件的可参考的正确的格式文件。

通过使用UltraEdit、UltraCompare将两个文件对比，主要对比两个文件的表头信息和文件DATA部分的长度信息。具体如下（以下截图是用来做步骤说明的，非同一个案例截图，）

UltraEdit示例图如下：

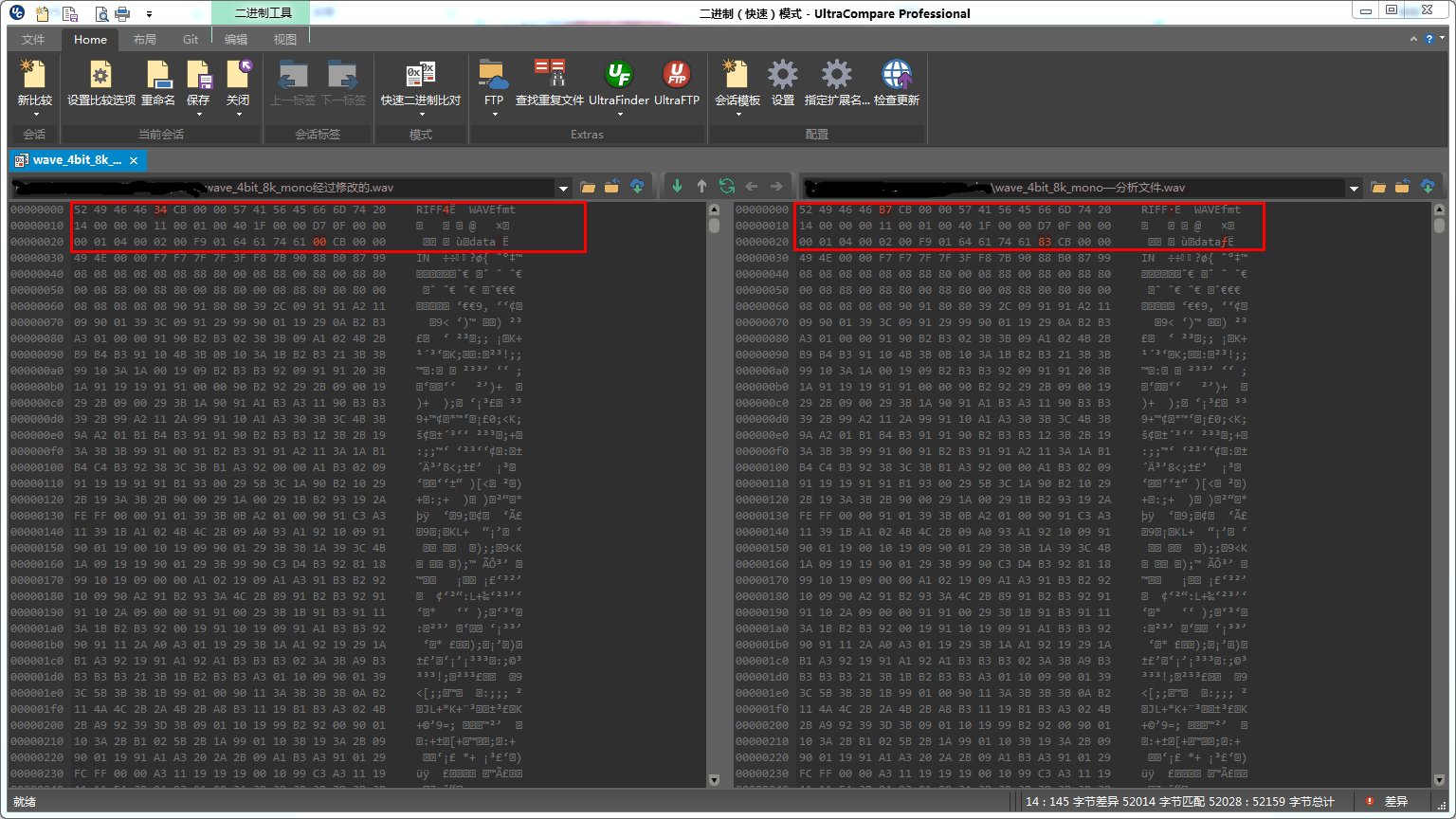
可以很好的定位到具体的16进制对应的ACSII码的具体符合。途中是找到了wave文件表头中data块的data字符串

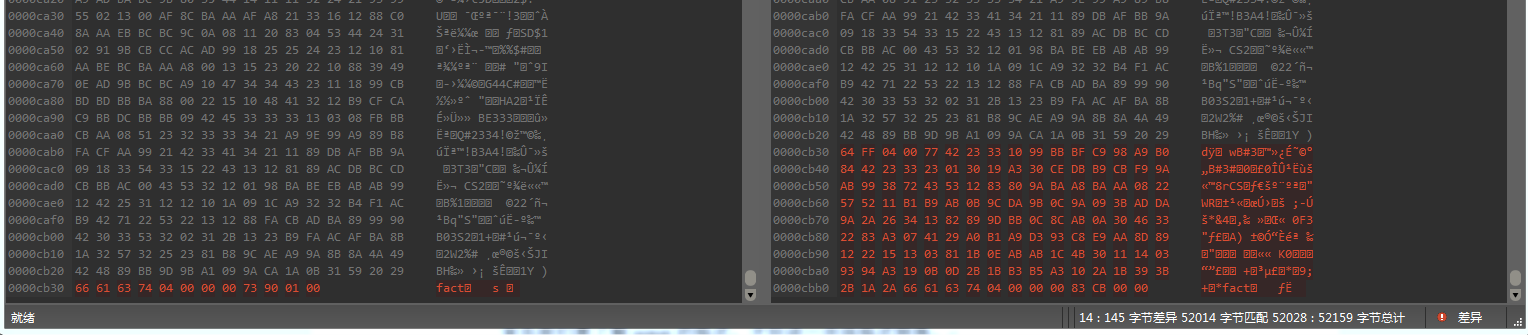


UltraCompare示例图如下，

左边的是经过GoldWave修改的正确的文件，右边是我自己代码生成的文件。

注意：GoldWave修改后的文件我的fact块跑到文件末尾了。需要使用UltraEdit进行文件编辑调整位置，以方便对比数据。





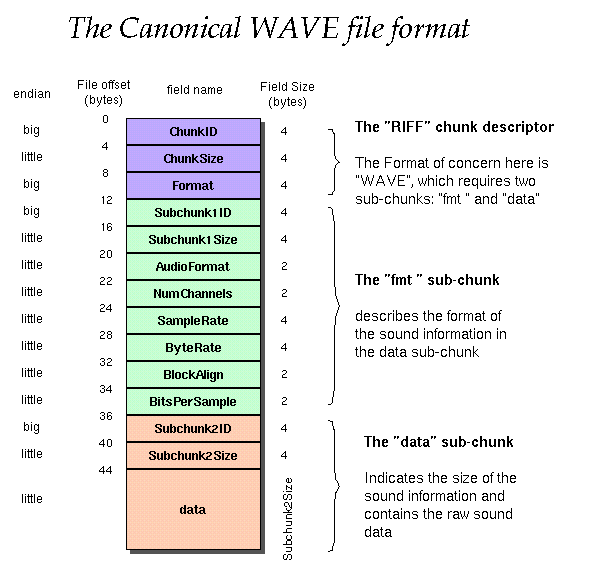
# Wave格式说明

首先我们要了解wave的格式，才好进一步将格式转换。

这块网上资料较详细，***我根据自己理解整理使用的地方如下，也请指正***：

Wave整体式说明

Wave PCM编码的文件头信息及解释：



wave具体格式总结:

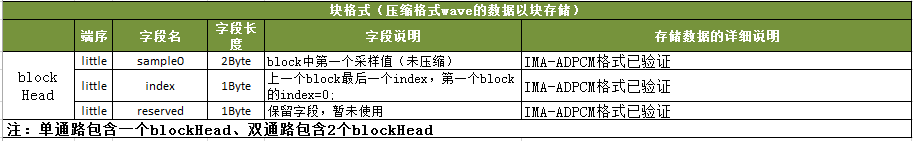


**以上灰色底纹部分仅在压缩格式下才会存在**。



BlockHead块组织格式：

压缩格式（这里参考IMA-ADPCM）下单Block块的格式如下：



DATA数据存储方式说明

采样的字节存储方式说明：

8 Bit 单声道：

|  |  |
| --- | --- |
| 采样1 | 采样2 |
| 数据1 | 数据2 |

8 Bit 双声道：

|  |  |  |  |
| --- | --- | --- | --- |
| 采样1 | | 采样2 | |
| 声道1数据1 | 声道2数据1 | 声道1数据2 | 声道2数据2 |

16 Bit 单声道：

|  |  |  |  |
| --- | --- | --- | --- |
| 采样1 | | 采样2 | |
| 数据1**低**字节 | 数据1**高**字节 | 数据1**低**字节 | 数据1**高**字节 |

16 Bit 双声道

|  |  |  |  |
| --- | --- | --- | --- |
| 采样1 | | | |
| 声道1数据1低字节 | 声道1数据1高字节 | 声道2数据1低字节 | 声道2数据1高字节 |
| 采样2 | | | |
| 声道1数据2低字节 | 声道1数据2高字节 | 声道2数据2低字节 | 声道2数据2高字节 |

单通路压缩后存储格式对比：

1、单通路16Bit 的pcm格式



单通路压缩为adpcm数据为 4bytes block head + 块内音频数据：



**其中sample1编码后存data0低4位，sample2编码后存data0高四位...**

2、双通路16Bit 的pcm格式



双通路压缩为adpcm数据为 4bytes block L head + 4bytes block R head + 4bytes raw L data + 4bytes raw R data…：

adpcm双通路block head：



接着双通路块内音频数据压缩数据4byte L， 4byte R …



**注意：需要特别留意双声道的处理和当数据不够1 block时的处理方式。**

举个栗子：下面是两个wave文件头的例子，帮助你了解分析文件头是否正确；

示例1：声音文件如下：

52 49 46 46 24 08 00 00 57 41 56 45

66 6d 74 20 10 00 00 00 01 00 02 00

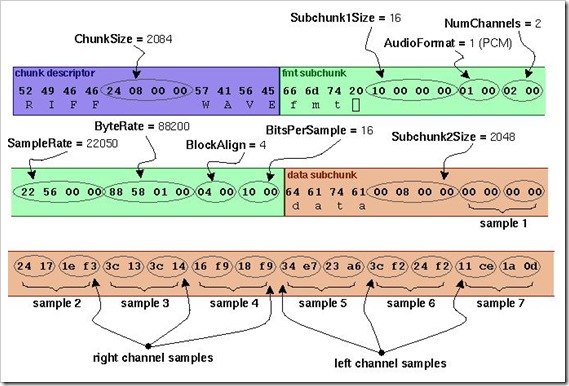
22 56 00 00 88 58 01 00 04 00 10 00

64 61 74 61 00 08 00 00 00 00 00 00

24 17 1e f3 3c 13 3c 14 16 f9 18 f9

34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d

对应的分析如下图所示：

[](http://images.cnblogs.com/cnblogs_com/liyiwen/WindowsLiveWriter/WAVE_133DA/untitled.jpg)

示例2：文件实例说明

***实例分析***

**

（1）“52 49 46 46”这个是Ascii字符“RIFF”，这部分是固定格式，表明这是一个WAVE文件头。  
（2）“22 60 28 00”，这个是我这个WAV文件的数据大小，这个大小包括除了前面4个字节的所有字节，也就等于文件总字节数减去8。16进制的“22 60 28 00”对应是十进制的“2646050”。  
（3）“57 41 56 45 66 6D 74 20”，也是Ascii字符“WAVEfmt”，这部分是固定格式。  
以后是PCMWAVEFORMAT部分

（4）“12 00 00 00”，这是一个DWORD，对应数字18，这个对应定义中的PCMWAVEFORMAT部分的大小，可以看到后面的这个段内容正好是18个字节。一般情况下大小为16，此时最后附加信息没有，上面这个文件多了两个字节的附加信息。  
（5）“01 00”，这是一个WORD，对应定义为编码格式（WAVE\_FORMAT\_PCM格式一般用的是这个）。  
（6）“01 00”，这是一个WORD，对应数字1，表示声道数为1，是个单声道Wav。  
（7）“22 56 00 00”对应数字22050，代表的是采样频率22050，采样率（每秒样本数），表示每个通道的播放速度  
（8）“44 AC 00 00”对应数字44100，代表的是每秒的数据量，波形音频数据传送速率，其值为通道数×每秒样本数×每样本的数据位数／8（1\*22050\*16/8）。播放软件利用此值可以估计缓冲区的大小。  
（9）“02 00”对应数字是2，表示块对齐的内容。数据块的调整数（按字节算的），其值为通道数×每样本的数据位值／8。播放软件需要一次处理多个该值大小的字节数据，以便将其值用于缓冲区的调整。  
（10）“10 00”数值为16，采样大小为16Bits，每样本的数据位数，表示每个声道中各个样本的数据位数。如果有多个声道，对每个声道而言，样本大小都一样。  
（11）“00 00”此处为附加信息（可选），和（4）中的size对应。

（12）“66 61 73 74” Fact是可选字段，一般当wav文件由某些软件转化而成，则包含该项，“04 00 00 00”Fact字段的大小为4字节，“F8 2F 14 00”是fact数据。

（13）“64 61 74 61”，这个是Ascii字符“data”，标示头结束，开始数据区域。  
（14）“F0 5F 28 00”十六进制数是“0x285ff0”,对应十进制2646000，是数据区的开头，以后数据总数，看一下前面正好可以看到，文件大小是2646050，从（2）到（13）包括（13）正好是2646050-2646000=50字节。

**录音文件编码/解码算法（编解码重点算法）**

了解了wave文件的组织格式，理论上来说就可以按照录音文件规范标准进行相应的格式转换了。转换过程中我们需要知道不同格式文件转向目标文件直接的数据转换逻辑，即每个文件的编码/解码逻辑。本文参考：《WAV系列之二：ADPCM编解码原理及代码实现》 描述下PCM\ADPCM数据算法。

IMA-ADPCM编码原理：

IMA-ADPCM 是Intel公司首先开发的是一种主要针对16bit采样波形数据的有损压缩算法，压缩比为 4：1*，它与通常的DVI-ADPCM是同一算法*。 (对8bit数据压缩时是3.2：1，也有非标准的IMA-ADPCM压缩算法，可以达到5：1甚至更高的压缩比) 4：1的压缩是目前使用最多的压缩方式。

ADPCM编码本质是一种预测编码，是一种采用变步长的量化器的预测编码算法，它的本质是根据预测值与实际的偏差范围，在量化表格中选择出合适的量化值，使预测变化的幅度保持在4bit的范围内。**ADPCM的核心公式如下(划重点)**，其中

**delta** 代表为量化后的值，

**diff** 当前实际值与预测值的偏差

**step** 为量化步长，

**vpdiff** 代表经过量化后有效的偏差值，

**valpred** 代表上**一次的**预测值。

**下一次的预测值** = vpdiff + 本次的预测值（即为blockHead中的一个未压缩的采样数据的值）

https://img-blog.csdnimg.cn/20200309104115501.png

在这里插入图片描述

整个ADPCM的**编码过程**分三步进行：

第一步：计算出当前实际值与预测值的偏差diff，val 代表了当前数据的实际值，valpred 为当前数的预测值。delta 为量化后的带符号的有效数据为4bit的数据，其最高位代表的数据的方向，bit3为1代表负数，代表-7~7的整型数据。

在这里插入图片描述

当 diff 小于0， delta bit3被置1。表示为负向偏差。

第二步：通过index（首次编码index为0）求出step，通过diff和step求出delta。

利用上面公式： https://img-blog.csdnimg.cn/20200309104115501.png

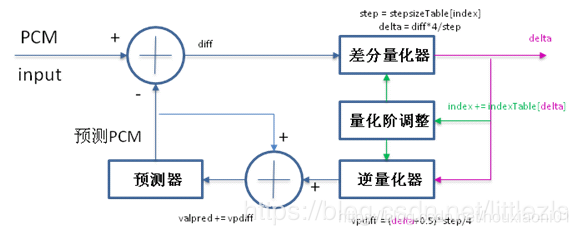
第三步：对 diff 进行量化，简易实现不考虑计算效率的情况下完全可以直接参考上面的公式，因为是在计算机平台进行了除法运算与小数运算，该作者很巧妙的把这些运算使用与或非来实现了，提高了运算的效率，有兴趣的读者可以看看代码，学习一下这种思路。我们细看一下公式，

在这里插入图片描述

可以发现公式可以拆分为两部分实现，小数部分的量化被转换为了固定的step/8，因此节约了计算的成本。vpdiff 就是对应这部分的值。

vpdiff = (step >> 3);

adpcm编码原理



编码步骤：

1、求出输入的pcm数据与预测的pcm数据（第一次为上一个pcm数据）的差值diff；

2、通过差分量化器算出delta（通过index（首次编码index为0）求出step，通过diff和step求出delta)。delta即为编码后的数据；

3、通过逆量化器求出vpdiff(通过求出的delta和step算出vpdiff）；

4、求出新的预测valpred，即上次预测的valpred+vpdiff；

5、通过预测器（归一化），求出当前输入pcm input的预测pcm值，为下一次计算用；

6、量化阶调整（通过delta查表及index，计算出新的index值）。为下次计算用；

**结合代码中的计算进行说明：**

/\*\*

\* step 1:求出 当前实际值(val) 与 上一次预测值(valPred)的偏差 diff

\*/

diff = val - valpred;

/\*\*

\* 获取偏差的方向 8表示负向（1000） 0表示正向（0000），用于设置delta的方向

\* 当 diff 小于0， delta bit3被置1。表示为负向偏差

\*/

sign = (diff < 0) ? 8 : 0;

**if** (sign != 0) {

diff = (-diff);

}

/\*

\* step 2:求出delta、vpdiff

\* 使用公式 1:

\* vpdiff = (delta + 0.5)\*step/4

\* = delta\*step/4 + step/8

\* = delta\*step/4 + step>>3

\* 公式2：

\* delta = diff \* 4 / step

\* 初始化 delta的范围为[0, 7]

\*/

delta = 0;

vpdiff = (step >> 3);

/\*\*

\* 当diff >= step时 分析如下：

\* 第一行：delta = 4;解释如下

\* delta = diff \* 4 / step

\* = (diff-step) \* 4 /step + 4 //diff-step>0

\* 第二行：diff -= step;解释如下

\* 重置diff的值，以便后面的进一步计算

\* 第三行：vpdiff += step;解释如下

\* 公式1 与 公式2 联合起来

\* vpdiff = (delta + 0.5)\*step/4

\* = delta\*step/4 + step/8

\* = delta\*step/4 + step>>3

\* = (diff \* 4 / step)\*step/4 + step>>3

\* = diff + step>>3

\* = (diff-step) + step + step>>3

\* 当 (diff-step) 可以忽略为0时

\* 所以 vpdiff += step;

\* 经过以上步骤 细分的diff = diff-step;后面继续计算

\*/

**if** (diff >= step) {

delta = 4;

diff -= step;

vpdiff += step;

}

/\*\*

\* 将step >>= 1; 即： step = (int)(step / 2);

\* 此时，当diff >= step时 diff >= (step / 2)

\* 第一行：delta |= 2;解释如下

\* delta = diff \* 4 / step

\* = ((diff \* 4)/2) / (step/2)

\* delta\*2 = diff \* 4 / (step/2)

\* = (diff-(step/2))\*4 / (step/2) + 4

\* delta\*2-4 = (diff-(step/2))\*4 / (step/2)

\* 因此当右边的可忽略为0时：delta\*2-4 = 0推出

\* delta = 4/2 = 2

\* 即 delta += 2;

\* 第二行：diff -= step;解释如下

\* 重置diff的值，以便后面的进一步计算

\* 第三行：vpdiff += step;解释如下

\* 公式1 与 公式2 联合起来

\* vpdiff = diff + step>>3

\* = (diff-(step/2)) + (step/2) + step>>3

\* 所以 vpdiff += (step/2); 因为step = (int)(step / 2);

\* 所以 vpdiff += step;

\*/

step >>= 1;

**if** (diff >= step) {

//delta |= 2;

delta += 2;

diff -= step;

vpdiff += step;

}

/\*\*

\* 继续将step >>= 1; 即：当前 step = step >>= 2; 则 step = (int)(step/4)

\* 此时，当diff >= step时 diff >= (step / 2)

\* 第一行：delta |= 2;解释如下

\* delta = diff \* 4 / step

\* = ((diff \* 4)/4) / (step/4)

\* delta\*4 = diff \* 4 / (step/4)

\* = (diff-(step/4))\*4 / (step/4) + 4

\* delta\*4-4 = (diff-(step/4))\*4 / (step/4)

\* 因此当右边的可忽略为0时：delta\*4-4 = 0推出

\* delta = 4/4 = 1

\* 即 delta += 1;

\* 第二行：diff -= step;解释如下

\* 重置diff的值，以便后面的进一步计算

\* 第三行：vpdiff += step;解释如下

\* 公式1 与 公式2 联合起来

\* vpdiff = diff + step>>3

\* = (diff-(step/4)) + (step/4) + step>>3

\* 所以 vpdiff += (step/4); 因为step = (int)(step / 4);

\* 所以 vpdiff += step;

\*

\* 此时，默认忽略diff-(step/4)的值，因为

\* 当diff的值是step的n倍(n>=2)时，例如2倍：即diff = 2\*step

\* 1、在第一次if (diff >= step) 时

\* diff -= step; 即 diff = 2\*step -step = 1\* step

\* delta = 4

\* 2、在第二次if (diff >= step) 时，step >>=1 即 step= step/2

\* diff -= step; 即diff -= (step/2)

\* diff = 0.5\*step;

\* delta = 4 +2 = 6

\* 3、在第三次if (diff >= step) 时，又一次step >>=1 即 step= step/4

\* diff -= step; 即diff -= (step/4)

\* diff = 0.25\*step;

\* delta = 6 +1 = 7

\* 4、如果继续处理也是无法将diff消除为0的，且delta已经达到最大值 7。

\* 当diff的值是step的n倍(n<2)时，例如1.5倍：即diff = 1.9\*step

\* 1、在第一次if (diff >= step) 时

\* diff -= step; 即 diff = 1.9\*step -step = 0.9\* step

\* delta = 4

\* 2、在第二次if (diff >= step) 时，step >>=1 即 step= step/2

\* diff -= step; 即diff -= (step/2)

\* diff = 0.4\*step;

\* delta = 4 +2 = 6

\* 3、在第三次if (diff >= step) 时，又一次step >>=1 即 step= step/4

\* diff -= step; 即diff -= (step/4)

\* diff = 0.15\*step;

\* delta = 6 +1 = 7

\* 4、此时diff已经小于0.15\*step可忽略，且delta已经达到最大值 7。

\*/

step >>= 1;

**if** (diff >= step) {

//delta |= 1;

delta += 1;

vpdiff += step;

}

/\*\*

\* step 3：将vpdiff的正负号加上，形成完整的vpdiff

\* step 4：求出新的预测valpred，即上次预测的valpred+vpdiff

\*/

**if** (sign != 0)

valpred -= vpdiff;

**else**

valpred += vpdiff;

**if** (valpred > 32767)

valpred = 32767;

**else** **if** (valpred < -32768)

valpred = -32768;

/\*\*

\* 量化后的值加上正负号

\*/

delta |= sign;

index += *indexTable*[delta];

**if** (index < 0)

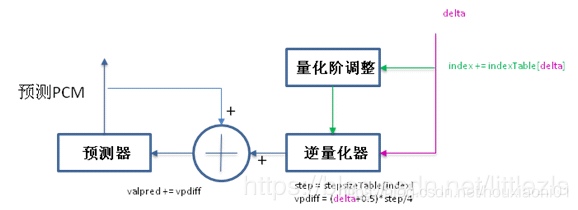
index = 0;

**if** (index > 88)

index = 88;

step = *stepsizeTable*[index];

adpcm解码原理



解码步骤（其实解码原理就是编码的第三到六步）：

1、通过逆量化器求出vpdiff(通过存储的delta和index，求出step，算出vpdiff）；

2、求出新的预测valpred，即上次预测的valpred+vpdiff；

3、通过预测器（归一化），求出当前输入pcm input的预测pcm值，为下一次计算用。预测的pcm值即为解码后的数据；

4、量化阶调整（通过delta查表及index，计算出新的index值）。为下次计算用；

注释说明：

通过编码和解码的原理我们可以看出其实第一次编码的时候已经进行了解码，即预测的pcm。

因为编码再解码后输出的数据已经被量化了。根据计算公式delta = diff\*4/step; vpdiff = (delta+0.5)\*step/4;考虑到都是整数运算，可以推导出：pcm数据经过编码再解码生成的预测pcm数据，如果预测pcm数据再次编码所得的数据与第一次编码所得的数据是相同的。故pcm数据经过一次编码有损后，不论后面经过几次解码再编码都是数据一样，音质不会再次损失。即相对于第一次编码后，以后数据不论多少次编解码，属于无损输出。

**结合代码中的计算进行说明：**

/\*\*

\* 根据量化后的值获取 步进的索引值

\* index 范围[0,88]

\*/

index += *indexTable*[delta];

/\*\*

\* 值域检查

\*/

**if** (index < 0)

index = 0;

**if** (index > 88)

index = 88;

/\*\*

\* 取符号位 &（与运算）都为1才为1

\*/

sign = delta & 8;// sing只会等于8或者0

// 取数据:除符合位以外的3位数据

delta = delta & 7;

// 下边四则运算将vpdiff = (delta+0.5)\*step/4四则运算转换成了二进制的与或运算（牛逼）

/\*\*

\* coderRealize的逆运算

\* 当(delta & 4) != 0时 ，即：delta >= 4 (不太清楚的可以自己算下这个与运算)

\* vpdiff = (delta + 0.5)\*step/4

\* = delta\*step/4 + step/8

\* = delta\*step/4 + step>>3 (当delta >= 4时)

\* 一种变形 = (delta-4)\*step/4 + step + step>>3 (当delta-4 小于1时可忽略(delta-4)\*step/4)

\* 另外一种变形= step\*(delta/4) + step>>3

\* 约 = step + step>>3

\* 则vpdiff += step;

\*/

vpdiff = step >> 3;

**if** ((delta & 4) != 0) {

vpdiff += step;

}

/\*\*

\* 当(delta & 2) != 0时 ，即：delta >= 2 (不太清楚的可以自己算下这个与运算)

\* vpdiff = (delta + 0.5)\*step/4

\* = delta\*step/4 + step/8

\* = delta\*step/4 + step>>3 (当delta >= 4时)

\* = step\*(delta/4) + step>>3

\* 约= step \* 1/2 + step>>3

\* 则vpdiff += step >> 1; 即 vpdiff += step/2;

\*/

**if** ((delta & 2) != 0) {

vpdiff += step >> 1;

}

/\*\*

\* 当(delta & 1) != 0时 ，即：delta >= 1 (不太清楚的可以自己算下这个与运算)

\* vpdiff = (delta + 0.5)\*step/4

\* = delta\*step/4 + step/8

\* = delta\*step/4 + step>>3 (当delta >= 4时)

\* = step\*(delta/4) + step>>3

\* 约= step \* 1/4 + step>>3

\* 则vpdiff += step >> 2; 即 vpdiff += step/4;

\*/

**if** ((delta & 1) != 0) {

vpdiff += step >> 2;

}

/\*\*

\* 给vpdiff加上正负号，并计算预测值

\*

\*/

**if** (sign != 0) {

valpred -= vpdiff;

} **else** {

valpred += vpdiff;

}

/\*\*

\* 值域检查

\*/

**if** (valpred > 32767)

valpred = 32767;

**else** **if** (valpred < -32768)

valpred = -32768;

/\*\*

\* 获取下一个步差，下次循环使用

\*/

step = *stepsizeTable*[index];

JAVA源码地址:

GIT ssh：[git@github.com:LJ0251/my\_study.git](mailto:git@github.com:LJ0251/my_study.git)

GIT https: <https://github.com/LJ0251/my_study.git>

参考列表：

wave是什么？参见：

wav文件格式分析与详解

<https://www.cnblogs.com/ranson7zop/p/7657874.html>

WAVE 文件格式分析

<https://www.cnblogs.com/muxue/archive/2010/04/19/1715715.html>

wave各个格式说明？参见：

WAV格式常见的几种压缩形式(compression code)：

<https://blog.csdn.net/stelalala/article/details/17058755>

WAV系列之二：ADPCM编解码原理及代码实现

<https://blog.csdn.net/houxiaoni01/article/details/104702570>