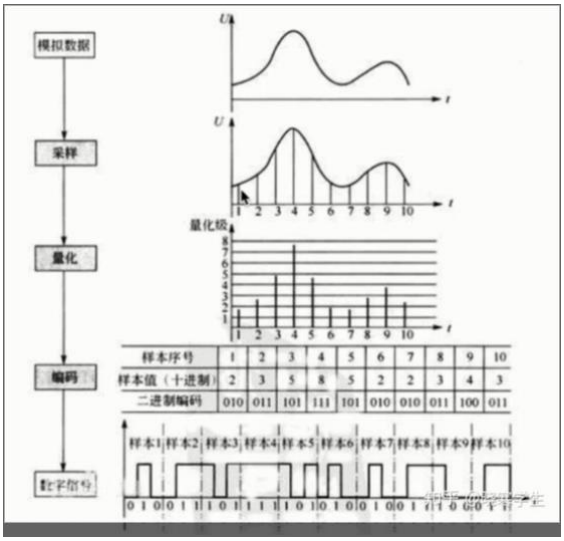
音频背景知识

音量（响度）
响度就是声音的大小强弱。取决于物体振动的幅度
自然界中的声音是模拟信号，计算机是不能直接存储模拟信号的，我们需要把声音转成二进制数据，



图中表示 PCM 编码的过程。通过抽样、量化、编码三个步骤将连续变化的模拟信号转换为数字信号

采样位数（采样精度 、 采样大小）
一个采样用多少 bit 存放。这个数值越大，解析度就越高，录制和回放的声音就越真实。8 位代表 2 的 8 次方--256，16 位则代表 2 的 16 次方--64K。常用的是 16bit。

采样率
单位时间内对媒体对象的采样次数，单位 Hz。采样频率是指录音设备在一秒钟内对声音信号的采样次数，用 4.1 K 的采样率，就是 1 秒采样 44100 次。采样频率越高，声音的还原就越真实越自然。

5kHz 的采样率仅能达到人们讲话的声音质量。
11kHz 的采样率是播放小段声音的最低标准，是 CD 音质的四分之一。
22kHz 采样率的声音可以达到 CD 音质的一半，目前大多数网站都选用这样的采样率。
44kHz 的采样率是标准的 CD 音质，可以达到很好的听觉效果。

声道数
分为单声道，双声道，多声道。举个列子，声道多，效果好，两个声道，说明只有左右两边有声音传过来， 四声道，说明前后左右都有声音传过来。

码率（比特率）

表示压缩编码后每秒的音频数据量大小。计算公式：比特率 = 采样率 x 采样精度 x 声道数。单位 kbps。例如：

采样率为 44.1KHz，采样大小为 16bit，双声道的 PCM 编码的 WAV 文件，它的码率为

44.1K×16×2=1411.2 kbps

编码器
音频编解码器是一个执行算法的计算机程序，能压缩与解压缩数字音频数据到音频文件或流媒体音频编码格式。该算法的目的是保证质量的前提下使用最少的字节表示高保真音频信号。这可以有效地减少存储空间和传输已存储音频文件所需的带宽。

PCM 脉冲编码调制是 Pulse CodeModulation 的缩写。上面的图片演示了 PCM 大致的工作流程，PCM 编码的最大的优点就是音质好，最大的缺点就是体积大。我们常见的 AudioCD 就采用了 PCM 编码，一张光盘的容量只能容纳 72 分钟的音乐信息。在选择编码器的时候，有一些因素是必须要考虑的。

编码器考虑的因素：
最佳压缩比；
算法的复杂度；
算法延时；
针对特殊场景下的特定设计；
兼容性。

通过一些特定的压缩算法，可以压缩音频文件至原来的 1/10，同时人耳也无法分辨压缩前后的声音质量差异，需要满足多种条件才能实现这种效果；而对于编码器，无论是设计阶段还是使用阶段，我们都需要考虑最佳压缩效果、算法的复杂度与算法的延时，结合特殊场景进行特定的设计；而兼容性也是我们不能不考虑的重点。

压缩
压缩音频，主要是为了存储和传输。

我们看看下面的例子。

长度为 4 分钟，采样频率为 44100Hz,采样深度为 16bits,双声音 Wav 文件大小：
44100Hz*16bits*4minutes*2=(44100/1second)*16bits*(4minutes*(60seconds/1minutes)*2=705600bits/second*240seconds=169344000bits=169344000/(8bits/1byte)*2=42336000bytes=42336000/(1048576/1M)bytes=40.37MB

MP3，128kbps 压缩后文件大小：
128kbps*4minutes=(128kbits/1second)*(4minutes*(60seconds/1minutes))=(128kbits/1second)*240seconds=30720kbits=30720kbits/(8bits/1byte)=3840kbytes=3840k/(1024k/1M)bytes=3.75Mbytes=3.75MB
正如上面的例子，声音压缩后，存储大小为原大小的十分之一，压缩率十分可观！
常见的音频编码器包括 OPUS、AAc、 Vorbis、Speex、iLBC、AMR、G.711 等

采集音频

采集音频，参数是采样率 8k Hz，单声道，采样位数 16，数据传输率 128kbps.
语音信号的带宽为 300-3400HZ,采样频率不少于 8KHZ,
若量化精度为 16 位,单声道输出,那么每秒钟的数据量和每小时的数据量各是多少?
数据传输率=8khz*16 b*1=128 kbps
每秒钟的数据量=128*1000*1/8B=16000B=16000kB/1024=15.625KB (也就是每秒 128/8 = 16000b/s(每秒 16k 字节))

PCM 格式音频, windows 能够识别音频的最小时间为 20ms，这个时间间隔，可以获取的数据量 8 kHz * (16b/8)

* 1 * 20ms = 320 byte

经过试验，通过编码器编码和解码，采集 640 字节对应的音频，播放起来没有问题(一次处理 320 字节 ，会出现声

音断断续续的问题 ，待研究)

使用 QT 模块采集音频和播放音频

获取声音和播放声音的组件是 Qt 自带的 QAudioInput 和 QAudioOutput ,
(使用时需要在 pro 文件中添加媒体库的引用 QT += multimedia).
1.首先初始化设备

```
//声卡采样格式
    QAudioFormat format;
    format.setSampleRate(8000);
    format.setChannelCount(1);
    format.setSampleSize(16);
    format.setCodec("audio/pcm");
    format.setByteOrder(QAudioFormat::LittleEndian);
    format.setSampleType(QAudioFormat::UnSignedInt);
    QAudioDeviceInfo info = QAudioDeviceInfo::defaultInputDevice();
    if (!info.isFormatSupported(format)) {
        QMessageBox::information(NULL , "提示", "打开音频设备失败");
        format = info.nearestFormat(format);
    }
```

2.开启采集

```
    /* QAudioInput* */audio_in = new QAudioInput(format, this);
    /*QIODevice*   */myBuffer_in = audio_in->start();//声音采集开始
    //通过定时器，每次超时，从缓冲区 m_buffer_in 中提取数据
    QTimer * timer = new QTimer(this); (父类负责回收子控件);
    connect( timer , &QTimer::timeout , this , &Audio_Read::readMore );
    timer->start(20);
```

3.定时器响应函数—将采集好的数据以信号的形式抛出

```
    void Audio_Read::readMore()
    {
```

```
        if (!audio_in)
            return;

        QByteArray m_buffer(2048,0);
        qint64 len = audio_in->bytesReady();
        if (len < 640)
        {
            return;
        }
        qint64 l = myBuffer_in->read(m_buffer.data(), 640);
        QByteArray frame;
        frame.append(m_buffer.data(),640);
        Q_EMIT SIG_audioFrame( frame );
    }
```

4.声音采集暂停

```
    audio_in->stop();
    声音采集恢复, 可以将原 QAudioInput 对象回收, 重新再申请
    if(audio_in)
    {
        delete audio_in;
    }
    timer->stop();
    delete m_timer;
    //为了更好的管理状态, 需要添加状态位, 避免出现 (开始->开始 暂停->暂停 状态的切换)
    //初始化状态为结束 , 开始状态为 recording , 暂停状态为 pausing , 先判断再执行开始和暂停.
```

5.播放音频

```
    对于播放音频, 初始化要和上面的情况类似.
    /* QAudioOutput* */audio_out = new QAudioOutput(format, this);
    /*QIODevice*    */myBuffer_out = audio_out->start();
    通过该函数 就可以完成声音播放
    void Audio_Write::slot_net_rx(QByteArray ba)
    {
        myBuffer_out->write( ba.data() , 640 );
    }
```

编码和解码

该示例中, 使用 speex 进行编码和解码,

0.包含库和头文件

```
    LIBS += $$PWD/speex/lib/libspeex.lib\
    #include <speex/include/speex.h>
```

```
//SPEEX 相关变量
SpeexBits bits_enc;
void *Enc_State;
```

1. 初始化

```
//speex 初始化
speex_bits_init(&bits_enc);
Enc_State = speex_encoder_init(speex_lib_get_mode(SPEEX_MODEID_NB));
//设置压缩质量
#define SPEEX_QUALITY      (8)
//设置质量为 8(15kbps)
int tmp = SPEEX_QUALITY;
speex_encoder_ctl(Enc_State,SPEEX_SET_QUALITY,&tmp);
```

2.编码数据

```
char bytes[800] = {0};
int i = 0;
float input_frame1[320];
char buf[800] = {0};

char* pInData = (char*)m_buffer.data() ;
memcpy( buf , pInData , 640);
int nbytes = 0;
{
        short num = 0;
        for (i = 0;i < 160;i++)
        {
            num = (uint8_t)buf[2 * i] | (((uint8_t)buf[2 * i + 1]) << 8);
            input_frame1[i] = num;
            //num = m_buffer[2 * i] | ((short)(m_buffer[2 * i + 1]) << 8);
            //qDebug() << "float in" << num << input_frame1[i];
        }
        //压缩数据
        speex_bits_reset(&bits_enc);
        speex_encode(Enc_State,input_frame1,&bits_enc);
        nbytes = speex_bits_write(&bits_enc,bytes,800);

        frame.append(bytes,nbytes);
        //大端
        for (i = 0;i < 160;i++)
        {
            num = (uint8_t)buf[2 * i + 320] | (((uint8_t)buf[2 * i + 1 + 320]) << 8);
            input_frame1[i] = num;
        }
```

```
        //压缩数据
        speex_bits_reset(&bits_enc);
        speex_encode(Enc_State,input_frame1,&bits_enc);
        nbytes = speex_bits_write(&bits_enc,bytes,800);

        frame.append(bytes,nbytes);

        qDebug() << "nbytes = " << frame.size();
        Q_EMIT   SIG_audioFrame (frame);
        return;
}
```

3.0 解码初始化

```
    //SPEEX 相关全局变量
    SpeexBits bits_dec;
    void *Dec_State;

     //speex 初始化
    speex_bits_init(&bits_dec);
    Dec_State = speex_decoder_init(speex_lib_get_mode(SPEEX_MODEID_NB));
```

3.1 解码数据

```
void Audio_Write::slot_net_rx(QByteArray ba)
{
    char bytes[800] = {0};
    int i = 0;
    float output_frame1[320] = {0};
     char buf[800] = {0};
          memcpy(bytes,ba.data(),ba.length() / 2);
    //解压缩数据 106 62
    //speex_bits_reset(&bits_dec);
    speex_bits_read_from(&bits_dec,bytes,ba.length() / 2);
    int error = speex_decode(Dec_State,&bits_dec,output_frame1);

    //将解压后数据转换为声卡识别格式
    //大端
    short num = 0;
    for (i = 0;i < 160;i++)
    {
        num = output_frame1[i];
        buf[2 * i] = num;
        buf[2 * i + 1] = num >> 8;
```

```
        //qDebug() << "float out" << num << output_frame1[i];
    }

    memcpy(bytes,ba.data() + ba.length() / 2,ba.length() / 2);
    //解压缩数据
    //speex_bits_reset(&bits_dec);
    speex_bits_read_from(&bits_dec,bytes,ba.length() / 2);
    error = speex_decode(Dec_State,&bits_dec,output_frame1);
    //将解压后数据转换为声卡识别格式

    //大端
    for (i = 0;i < 160;i++)
    {
        num = output_frame1[i];
        buf[2 * i + 320] = num;
        buf[2 * i + 1 + 320] = num >> 8;
    }
    myBuffer_out->write(buf,640);
    return;
}
```