# 语音格式转换-FFMPEG

## 离线识别引擎FFMPEG源代码清单

### 概述

#### 编写目的

文档编写目的是为代码维护者能清楚了解代码模块的具体功能及分布位置，本文档的读者对象为开发人员和测试人员。

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

### 运行环境

#### 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.0以上 |
| 数据库 | Redis | 3.2.0 |
| Mysql | 5.7.6 |
| 能力接口 | Tomcat | 9.0.35 |

#### 硬件环境

|  |  |  |
| --- | --- | --- |
| 服务器 | 最低配置 | 推荐配置 |
| redis服务器 | CPU：1  内存：256M | CPU：1  内存：1G |
| 识别服务器 | CPU：4  内存：10G  磁盘：20G | CPU：10  内存：15G  磁盘：30G |

### 3.源代码清单

#### 代码清单

|  |  |
| --- | --- |
| 程序名 | 程序路径 |
| cmdutils.c  cmdutils.h  ffmpeg.c  ffmpeg\_filter.c  ffmpeg.h  ffmpeg\_hw.c  ffmpeg\_opt.c  ffmpeg\_qsv.c  ffmpeg\_videotoolbox.c  ffplay.c  ffprobe.c  Makefile | ffmpeg-4.3/fftools |
| 012v.c  4xm.c  8bps.c  8svx.c  a64colors.h  a64multienc.c  a64tables.h  aac\_ac3\_parser.c  aac\_ac3\_parser.h  aac\_adtstoasc\_bsf.c  aaccoder.c  aaccoder\_trellis.h  aaccoder\_twoloop.h  aacdec.c  aacdec\_fixed.c  aacdectab.h  aacdec\_template.c  aac\_defines.h  aacenc.c  aacenc.h  aacenc\_is.c  aacenc\_is.h  aacenc\_ltp.c  aacenc\_ltp.h  aacenc\_pred.c  aacenc\_pred.h  aacenc\_quantization.h  aacenc\_quantization\_misc.h  aacenctab.c  aacenctab.h  aacenc\_tns.c  aacenc\_tns.h  aacenc\_utils.h  aac.h  aac\_parser.c  aacps.c  aacpsdata.c  aacpsdsp\_fixed.c  aacpsdsp\_float.c  aacpsdsp.h  aacpsdsp\_template.c  aacps\_fixed.c  aacps\_fixed\_tablegen.c  aacps\_fixed\_tablegen.h  aacps\_float.c  aacps.h  aacps\_tablegen.c  aacps\_tablegen.h  aacps\_tablegen\_template.c  aacpsy.c  aacsbr.c  aacsbrdata.h  aacsbr\_fixed.c  aacsbr\_fixed\_tablegen.h  aacsbr.h  aacsbr\_tablegen\_common.h  aacsbr\_tablegen.h  aacsbr\_template.c  aactab.c  aactab.h  aandcttab.c  aandcttab.h  aasc.c  ac3.c  ac3dec.c  ac3dec\_data.c  ac3dec\_data.h  ac3dec\_fixed.c  ac3dec\_float.c  ac3dec.h  ac3dsp.c  ac3dsp.h  ac3enc.c  ac3enc\_fixed.c  ac3enc\_float.c  ac3enc.h  ac3enc\_opts\_template.c  ac3enc\_template.c  ac3.h  ac3\_parser.c  ac3\_parser.h  ac3\_parser\_internal.h  ac3tab.c  ac3tab.h  acelp\_filters.c  acelp\_filters.h  acelp\_pitch\_delay.c  acelp\_pitch\_delay.h  acelp\_vectors.c  acelp\_vectors.h  adpcm.c  adpcm\_data.c  adpcm\_data.h  adpcmenc.c  adpcm.h  adts\_header.c  adts\_header.h  adts\_parser.c  adts\_parser.h  adx.c  adxdec.c  adxenc.c  adx.h  adx\_parser.c  agm.c  aic.c  alac.c  alac\_data.c  alac\_data.h  alacdsp.c  alacdsp.h  alacenc.c  aliaspixdec.c  aliaspixenc.c  allcodecs.c  alsdec.c  amfenc.c  amfenc.h  amfenc\_h264.c  amfenc\_hevc.c  amr.h  amrnbdata.h  amrnbdec.c  amrwbdata.h  amrwbdec.c  anm.c  ansi.c  apedec.c  apng.h  aptx.c  aptxdec.c  aptxenc.c  aptx.h  arbc.c | ffmpeg-4.3/libavcodec |
| alldevices.c  alsa.c  alsa\_dec.c  alsa\_enc.c  alsa.h  android\_camera.c  avdevice.c  avdevice.h  avdeviceres.rc  avfoundation.m  bktr.c  caca.c  decklink\_common\_c.h  decklink\_common.cpp  decklink\_common.h  decklink\_dec\_c.c  decklink\_dec.cpp  decklink\_dec.h  decklink\_enc\_c.c  decklink\_enc.cpp  decklink\_enc.h  dshow.c  dshow\_capture.h  dshow\_common.c  dshow\_crossbar.c  dshow\_enummediatypes.c  dshow\_enumpins.c  dshow\_filter.c  dshow\_pin.c  fbdev\_common.c  fbdev\_common.h  fbdev\_dec.c  fbdev\_enc.c  file\_open.c  gdigrab.c  iec61883.c  internal.h  jack.c  kmsgrab.c  lavfi.c  libavdevice.v  libcdio.c  libdc1394.c  Makefile  openal-dec.c  opengl\_enc.c  opengl\_enc\_shaders.h  oss.c  oss\_dec.c  oss\_enc.c  oss.h  pulse\_audio\_common.c  pulse\_audio\_common.h  pulse\_audio\_dec.c  pulse\_audio\_enc.c  reverse.c  sdl2.c  sndio.c  sndio\_dec.c  sndio\_enc.c  sndio.h | ffmpeg-4.3/libavdevice |
| aeval.c  af\_acontrast.c  af\_acopy.c  af\_acrossover.c  af\_acrusher.c  af\_adeclick.c  af\_adelay.c  af\_aderivative.c  af\_aecho.c  af\_aemphasis.c  af\_afade.c  af\_afftdn.c  af\_afftfilt.c  af\_afir.c  af\_afir.h  af\_aformat.c  af\_agate.c  af\_aiir.c  af\_alimiter.c  af\_amerge.c  af\_amix.c  af\_amultiply.c  af\_anequalizer.c  af\_anlmdn.c  af\_anlmdndsp.h  af\_anlms.c  af\_anull.c  af\_apad.c  af\_aphaser.c  af\_apulsator.c  af\_aresample.c  af\_arnndn.c  af\_asetnsamples.c  af\_asetrate.c  af\_ashowinfo.c  af\_asoftclip.c  af\_asr.c  af\_astats.c  af\_asubboost.c  af\_atempo.c  af\_axcorrelate.c  af\_biquads.c  af\_bs2b.c  af\_channelmap.c  af\_channelsplit.c  af\_chorus.c  af\_compand.c  af\_compensationdelay.c  af\_crossfeed.c  af\_crystalizer.c  af\_dcshift.c  af\_deesser.c  af\_drmeter.c  af\_dynaudnorm.c  af\_earwax.c  af\_extrastereo.c  af\_firequalizer.c  af\_flanger.c  af\_haas.c  af\_hdcd.c  af\_headphone.c  af\_join.c  af\_ladspa.c  af\_loudnorm.c  af\_lv2.c  af\_mcompand.c  af\_pan.c  af\_replaygain.c  af\_resample.c  af\_rubberband.c  af\_sidechaincompress.c  af\_silencedetect.c  af\_silenceremove.c  af\_sofalizer.c  af\_stereotools.c  af\_stereowiden.c  af\_superequalizer.c  af\_surround.c  af\_tremolo.c  af\_vibrato.c  af\_volume.c  af\_volumedetect.c  af\_volume.h  allfilters.c  asink\_anullsink.c  asrc\_afirsrc.c  asrc\_anoisesrc.c  asrc\_anullsrc.c  asrc\_flite.c  asrc\_hilbert.c  asrc\_sinc.c  asrc\_sine.c  atadenoise.h  audio.c  audio.h  avf\_abitscope.c  avf\_ahistogram.c  avf\_aphasemeter.c  avf\_avectorscope.c  avf\_concat.c  avfilter.c  avfiltergraph.c  avfilter.h  avfilterres.rc  avf\_showcqt.c  avf\_showcqt.h  avf\_showfreqs.c  avf\_showspatial.c  avf\_showspectrum.c  avf\_showvolume.c  avf\_showwaves.c  bbox.c  bbox.h  blend.h  boxblur.c  boxblur.h  bufferqueue.h  buffersink.c  buffersink.h  buffersrc.c  buffersrc.h  bwdif.h  colorspace.c  colorspacedsp.c  colorspacedsp.h  colorspacedsp\_template.c  colorspacedsp\_yuv2yuv\_template.c  colorspace.h  convolution.h  deshake.h | ffmpeg-4.3/libavfilter |
| 3dostr.c  4xm.c  a64.c  aacdec.c  aadec.c  ac3dec.c  acm.c  act.c  adp.c  ads.c  adtsenc.c  adxdec.c  aea.c  afc.c  aiffdec.c  aiffenc.c  aiff.h  aixdec.c  allformats.c  alp.c  amr.c  anm.c  apc.c  ape.c  apetag.c  apetag.h  apm.c  apngdec.c  apngenc.c  aptxdec.c  aqtitledec.c  argo\_asf.c  asf.c  asfcrypt.c  asfcrypt.h  asfdec\_f.c  asfdec\_o.c  asfenc.c  asf.h  assdec.c  assenc.c  ast.c  astdec.c  astenc.c  ast.h  async.c  au.c  av1.c  av1dec.c  av1.h  avc.c  avc.h  avformat.h  avformatres.rc  avidec.c  avienc.c  avi.h  aviobuf.c  avio.c  avio.h  avio\_internal.h  avisynth.c  avlanguage.c  avlanguage.h  avr.c  avs.c  bethsoftvid.c  bfi.c  bink.c  bintext.c  bit.c  bluray.c  bmv.c  boadec.c  brstm.c  c93.c  cache.c  caf.c  cafdec.c  cafenc.c  caf.h  cavsvideodec.c  cdg.c  cdxl.c  chromaprint.c  cinedec.c  codec2.c  concat.c  concatdec.c  crcenc.c  crypto.c  cutils.c  dash.c  dashdec.c  dashenc.c  dash.h  data\_uri.c  dauddec.c  daudenc.c  davs2.c  dcstr.c  derf.c  dfa.c  dhav.c  diracdec.c  dnxhddec.c  dsfdec.c  dsicin.c  dss.c  dtsdec.c  dtshddec.c  dump.c  dvbsub.c  dvbtxt.c  dv.c  dvenc.c  dv.h  dxa.c  eacdata.c  electronicarts.c  epafdec.c  ffmetadec.c  ffmetaenc.c  ffmeta.h  fifo.c  fifo\_test.c  file.c  file\_open.c  filmstripdec.c  filmstripenc.c  fitsdec.c  fitsenc.c  flacdec.c  flacenc.c  flacenc.h  flacenc\_header.c  flac\_picture.c  flac\_picture.h  flic.c  flvdec.c  flvenc.c  flv.h  format.c  framecrcenc.c  framehash.c  frmdec.c  fsb.c  ftp.c  fwse.c  g722.c  g723\_1.c  g726.c  g729dec.c  gdv.c  genh.c  gif.c  gifdec.c  golomb\_tab.c  gopher.c  gsmdec.c  gxf.c  gxfenc.c  gxf.h  h261dec.c  h263dec.c  h264dec.c  hashenc.c  hca.c  hcom.c  hdsenc.c  hevc.c  hevcdec.c  hevc.h  hls.c  hlsenc.c  hlsplaylist.c  hlsplaylist.h  hlsproto.c  hnm.c  httpauth.c  httpauth.h  http.c  http.h  icecast.c  icodec.c  icoenc.c  id3v1.c  id3v1.h  id3v2.c  id3v2enc.c  id3v2.h  idcin.c  idroqdec.c  idroqenc.c  iff.c  ifv.c  ilbc.c  img2\_alias\_pix.c  img2\_brender\_pix.c  img2.c  img2dec.c  img2enc.c  img2.h  ingenientdec.c  internal.h  ip.c  ip.h  ipmovie.c  ircam.c  ircamdec.c  ircamenc.c  ircam.h  isom.c  isom.h  iss.c  iv8.c  ivfdec.c  ivfenc.c  jacosubdec.c  jacosubenc.c  jvdec.c  kvag.c  latmenc.c  libamqp.c  libavformat.v  libgme.c  libmodplug.c  libopenmpt.c  librtmp.c  libsmbclient.c  libsrt.c  libssh.c  libzmq.c  lmlm4.c  loasdec.c  log2\_tab.c  lrc.c  lrcdec.c  lrcenc.c  lrc.h  lvfdec.c  lxfdec.c  m4vdec.c  Makefile  matroska.c  matroskadec.c  matroskaenc.c  matroska.h  md5proto.c  metadata.c  metadata.h  mgsts.c  microdvddec.c  microdvdenc.c  mj2kdec.c  mkvtimestamp\_v2.c  mlpdec.c  mlvdec.c  mm.c  mmf.c  mms.c  mms.h  mmsh.c  mmst.c  mov.c  mov\_chan.c  mov\_chan.h  movenc.c  movenccenc.c  movenccenc.h  movenc.h  movenchint.c  mov\_esds.c  mp3dec.c  mp3enc.c  mpc8.c  mpc.c  mpeg.c  mpegenc.c  mpeg.h  mpegts.c  mpegtsenc.c  mpegts.h  mpegvideodec.c  mpjpeg.c  mpjpegdec.c  mpl2dec.c  mpsubdec.c  msf.c  msnwc\_tcp.c  mtaf.c  mtv.c  musx.c  mux.c  mvdec.c  mvi.c  mxf.c  mxfdec.c  mxfenc.c  mxf.h  mxg.c  ncdec.c  network.c  network.h  nistspheredec.c  nspdec.c  nsvdec.c  nullenc.c  nut.c  nutdec.c  nutenc.c  nut.h  nuv.c  oggdec.c  oggdec.h  oggenc.c  oggparsecelt.c  oggparsedirac.c  oggparseflac.c  oggparseogm.c  oggparseopus.c  oggparseskeleton.c  oggparsespeex.c  oggparsetheora.c  oggparsevorbis.c  oggparsevp8.c  oma.c  omadec.c  omaenc.c  oma.h  options.c  options\_table.h  os\_support.c  os\_support.h  paf.c  pcm.c  pcmdec.c  pcmenc.c  pcm.h  pjsdec.c  pmpdec.c  pp\_bnk.c  prompeg.c  protocols.c  psxstr.c  pva.c  pvfdec.c  qcp.c  qtpalette.c  qtpalette.h  r3d.c  rawdec.c  rawdec.h  rawenc.c  rawenc.h  rawutils.c  rawvideodec.c  rdt.c  rdt.h  realtextdec.c  redspark.c  replaygain.c  replaygain.h  riff.c  riffdec.c  riffenc.c  riff.h  rl2.c  rm.c  rmdec.c  rmenc.c  rm.h  rmsipr.c  rmsipr.h  rpl.c  rsd.c  rso.c  rsodec.c  rsoenc.c  rso.h  rtmpcrypt.c  rtmpcrypt.h  rtmpdh.c  rtmpdh.h  rtmpdigest.c  rtmp.h  rtmphttp.c  rtmppkt.c  rtmppkt.h  rtmpproto.c  rtp.c  rtpdec\_ac3.c  rtpdec\_amr.c  rtpdec\_asf.c  rtpdec.c  rtpdec\_dv.c  rtpdec\_formats.h  rtpdec\_g726.c  rtpdec.h  rtpdec\_h261.c  rtpdec\_h263.c  rtpdec\_h263\_rfc2190.c  rtpdec\_h264.c  rtpdec\_hevc.c  rtpdec\_ilbc.c  rtpdec\_jpeg.c  rtpdec\_latm.c  rtpdec\_mpa\_robust.c  rtpdec\_mpeg12.c  rtpdec\_mpeg4.c  rtpdec\_mpegts.c  rtpdec\_qcelp.c  rtpdec\_qdm2.c  rtpdec\_qt.c  rtpdec\_rfc4175.c  rtpdec\_svq3.c  rtpdec\_vc2hq.c  rtpdec\_vp8.c  rtpdec\_vp9.c  rtpdec\_xiph.c  rtpenc\_aac.c  rtpenc\_amr.c  rtpenc.c  rtpenc\_chain.c  rtpenc\_chain.h  rtpenc.h  rtpenc\_h261.c  rtpenc\_h263.c  rtpenc\_h263\_rfc2190.c  rtpenc\_h264\_hevc.c  rtpenc\_jpeg.c  rtpenc\_latm.c  rtpenc\_mpegts.c  rtpenc\_mpv.c  rtpenc\_vc2hq.c  rtpenc\_vp8.c  rtpenc\_vp9.c  rtpenc\_xiph.c  rtp.h  rtpproto.c  rtpproto.h  rtsp.c  rtspcodes.h  rtspdec.c  rtspenc.c  rtsp.h  s337m.c  samidec.c  sapdec.c  sapenc.c  sauce.c  sauce.h  sbcdec.c  sbgdec.c  sccdec.c  sccenc.c  sctp.c  sdp.c  sdr2.c  sdsdec.c  sdxdec.c  segafilm.c  segafilmenc.c  segment.c  serdec.c  shortendec.c  sierravmd.c  siff.c  smacker.c  smjpeg.c  smjpegdec.c  smjpegenc.c  smjpeg.h  smoothstreamingenc.c  smush.c  sol.c  soxdec.c  soxenc.c  sox.h  spdif.c  spdifdec.c  spdifenc.c  spdif.h  srtdec.c  srtenc.c  srtp.c  srtp.h  srtpproto.c  stldec.c  subfile.c  subtitles.c  subtitles.h  subviewer1dec.c  subviewerdec.c  supdec.c  supenc.c  svag.c  swf.c  swfdec.c  swfenc.c  swf.h  takdec.c  tcp.c  tedcaptionsdec.c  tee.c  tee\_common.c  tee\_common.h  teeproto.c | ffmpeg-4.3/libavformat |
| audio\_convert.c  audio\_convert.h  audio\_data.c  audio\_data.h  audio\_mix.c  audio\_mix.h  audio\_mix\_matrix.c  avresample.h  avresampleres.rc  dither.c  dither.h  internal.h  libavresample.v  Makefile  options.c  resample.c  resample.h  resample\_template.c | ffmpeg-4.3/libavresample |
| adler32.c  adler32.h  aes.c  aes\_ctr.c  aes\_ctr.h  aes.h  aes\_internal.h  attributes.h  audio\_fifo.c  audio\_fifo.h  avassert.h  avsscanf.c  avstring.c  avstring.h  avutil.h  avutilres.rc  base64.c  base64.h  blowfish.c  blowfish.h  bprint.c  bprint.h  bswap.h  buffer.c  buffer.h  buffer\_internal.h  camellia.c  camellia.h  cast5.c  cast5.h  channel\_layout.c  channel\_layout.h  colorspace.h  color\_utils.c  color\_utils.h  common.h  cpu.c  cpu.h  cpu\_internal.h  crc.c  crc.h  cuda\_check.h  des.c  des.h  dict.c  dict.h  display.c  display.h  dovi\_meta.c  dovi\_meta.h  downmix\_info.c  downmix\_info.h  dynarray.h  encryption\_info.c  encryption\_info.h  error.c  error.h  eval.c  eval.h  ffmath.h  fifo.c  fifo.h  file.c  file.h  file\_open.c  fixed\_dsp.c  fixed\_dsp.h  float\_dsp.c  float\_dsp.h  frame.c  frame.h  hash.c  hash.h  hdr\_dynamic\_metadata.c  hdr\_dynamic\_metadata.h  hmac.c  hmac.h  hwcontext.c  hwcontext\_cuda.c  hwcontext\_cuda.h  hwcontext\_cuda\_internal.h  hwcontext\_d3d11va.c  hwcontext\_d3d11va.h  hwcontext\_drm.c  hwcontext\_drm.h  hwcontext\_dxva2.c  hwcontext\_dxva2.h  hwcontext.h  hwcontext\_internal.h  hwcontext\_mediacodec.c  hwcontext\_mediacodec.h  hwcontext\_opencl.c  hwcontext\_opencl.h  hwcontext\_qsv.c  hwcontext\_qsv.h  hwcontext\_vaapi.c  hwcontext\_vaapi.h  hwcontext\_vdpau.c  hwcontext\_vdpau.h  hwcontext\_videotoolbox.c  hwcontext\_videotoolbox.h  hwcontext\_vulkan.c  hwcontext\_vulkan.h  imgutils.c  imgutils.h  imgutils\_internal.h  integer.c  integer.h  internal.h  intfloat.h  intmath.c  intmath.h  intreadwrite.h  lfg.c  lfg.h  libavutil.v  libm.h  lls.c  lls.h  log2\_tab.c  log.c  log.h  lzo.c  lzo.h | ffmpeg-4.3/libavutil |

## 离线FFMPEG概要设计说明

### 概述

1

#### 简介

FFmpeg是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用LGPL或GPL许可证。它提供了录制、转换以及流化音视频的完整解决方案。

#### 读者对象

本文档的读者对象为离线识别系统的开发人员、测试人员、系统维护人员及接入识别系统的第三方业务人员，通过本文档能够从总体上了解FFMPEG的架构形式及数据流向。

本说明给出FFMPEG的设计说明，包括最终实现的系统必须满足的功能、性能、接口、附属测试工具程序及设计约束等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

#### 术语定义

语音识别（Automatic Speech Recognition），简称（ASR）。

### 2总体设计

#### 总体需求

功能需求：对语音进行转码；

性能需求：转码耗时低；

接口需求：接口参数简单明了，接口中的每个参数都要有实际意义，保证接口调用流程清晰。

系统验证工具：提供系统完整性，功能正确性的验证工具；

#### 运行环境

##### 软件环境

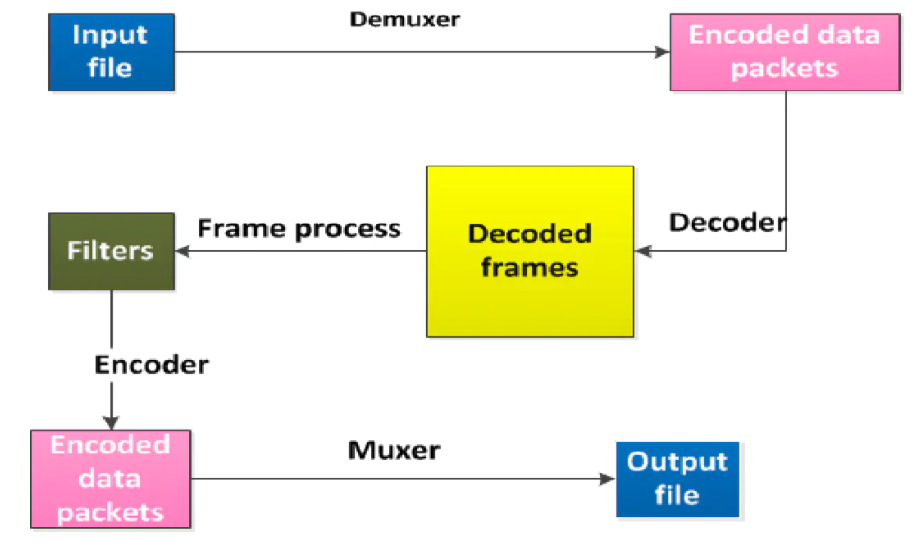
|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.0以上 |
| 数据库 | Redis | 3.2.0 |
| 能力接口 | Tomcat | 9.0.35 |

##### 硬件环境

|  |  |  |
| --- | --- | --- |
| 服务器 | 最低配置 | 推荐配置 |
| redis服务器 | CPU：1  内存：256M | CPU：1  内存：1G |
| 识别服务器 | CPU：4  内存：10G  磁盘：20G | CPU：10  内存：15G  磁盘：30G |

#### 系统总体结构设计

ffmpeg整个流程如下图所示。



1、FFmpeg程序把-i参数指定的若干文件内容读入到内存，按照输入的参数或者程序默认的参数来处理并且把结果写入到若干的文件中。输入和输出文件可以是计算机文件、管道、网络流、捕获设备等。

2、FFmpeg用libavformat包调用解复用器（demuxers）来读取输入文件中被编码的数据包(packets)，如果有多个输入文件，FFmpeg以有效输入流的最小时间戳来同步，

3、然后解码器（decoder）从已编码的数据包中产生未被压缩的帧（frame），在那之后调用可选的过滤器。

4、这些帧被传递到编码器，编码器会产生新的编码包

5、把新的编码包传递给复用器(muxer)处理并且把结果写入到输出文件中。

### 模块设计

#### ffmpeg主要组成部分

1、libavformat：用于各种音视频封装格式的生成和解析，包括获取解码所需信息以生成解码上下文结构和读取音视频帧等功能，包含demuxers和muxer库；

2、libavcodec：用于各种类型声音/图像编解码；

3、libavutil：包含一些公共的工具函数；

4、libswscale：用于视频场景比例缩放、色彩映射转换；

5、libpostproc：用于后期效果处理；

6、ffmpeg：是一个命令行工具，用来对视频文件转换格式，也支持对电视卡实时编码；

7、ffsever：是一个HTTP多媒体实时广播流服务器，支持时光平移；

8、ffplay：是一个简单的播放器，使用ffmpeg 库解析和解码，通过SDL显示；

在这组成部分中，需要熟悉基础概念有

容器(Container)

容器就是一种文件格式，比如flv，mkv等。包含下面5种流以及文件头信息。

流(Stream)

是一种视频数据信息的传输方式，5种流：音频，视频，字幕，附件，数据。

帧(Frame)

帧代表一幅静止的图像，分为I帧，P帧，B帧。

编解码器(Codec)

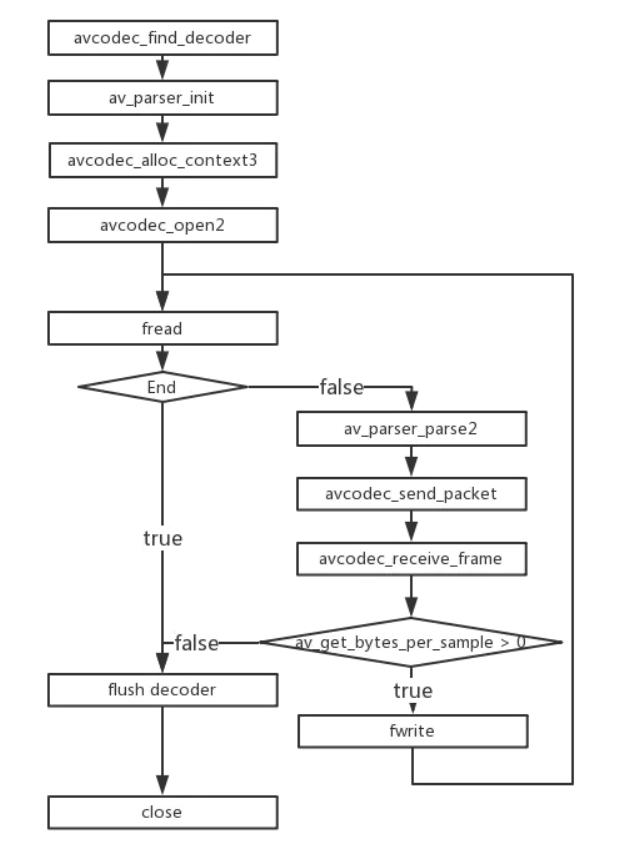
是对视频进行压缩或者解压缩，CODEC =COde （编码） +DECode（解码）

复用/解复用(mux/demux)

把不同的流按照某种容器的规则放入容器，这种行为叫做复用（mux）

把不同的流从某种容器中解析出来，这种行为叫做解复用(demux)

ffmpeg具体处理流程如下：



主要函数介绍：

函数原型：

AVCodec \*avcodec\_find\_decoder(enum AVCodecID id);

参数说明：enum AVCodecID id 解码器的ID

返回值：注册的解码器

功能：

根据指定的AVCodecID查找注册的解码器。

函数原型：

AVCodecParserContext \*av\_parser\_init(int codec\_id)；

参数说明：int codec\_id 解码器的ID

返回值：AVCodecParserContext指针

功能：

初始化AVCodecParserContext。

函数原型：

AVCodecContext \*avcode\_alloc\_context3(const AVCodec \*codec)；

参数说明：AVCodec \*codec 解码器

返回值：AVCodecParserContext指针

功能：

为AVCodecContext分配内存。

函数原型：

int avcodec\_open2(AVCodecContext \*avctx, const AVCodec \*codec, AVDictionary \*\*options);

参数说明：

AVCodecContext \*avctx 需要初始化的AVCodecContext

const AVCodec \*codec 输入的AVCodec

AVDictionary \*\*options 一些选项。例如使用libx264编码的时候，“preset”，“tune”等都可以通过该参数设置

返回值：

int类型。取值及说明如下：

0 ：成功

-1 ：失败

功能：

打开解码器。

函数原型：

int av\_parser\_parse2()；

参数说明：

返回值：

int类型。取值及说明如下：

0 ：成功

<0 ：失败

功能：

解析获得一个Packet。

函数原型：

int avcodec\_send\_packet(AVCodecContext \*avctx, const AVPacket \*avpkt);

参数说明：

AVCodecContext \*avctx 需要初始化的AVCodecContext

const AVPacket \*avpkt 输入的AVPacket

返回值：

int类型。取值及说明如下：

0 ：成功

<0 ：失败

功能：

将AVPacket压缩数据给解码器。

函数原型：

int avcodec\_receive\_frame(AVCodecContext \*avctx, AVFrame \*frame);

参数说明：

AVCodecContext \*avctx 需要初始化的AVCodecContext

AVFrame \*frame 输入的AVFrame

返回值：

int类型。取值及说明如下：

0 ：成功

<0 ：失败

功能：

获取到解码后的AVFrame数据。

函数原型：

int av\_get\_bytes\_per\_sample(enum AVSampleFormat sample\_fmt);

参数说明：

enum AVSampleFormat sample\_fmt sample\_fmt入参

返回值：

int类型。取值及说明如下：

0 ：成功

<0 ：失败

功能：

获取每个sample中的字节数。

关键数据结构说明：

AVCodecParser：用于解析输入的数据流并把它分成一帧一帧的压缩编码数据。比较形象的说法就是把长长的一段连续的数据“切割”成一段段的数据。

### 测试设计

#### 配置文件设计

设置配置文件，方便使用不同控制逻辑，支持不同的功能及结果输出，配置参数实现如下目的：

* 通过参数控制支持特殊功能

配置主要参数如下表：

|  |  |  |
| --- | --- | --- |
| 参数 | 取值 | 说明 |
| inwav | 字符串 | 待转码语音 |
| -ar |  | 设定声音采样率 |
| -ac |  | 设定声道数 |
| -ab |  | 设定声道数 |
| -f |  | 输出格式 |
| outwav | 字符串 | 转码后语音 |

#### 测试功能点

根据设计需求，对服务必须满足的功能点进行验证测试。

|  |  |
| --- | --- |
| 测试功能 | 期望结果 |
| 使用不同的语音进行转码 | 正确配置参数，转码后语音正常。 |

### 可靠性设计

离线系统FFMPEG设计考虑了实际应用场景的复杂性和输入的多样性的情况，内部加入了大部分实际情况中会出现的异常情况的处理方案，保证稳定性和可靠性。

### 扩展性设计

系统本身遵循模块化设计，可以随时添加定制化功能，满足不同应用场景的需要。

### 维护性设计

离线系统为了方便后期的引擎维护，采用标准c/c++函数同时在较低编译器上进行编译

### 易用性设计

系统安装部署时有一键安装部署脚本，只需要执行脚本即可安装,减少操作量。系统启停均有对应的脚本，实现自动启动、停止服务。

## 离线识别引擎FFMPEG接口设计

### 概述

#### 编写目的

本文档主要描述ffmpeg内部接口设计及接口中参数的数据结构，本文档的读者对象为离线识别系统的开发人员、测试人员、系统维护人员及接入识别系统的第三方业务人员，通过本文档能够从总体上了解识别系统内部的数据流向及处理方式。

本说明给出FFMPEG的接口设计说明，包括接口安全、版本兼容性、数据格式、服务端的异常处理等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

### 总体设计

#### 总体需求

接口需求：

1，接口参数简单明了，参数有实际意义；

2，接口功能定义明确；

3，接口的数据特性明确；

4，接口的规格和技术要求明确；

5，接口程序的数据处理逻辑清晰；

### ffmpeg接口设计

ffmpeg包含通用API、解码API、编码API。

#### 通用 API

1.av\_register\_all：

编译配置（–enable、–disable） FFmpeg 的时候，会生成两个文件：config.mk、config.h。config.mk 会实际上就是 makefile 文件需要包含进去的子模块，会作用在编译阶段，帮助开发者编译出正确的库；而 config.h 是作用在运行阶段，这一阶段将确保需要注册哪些容器以及编解码格式到 FFmpeg 框架中。所以该函数的内部实现会先调用 avcodec\_register\_all 来注册所有 config.h 里面开放的编解码器，然后会注册所有的 Muxer 和 Demuxer，最后注册所有的 Protocol。如此，在 config 的过程中，enable、disable 的选项就作用到了运行时，该函数的源码分析涉及的源文件包括 url.c、alformats.c、mux.c、format.c 等文件。

av\_register\_all()的作用是：初始化所有组件，只有调用了该函数，才能使用复用器和编解码器，该接口内部的调用为：

(1).avcodec\_register\_all()，该接口内部执行步骤：

- 注册硬件加速器：REGISTER\_HWACCEL()

- 注册音视频编码器：REGISTER\_ENCODER()

- 注册音视频解码器：REGISTER\_DECODER()

- 打包注册：REGISTER\_ENCDEC()

- 注册解析器：REGISTER\_PARSER()

(2).执行复用器和解复用器的注册：

- 注册复用器：REGISTER\_MUXER()

- 注册解复用器：REGISTER\_DEMUXER()

- 打包注册：REGISTER\_MUXDEMUX()

2.av\_find\_codec：

在 avcodec\_register\_all 函数里面已经把编码器和解码器都存放到了一个链表中，因此都是从该链表进行遍历查找

3.avcodec\_open2：

打开编解码其的时候就会用到该函数，参数有三个，第一个是 AVCodecContext，如果想要传入私有参数，比如 preset、tune、profile，则可以为设置到 priv\_data 参数中。具体到函数实现时，它会找到对应的实现文件，比如，如果打开的是 libx264 编码器，那么实际上的 Codec 为 libx264.c 中的 ff\_libx264\_encoder，Codec 的生命周期方法就会委托给该结构体对应的函数指针所指向的函数。open 对应的就是 init 函数指针所指向的函数，该函数会调用具体的编码库的 API，并以对应的 AVCodecContext 中的 priv\_data 来填充对应第三方库所需要的私有参数。

4.av\_codec\_close：

和 open 类似，找到对应的实现文件中的 close 函数指针所指向的函数，然后该函数会调用对应第三方库的 API 来关闭掉对应编码库。其实 FFmpeg 所做的事情就是透明化所有的编码库，用自己的封装来为开发者提供统一的接口，开发者只需要在打开编解码库时指定编解码器的 ID 即可，之后编码、解码、关闭资源都会找到对应的实现文件去做具体的事情。

#### 解码API

1.avformat\_open\_input

该函数会根据提供的的文件路径判断文件的格式，继而决定使用哪一个 Demuxer。比如，如果是 flv 文件，那么 Demuxer 就会使用对应的 ff\_flv\_demuxer，之后关键的生命周期方法 read\_header、read\_packet、read\_seek、read\_close 都会使用 ff\_flv\_demuxer 中函数指针指定的函数。read\_header 函数会将 AVStream 结构体构造好。

2.avformat\_find\_stream\_info

这个函数非常重要，该方法的作用是将所有 Stream 的MetaData信息填充好，方法内部会先查找对应的解码器，并打开，紧接着利用 Demuxer 中的 read\_packet 函数读取一段数据进行解码，解码数据越多，分析出的流数据就会越准确，本地资源会比较快，网络资源则较慢。该函数提供了几个参数可以控制读取数据的长度，分别为：probe\_size、max\_analyze、fps\_probe\_size，这几个参数的值越小，读取速度越快，信息则相对不够准确。

3.av\_read\_frame

该方法读取出来的数据是 AVPacket，该函数的实现首先会委托到 Demuxer 的 read\_packet 方法，然后在该函数中把未处理完的压缩数据进行缓存处理。

4.avcodec\_decode

如果要解码 H264，会找到 ff\_h264\_decoder，其中最重要的三个声明周期方法为 init、decode、close。

#### 编码API

1.avformat\_alloc\_output\_context

和avformat\_open\_input类似，该函数最终会找到对应的格式复制给 AVFormatContext中的oformat。

2.avio\_open2

该方法首先调用函数ffurl\_open，构造除URLContext，这个结构体包含了 URLProtocol，接着调用avio\_alloc\_context方法，分配除AVIOContext结构体，并将上一步构造出来的URLProtocol传递进来，然后复制给AVFormatContext的属性。

编码步骤其实是解码的一个逆过程，解码过程中的av\_find\_stream\_info对应到编码就是avformat\_new\_stream和av\_format\_write\_header，该步骤会将音频流或视频流的信息填充好，分配出AVStream结构体。read\_header则对应于av\_write\_header，再之后是av\_write\_frame、av\_write\_tailer。

### 代码目录

#### 代码目录结构

ffmpeg-4.3

|--compat

|--configure 配置项

|--doc 说明文档

|--ffbuild

|--fftools

|--libavcodec 编解码库

|--libavdevice 输入输出设备库

|--libavfilter 滤镜库

|--libavformat 文件格式与协议库

|--libavresample

|--libavutil 核心工具库

|--libpostproc

|--libswresample 重采样、转换声道数

|--libswscale 像素格式转换

|--Makefile 编译规则与命令

|--presets

|--README.md 说明

|--tests

|--tools

|--VERSION

#### 主要代码模块说明

##### libavutil

核心工具库，最基础的模块之一，其它模块经常依赖该库做一些基本的音视频处理操作，比如 av\_image\_fill\_arrays（填充原始图像数据到 AVFrame）、av\_image\_get\_buffer\_size（根据图像宽高、格式获取填充该图像需要的字节数）、av\_get\_pix\_fmt\_name（获取像素格式的名称） 等等。

##### libavformat

文件格式和协议库，最重要的模块之一，封装了 Protocol 层和 Demuxer、Muxer 层。常用于读写文件及文件信息，比如 avformat\_write\_header（写文件头）、av\_write\_trailer（写文件尾）、av\_read\_frame （从文件中读取一帧编码后的图像/音频数据）、av\_write\_frame（往文件中写一帧编码后的图像/音频数据）、av\_seek\_frame（给定一个时间戳，移动读指针到对应位置）等等。

##### libavcodec

编解码库，最重要的模块之一。FFmpeg 默认不会添加 libx264、FDK-AAC 等库，但 FFmpeg 可以像一个平台一样，将其它第三方的 Codec 以插件的形式添加进来，并为开发者提供统一的接口。编解码需要用到的函数基本都在该库中，比如 avcodec\_find\_decoder（找到对应的第三方解码器）、avcodec\_decode\_video2（使用对应的解码器解码一帧图像/音频数据）。

## 离线ffmpeg数据结构设计说明

### 概述

#### 简介

FFmpeg是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用LGPL或[GPL](https://baike.baidu.com/item/GPL/2357903" \t "https://baike.baidu.com/item/ffmpeg/_blank)许可证。它提供了录制、转换以及流化音视频的完整解决方案。

#### 读者对象

本文档是对SOX代码流程和相关技术点的整体说明，因此读者最好满足如下几点条件后再读此文档：

* 有一定的C和C++语言基础

### 数据结构设计

本节主要介绍AVFrame、AVCodec、AVStream结构体定义说明。

#### AVFrame

AVFrame结构体一般用于存储原始数据（即非压缩数据，例如对视频来说是YUV，RGB，对音频来说是PCM），此外还包含了一些相关的信息。

typedef struct AVFrame {

#define AV\_NUM\_DATA\_POINTERS 8

uint8\_t \*data[AV\_NUM\_DATA\_POINTERS];

int linesize[AV\_NUM\_DATA\_POINTERS];

uint8\_t \*\*extended\_data;

int width, height;

int nb\_samples;

int format;

int key\_frame;

enum AVPictureType pict\_type;

AVRational sample\_aspect\_ratio;

int64\_t pts;

#if FF\_API\_PKT\_PTS

attribute\_deprecated

int64\_t pkt\_pts;

#endif

int64\_t pkt\_dts;

int coded\_picture\_number;

int display\_picture\_number;

int quality;

void \*opaque;

#if FF\_API\_ERROR\_FRAME

attribute\_deprecated

uint64\_t error[AV\_NUM\_DATA\_POINTERS];

#endif

int repeat\_pict;

int interlaced\_frame;

int top\_field\_first;

int palette\_has\_changed;

int64\_t reordered\_opaque;

int sample\_rate;

uint64\_t channel\_layout;

AVBufferRef \*buf[AV\_NUM\_DATA\_POINTERS];

AVBufferRef \*\*extended\_buf;

int nb\_extended\_buf;

AVFrameSideData \*\*side\_data;

int nb\_side\_data;

#define AV\_FRAME\_FLAG\_CORRUPT (1 << 0)

#define AV\_FRAME\_FLAG\_DISCARD (1 << 2)

int flags;

enum AVColorRange color\_range;

enum AVColorPrimaries color\_primaries;

enum AVColorTransferCharacteristic color\_trc;

enum AVColorSpace colorspace;

enum AVChromaLocation chroma\_location;

int64\_t best\_effort\_timestamp;

int64\_t pkt\_pos;

int64\_t pkt\_duration;

AVDictionary \*metadata;

int decode\_error\_flags;

#define FF\_DECODE\_ERROR\_INVALID\_BITSTREAM 1

#define FF\_DECODE\_ERROR\_MISSING\_REFERENCE 2

#define FF\_DECODE\_ERROR\_CONCEALMENT\_ACTIVE 4

#define FF\_DECODE\_ERROR\_DECODE\_SLICES 8

int channels;

int pkt\_size;

#if FF\_API\_FRAME\_QP

attribute\_deprecated

int8\_t \*qscale\_table;

attribute\_deprecated

int qstride;

attribute\_deprecated

int qscale\_type;

attribute\_deprecated

AVBufferRef \*qp\_table\_buf;

#endif

AVBufferRef \*hw\_frames\_ctx;

AVBufferRef \*opaque\_ref;

size\_t crop\_top;

size\_t crop\_bottom;

size\_t crop\_left;

size\_t crop\_right;

AVBufferRef \*private\_ref;

} AVFrame;

重要变量说明：

uint8\_t \*data[AV\_NUM\_DATA\_POINTERS]：解码后原始数据（对视频来说是YUV，RGB，对音频来说是PCM）

int linesize[AV\_NUM\_DATA\_POINTERS]：data中“一行”数据的大小。注意：未必等于图像的宽，一般大于图像的宽。

int width, height：视频帧宽和高（1920x1080,1280x720...）

int nb\_samples：音频的一个AVFrame中可能包含多个音频帧，在此标记包含了几个

int format：解码后原始数据类型（YUV420，YUV422，RGB24...）

int key\_frame：是否是关键帧

enum AVPictureType pict\_type：帧类型（I,B,P...）

AVRational sample\_aspect\_ratio：宽高比（16:9，4:3...）

int64\_t pts：显示时间戳

int coded\_picture\_number：编码帧序号

int display\_picture\_number：显示帧序号

int8\_t \*qscale\_table：QP表

uint8\_t \*mbskip\_table：跳过宏块表

int16\_t (\*motion\_val[2])[2]：运动矢量表

uint32\_t \*mb\_type：宏块类型表

short \*dct\_coeff：DCT系数，这个没有提取过

int8\_t \*ref\_index[2]：运动估计参考帧列表（貌似H.264这种比较新的标准才会涉及到多参考帧）

int interlaced\_frame：是否是隔行扫描

uint8\_t motion\_subsample\_log2：一个宏块中的运动矢量采样个数，取log的

#### AVCodec

AVCodec是存储编解码器信息的结构体。

typedef struct AVCodec {

const char \*name;

const char \*long\_name;

enum AVMediaType type;

enum AVCodecID id;

int capabilities;

const AVRational \*supported\_framerates; ///< array of supported framerates, or NULL if any, array is terminated by {0,0}

const enum AVPixelFormat \*pix\_fmts; ///< array of supported pixel formats, or NULL if unknown, array is terminated by -1

const int \*supported\_samplerates; ///< array of supported audio samplerates, or NULL if unknown, array is terminated by 0

const enum AVSampleFormat \*sample\_fmts; ///< array of supported sample formats, or NULL if unknown, array is terminated by -1

const uint64\_t \*channel\_layouts; ///< array of support channel layouts, or NULL if unknown. array is terminated by 0

uint8\_t max\_lowres; ///< maximum value for lowres supported by the decoder

const AVClass \*priv\_class; ///< AVClass for the private context

const AVProfile \*profiles; ///< array of recognized profiles, or NULL if unknown, array is terminated by {FF\_PROFILE\_UNKNOWN}

const char \*wrapper\_name;

int priv\_data\_size;

struct AVCodec \*next;

int (\*update\_thread\_context)(struct AVCodecContext \*dst, const struct AVCodecContext \*src);

const AVCodecDefault \*defaults;

void (\*init\_static\_data)(struct AVCodec \*codec);

int (\*init)(struct AVCodecContext \*);

int (\*encode\_sub)(struct AVCodecContext \*, uint8\_t \*buf, int buf\_size,

const struct AVSubtitle \*sub);

int (\*encode2)(struct AVCodecContext \*avctx, struct AVPacket \*avpkt,

const struct AVFrame \*frame, int \*got\_packet\_ptr);

int (\*decode)(struct AVCodecContext \*, void \*outdata, int \*outdata\_size, struct AVPacket \*avpkt);

int (\*close)(struct AVCodecContext \*);

int (\*send\_frame)(struct AVCodecContext \*avctx, const struct AVFrame \*frame);

int (\*receive\_packet)(struct AVCodecContext \*avctx, struct AVPacket \*avpkt);

int (\*receive\_frame)(struct AVCodecContext \*avctx, struct AVFrame \*frame);

void (\*flush)(struct AVCodecContext \*);

int caps\_internal;

const char \*bsfs;

const struct AVCodecHWConfigInternal \*\*hw\_configs;

const uint32\_t \*codec\_tags;

} AVCodec;

重要变量说明：

const char \*name：编解码器的名字，比较短

const char \*long\_name：编解码器的名字，全称，比较长

enum AVMediaType type：指明了类型，是视频，音频，还是字幕

enum AVCodecID id：ID，不重复

const AVRational \*supported\_framerates：支持的帧率（仅视频）

const enum AVPixelFormat \*pix\_fmts：支持的像素格式（仅视频）

const int \*supported\_samplerates：支持的采样率（仅音频）

const enum AVSampleFormat \*sample\_fmts：支持的采样格式（仅音频）

const uint64\_t \*channel\_layouts：支持的声道数（仅音频）

int priv\_data\_size：私有数据的大小

#### AVStream

AVStream是存储每一个视频/音频流信息的结构体。

typedef struct AVStream {

int index; /\*\*< stream index in AVFormatContext \*/

int id;

#if FF\_API\_LAVF\_AVCTX

attribute\_deprecated

AVCodecContext \*codec;

#endif

void \*priv\_data;

AVRational time\_base;

int64\_t start\_time;

int64\_t duration;

int64\_t nb\_frames; ///< number of frames in this stream if known or 0

int disposition; /\*\*< AV\_DISPOSITION\_\* bit field \*/

enum AVDiscard discard; ///< Selects which packets can be discarded at will and do not need to be demuxed.

AVRational sample\_aspect\_ratio;

AVDictionary \*metadata;

AVRational avg\_frame\_rate;

AVPacket attached\_pic;

AVPacketSideData \*side\_data;

int nb\_side\_data;

int event\_flags;

#define AVSTREAM\_EVENT\_FLAG\_METADATA\_UPDATED 0x0001 ///< The call resulted in updated metadata.

AVRational r\_frame\_rate;

#if FF\_API\_LAVF\_FFSERVER

attribute\_deprecated

char \*recommended\_encoder\_configuration;

#endif

AVCodecParameters \*codecpar;

#define MAX\_STD\_TIMEBASES (30\*12+30+3+6)/

struct {

int64\_t last\_dts;

int64\_t duration\_gcd;

int duration\_count;

int64\_t rfps\_duration\_sum;

double (\*duration\_error)[2][MAX\_STD\_TIMEBASES];

int64\_t codec\_info\_duration;

int64\_t codec\_info\_duration\_fields;

int frame\_delay\_evidence;

int found\_decoder;

int64\_t last\_duration;

int64\_t fps\_first\_dts;

int fps\_first\_dts\_idx;

int64\_t fps\_last\_dts;

int fps\_last\_dts\_idx;

} \*info;

int pts\_wrap\_bits; /\*\*< number of bits in pts (used for wrapping control) \*/

int64\_t first\_dts;

int64\_t cur\_dts;

int64\_t last\_IP\_pts;

int last\_IP\_duration;

int probe\_packets;

int codec\_info\_nb\_frames;

enum AVStreamParseType need\_parsing;

struct AVCodecParserContext \*parser;

struct AVPacketList \*last\_in\_packet\_buffer;

AVProbeData probe\_data;

#define MAX\_REORDER\_DELAY 16

int64\_t pts\_buffer[MAX\_REORDER\_DELAY+1];

AVIndexEntry \*index\_entries; /\*\*< Only used if the format does not support seeking natively. \*/

int nb\_index\_entries;

unsigned int index\_entries\_allocated\_size;

int stream\_identifier;

int program\_num;

int pmt\_version;

int pmt\_stream\_idx;

int64\_t interleaver\_chunk\_size;

int64\_t interleaver\_chunk\_duration;

int request\_probe;

int skip\_to\_keyframe;

int skip\_samples;

int64\_t start\_skip\_samples;

int64\_t first\_discard\_sample;

int64\_t last\_discard\_sample;

int nb\_decoded\_frames;

int64\_t mux\_ts\_offset;

int64\_t pts\_wrap\_reference;

int pts\_wrap\_behavior;

int update\_initial\_durations\_done;

int64\_t pts\_reorder\_error[MAX\_REORDER\_DELAY+1];

uint8\_t pts\_reorder\_error\_count[MAX\_REORDER\_DELAY+1];

int64\_t last\_dts\_for\_order\_check;

uint8\_t dts\_ordered;

uint8\_t dts\_misordered;

int inject\_global\_side\_data;

AVRational display\_aspect\_ratio;

AVStreamInternal \*internal;

} AVStream;

重要变量说明：

int index：标识该视频/音频流

AVCodecContext \*codec：指向该视频/音频流的AVCodecContext（它们是一一对应的关系）

AVRational time\_base：时基。通过该值可以把PTS，DTS转化为真正的时间。

int64\_t duration：该视频/音频流长度

AVDictionary \*metadata：元数据信息

AVRational avg\_frame\_rate：帧率（注：对视频来说，这个挺重要的）

AVPacket attached\_pic：附带的图片。

## 离线FFMPEG开发测试环境说明

### 编译环境

#### 硬件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 最低配置 | 推荐配置 |
| CPU | 4C | 16 |
| 内存 | 8G | 32G |
| 磁盘 | 64G | 128G |

#### 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.x |
| 数据库 | Redis | 3.2.0 |
| Mysql | 5.7.30 |
| 能力接口 | Tomcat | 9.0.35 |
| GCC | GCC | 4.8.5 |
| G++ | G++ | 4.8.5 |
| GUN | make | 3.82 |
| ffmpeg源码 | ffmpeg-4.3.tar.gz | 4.3 |

#### 编译方式

编译安装

tar -xvjf ffmpeg-4.3.tar.gz

cd ffmpeg-4.3

./configure --disable-yasm --enable-shared --prefix=~/local/ffmpeg

make -j

make install

配置环境变量

echo "export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:$ffmpeg\_dir/lib/">>~/.bash\_profile

echo "export LIBRARY\_PATH=\$LIBRARY\_PATH:$ffmpeg\_dir/lib/">>~/.bash\_profile

echo "export PATH=\$PATH:$ffmpeg\_dir/bin/">>~/.bash\_profile

echo "export C\_INCLUDE\_PATH=\$C\_INCLUDE\_PATH:$ffmpeg\_dir/include/" >> ~/.bash\_profile

echo "export CPLUS\_INCLUDE\_PATH=\$CPLUS\_INCLUDE\_PATH:$ffmpeg\_dir/include/" >> ~/.bash\_profile

echo "export PKG\_CONFIG\_PATH=\$PKG\_CONFIG\_PATH:$ffmpeg\_dir/lib/pkgconfig/" >> ~/.bash\_profile

echo "export LD\_LIBRARY\_PATH=\$LD\_LIBRARY\_PATH:$ffmpeg\_dir/lib/">>~/.bashrc

echo "export LIBRARY\_PATH=\$LIBRARY\_PATH:$ffmpeg\_dir/lib/">>~/.bashrc

echo "export PATH=\$PATH:$ffmpeg\_dir/bin/">>~/.bashrc

echo "export C\_INCLUDE\_PATH=\$C\_INCLUDE\_PATH:$ffmpeg\_dir/include/">>~/.bashrc

echo "export CPLUS\_INCLUDE\_PATH=\$CPLUS\_INCLUDE\_PATH:$ffmpeg\_dir/include/" >> ~/.bashrc

echo "export PKG\_CONFIG\_PATH=\$PKG\_CONFIG\_PATH:$ffmpeg\_dir/lib/pkgconfig/" >> ~/.bashrc

#### 输出结果

编译输出四个目录

bin 可执行文件ffmpeg

include 头文件

lib lib\*.so动态库

### 测试环境部署

#### 系统安装

如上1.3方式已经编译好FFMPEG,无需在安装部署；

#### 运行环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.x |
| GCC | GCC | 4.8.5 |
| G++ | G++ | 4.8.5 |
| GUN | make | 3.82 |

### 系统测试

#### 测试工具

ffmpeg可执行文件

#### 测试脚本

无

#### 测试数据

1.mp3单声道语音。

#### 测试方法

执行命令：ffmpeg -i 1.mp3 -f wav 1.wav

#### 测试结果

生成wav格式语音。

# 语音格式转换-SOX

## 离线识别引擎SOX源代码清单

### 概述

#### 编写目的

文档编写目的是为代码维护者能清楚了解代码模块的具体功能及分布位置，本文档的读者对象为开发人员和测试人员。

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

### 运行环境

#### 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.0以上 |
| 数据库 | Redis | 3.2.0 |
| Mysql | 5.7.6 |
| 能力接口 | Tomcat | 9.0.35 |

#### 硬件环境

|  |  |  |
| --- | --- | --- |
| 服务器 | 最低配置 | 推荐配置 |
| redis服务器 | CPU：1  内存：256M | CPU：1  内存：1G |
| 识别服务器 | CPU：4  内存：10G  磁盘：20G | CPU：10  内存：15G  磁盘：30G |

### 源代码清单

#### 代码清单

|  |  |
| --- | --- |
| 程序名 | 程序路径 |
| 8svx.c  adpcm.c  adpcm.h  adpcms.c  adpcms.h  aifc-fmt.c  aiff.c  aiff-fmt.c  aiff.h  al-fmt.c  alsa.c  amr.h  amr-nb.c  amr-wb.c  ao.c  au.c  avr.c  band.h  bend.c  biquad.c  biquad.h  biquads.c  caf.c  cdr.c  chorus.c  CMakeLists.txt  compand.c  compandt.c  compandt.h  contrast.c  coreaudio.c  cvsd.c  cvsdfilt.h  cvsd-fmt.c  cvsd.h  dat.c  dcshift.c  delay.c  dft\_filter.c  dft\_filter.h  dither.c  dither.h  divide.c  downsample.c  dvms-fmt.c  earwax.c  echo.c  echos.c  effects.c  effects.h  effects\_i.c  effects\_i\_dsp.c  example0.c  example1.c  example2.c  example3.c  example4.c  example5.c  example6.c  f4-fmt.c  f8-fmt.c  fade.c  fap.c  fft4g.c  fft4g.h  fifo.h  fir.c  firfit.c  flac.c  flanger.c  formats.c  formats.h  formats\_i.c  g711.c  g711.h  g721.c  g723\_24.c  g723\_40.c  g72x.c  g72x.h  gain.c  getopt.c  gsm.c  gsrt.c  hcom.c  hilbert.c  htk.c  ignore-warning.h  ima-fmt.c  ima\_rw.c  ima\_rw.h  input.c  ladspa.c  ladspa.h  la-fmt.c  libsox.c  libsox\_i.c  loudness.c  lpc10.c  lu-fmt.c  Makefile.am  Makefile.in  mat4.c  mat5.c  maud.c  mcompand.c  mcompand\_xover.h  monkey.wav  mp3.c  mp3-util.h  noiseprof.c  noisered.c  noisered.h  nulfile.c  optional-fmts.am  opus.c  oss.c  output.c  overdrive.c  pad.c  paf.c  phaser.c  prc.c  pulseaudio.c  pvf.c  rate.c  rate\_filters.h  rate\_half\_fir.h  rate\_poly\_fir0.h  rate\_poly\_fir.h  raw.c  raw-fmt.c  raw.h  remix.c  repeat.c  reverb.c  reverse.c  s1-fmt.c  s2-fmt.c  s3-fmt.c  s4-fmt.c  sd2.c  sf.c  silence.c  sinc.c  skeleff.c  skelform.c  smp.c  sndfile.c  sndio.c  sounder.c  soundtool.c  sox.c  soxconfig.h.cmake  soxconfig.h.in  sox-fmt.c  sox.h  sox\_i.h  soxomp.h  sox\_sample\_test.c  sox\_sample\_test.h  spectrogram.c  speed.c  sphere.c  splice.c  stat.c  stats.c  stretch.c  sunaudio.c  swap.c  synth.c  tempo.c  testall.bat  testall.sh  test-comments  tests.bat  tests.sh  tremolo.c  trim.c  tx16w.c  u1-fmt.c  u2-fmt.c  u3-fmt.c  u4-fmt.c  ul-fmt.c  upsample.c  util.c  util.h  vad.c  voc.c  vol.c  vorbis.c  vox.c  vox-fmt.c  vox.h  w64.c  wav.c  waveaudio.c  wavpack.c  win32-glob.c  win32-glob.h  win32-ltdl.c  win32-ltdl.h  wve.c  xa.c  xi.c  xmalloc.c  xmalloc.h | sox-14.4.2/src |

## 离线SOX概要设计说明

### 概述

#### 简介

Sox项目是由Lance Norskog创立的，后来被众多的开发者逐步完善，现在已经能够支持很多种声音文件格式和声音处理效果。基本上常见的声音格式都能够支持。更加有用的是，Sox能够进行声音滤波、采样频率转换。

SoX是Sound eXchange的简称，当ffmpeg不支持某些语音格式的转码时，负责对这些特殊的语音进行转码。

SoX可以读写最流行格式的音频文件，并且可以对音频进行任意操作。它可以将多个输入源合成音频。在许多系统，它可以作为一个大众的音频播放器或者多轨录音机。它也可以进行少量的将输入文件切割成多个文件。

#### 读者对象

本文档的读者对象为离线识别系统的开发人员、测试人员、系统维护人员及接入识别系统的第三方业务人员，通过本文档能够从总体上了解SOX的架构形式及数据流向。

本说明给出FFMPEG的设计说明，包括最终实现的系统必须满足的功能、性能、接口、附属测试工具程序及设计约束等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

#### 术语定义

语音识别（Automatic Speech Recognition），简称（ASR）。

### 总体设计

#### 总体需求

功能需求：对语音进行转码；

性能需求：转码耗时低；

接口需求：接口参数简单明了，接口中的每个参数都要有实际意义，保证接口调用流程清晰。

系统验证工具：提供系统完整性，功能正确性的验证工具；

#### 运行环境

##### 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.0以上 |
| 数据库 | Redis | 3.2.0 |
| 能力接口 | Tomcat | 9.0.35 |

##### 硬件环境

|  |  |  |
| --- | --- | --- |
| 服务器 | 最低配置 | 推荐配置 |
| redis服务器 | CPU：1  内存：256M | CPU：1  内存：1G |
| 识别服务器 | CPU：4  内存：10G  磁盘：20G | CPU：10  内存：15G  磁盘：30G |

#### 系统总体结构设计

SoX 可以读取和写入常见格式的音频文件，并在此过程中选择性的加入一些声音效果。它可以组合多个输入源及合成音效，在许多系统上也可以作为音频播放器或多轨录音机使用。sox整个流程如下图所示。



### 模块设计

#### sox基本使用

1、libavformat获取音频文件的元数据：

soxi 或 sox --i 命令可以通过分析音频文件的文件头，获取其元数据（如通道数、采样率、编码等）。

soxi Faded.wav

Input File : 'Faded.wav'

Channels : 2

Sample Rate : 44100

Precision : 16-bit

Duration : 00:03:32.63 = 9376836 samples = 15947 CDDA sectors

File Size : 37.5M

Bit Rate : 1.41M

Sample Encoding: 16-bit Signed Integer PCM

1. 获取音频的统计信息：

可以使用 sox <inputfile> -n stat 命令获取某音频文件的统计信息。示例如下：

sox Faded.wav -n stat

Samples read: 18753672

Length (seconds): 212.626667

Scaled by: 2147483647.0

Maximum amplitude: 0.977417

Minimum amplitude: -0.977478

Midline amplitude: -0.000031

Mean norm: 0.229415

Mean amplitude: -0.000006

RMS amplitude: 0.302594

Maximum delta: 1.765564

Minimum delta: 0.000000

Mean delta: 0.202369

RMS delta: 0.273320

Rough frequency: 6339

Volume adjustment: 1.023

1. 播放与录制：

play 和 rec 命令提供了最基本的播放和录制功能。  
 播放：$ play existing-file.wav  
 录制：$ rec new-file.wav

上述命令等同于 sox 命令的如下形式：  
 $ sox existing-file.wav −d（播放）和 sox −d new-file.wav（录制）  
 其中 -d 选项用于指定播放或录制时使用的音频设备，不指定时则表示使用默认设备。

1. 音频格式转换：

文件格式类型

对于音频数据格式的描述，主要通过以下 4 种属性：

采样率（sample rate）：指声音由模拟信号转换成数字信号的过程中，每秒从连续信号中提取的用于组成离散信号的样本个数。

音频CD所用的采样率为 44100 Hz，数字音频磁带和许多计算机系统使用 48000 Hz，专业级音频系统通常使用 96000 Hz。

采样大小（sample size 或 Precision）：音频采样时用于存储每个样本的数据位数（bits）。

编码格式（data encoding）：即每个音频样本的表示（即“编码”）方式。常用的编码类型包括 floating-point、μ-law、ADPCM、singed-integer PCM、MP3 和 FLAC 等。

通道（channel）：即文件中包含的音频通道的数量。其中单声道（mono）和双声道（stereo）是最常见的两种，“环绕声”音频（Surround sound）通常包含六个或更多声道。

此外，音频文件还使用比特率（Bit Rate）表示一个单位时间内编码音频信号占用的存储空间大小， 它的数值一般取决于所有的上述四个参数。

MP3 编码的立体声音乐通常具有 128-196kbps 的比特率， FLAC 编码的立体声音乐通常具有 550-760kbps 的比特率。

格式转换

形式最简单的 sox 命令即使用两个文件名作为参数，如：

sox Faded.wav Faded.mp3 ：将 Faded.wav 文件的格式由 wav 转为 mp3

上述命令执行时，SoX 会先从 Faded.wav 文件中读取音频数据，再将其输出到 Faded.mp3 文件中。而 SoX 程序会根据参数中文件名的后缀推断出相应的格式，并在复制音频数据的过程中自动进行转码。

SoX 可以处理 self-describing 和 raw 格式的音频文件。

self-describing 格式（如 WAV、FLAC、MP3）的文件包含一个用于描述信号和编码属性的文件头，而 raw 或 headless 格式的音频则不包含这些信息。

所以当 raw 格式的音频作为输入文件时，需要在 sox 命令的格式选项里指定其信号和编码属性。

常用的音频格式选项：

选项 描述

-b, --bits BITS 每个编码样本占用的数据位数

-c, --channels CHANNELS 音频文件包含的通道数

-e, --encoding ENCODING 音频文件的编码类型

-r, --rate RATE 音频文件的采样率

-t, --type FILE-TYPE 音频文件的文件类型

上述选项适用于输入或输出文件，主要用于说明 raw（或 headless）文件作为输入时的格式信息，或格式转换时指定输出文件的具体参数。

sox −r 48k −e float −b 32 −c 2 input.raw output.wav

将某个特定的 raw 格式的音频文件转换为 wav 格式。

sox Faded.wav Faded.raw

将音频文件 Faded.wav 转为 raw 格式。

play -r 44800 -b 16 -e signed-integer -c 2 Faded.raw

播放 raw 格式的音频文件。

sox Faded.wav -c 1 Faded-mono.wav

将 Faded.wav 文件转换成单声道（-c 1）后输出。

#### sox转码流程

Sox具体处理流程如下：



关键函数说明：

sox\_init()：初始化全局参数

sox\_open\_mem\_read：打开文件；

sox\_open\_men\_write：打开输出文件；

sox\_create\_effects\_chain：构建一个效果器链；

sox\_create\_effect：创建一个简单的效果输入文件；

sox\_add\_effect：添加效果器到效果器链；

sox\_flow\_effects：效果器运行；

sox\_delete\_effects\_chain：释放资源；

sox\_close：关闭文件；

sox\_quit：退出；

主要函数介绍：

函数原型：

int LSX\_API sox\_init(void);

参数说明：

返回值：

int类型。取值及说明如下：

0 ：成功

非0 ：失败

功能：

初始化全局参数。

函数原型：

LSX\_RETURN\_OPT sox\_format\_t \* LSX\_API sox\_open\_mem\_read(

LSX\_PARAM\_IN\_BYTECOUNT(buffer\_size) void \* buffer,

size\_t buffer\_size,

LSX\_PARAM\_IN\_OPT sox\_signalinfo\_t const \* signal,

LSX\_PARAM\_IN\_OPT sox\_encodinginfo\_t const \* encoding,

LSX\_PARAM\_IN\_OPT\_Z char const \* filetype

);

参数说明：

LSX\_PARAM\_IN\_BYTECOUNT(buffer\_size) void \* buffer 数据指针

size\_t buffer\_size, 数据大小

LSX\_PARAM\_IN\_OPT sox\_signalinfo\_t const \* signal 信号量

LSX\_PARAM\_IN\_OPT sox\_encodinginfo\_t const \* encoding 编码信息

LSX\_PARAM\_IN\_OPT\_Z char const \* filetype 文件类型

返回值：sox\_format\_t指针

功能：

打开输入文件。

函数原型：

LSX\_RETURN\_OPT sox\_format\_t \* LSX\_API sox\_open\_mem\_write(

LSX\_PARAM\_OUT\_BYTECAP(buffer\_size) void \* buffer,

LSX\_PARAM\_IN size\_t buffer\_size,

LSX\_PARAM\_IN sox\_signalinfo\_t const \* signal,

LSX\_PARAM\_IN\_OPT sox\_encodinginfo\_t const \* encoding,

LSX\_PARAM\_IN\_OPT\_Z char const \* filetype

LSX\_PARAM\_IN\_OPT sox\_oob\_t const \* oob

);

参数说明：

LSX\_PARAM\_OUT\_BYTECAP(buffer\_size) void \* buffer 数据指针

size\_t buffer\_size, 数据大小

LSX\_PARAM\_IN\_OPT sox\_signalinfo\_t const \* signal 信号量

LSX\_PARAM\_IN\_OPT sox\_encodinginfo\_t const \* encoding 编码信息

LSX\_PARAM\_IN\_OPT\_Z char const \* filetype 文件类型

LSX\_PARAM\_IN\_OPT sox\_oob\_t const \* oob 输出数据指针

返回值：sox\_format\_t指针

功能：

打开输出文件。

函数原型：

LSX\_RETURN\_OPT sox\_effects\_chain\_t \* LSX\_API sox\_create\_effects\_chain(

LSX\_PARAM\_IN sox\_encodinginfo\_t const \* in\_enc,

LSX\_PARAM\_IN sox\_encodinginfo\_t const \* out\_enc

);

参数说明：

LSX\_PARAM\_IN sox\_encodinginfo\_t const \* in\_enc 效果器指针

LSX\_PARAM\_IN sox\_encodinginfo\_t const \* out\_enc效果器结果指针

返回值：sox\_effects\_chain\_t指针

功能：

构建效果器链。

函数原型：

LSX\_RETURN\_OPT sox\_effect\_t \* LSX\_API sox\_create\_effect(

LSX\_PARAM\_IN sox\_effect\_handler\_t const \* eh

);

参数说明：

LSX\_PARAM\_IN sox\_effect\_handler\_t const \* eh 效果器信息指针

返回值：sox\_effect\_t指针

功能：

构建简单效果器。

函数原型：

int LSX\_API sox\_add\_effect(

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \* chain,

LSX\_PARAM\_INOUT sox\_effect\_t \* effp,

LSX\_PARAM\_INOUT sox\_signalinfo\_t \* in,

LSX\_PARAM\_IN sox\_signalinfo\_t const \* out

);

参数说明：

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \* chain 效果器链指针

LSX\_PARAM\_INOUT sox\_effect\_t \* effp 效果器指针

LSX\_PARAM\_INOUT sox\_signalinfo\_t \* in Input format

LSX\_PARAM\_IN sox\_signalinfo\_t const \* out Output format

返回值：

int类型。取值及说明如下：

0 ：成功

非0 ：失败

功能：

添加效果器到效果器链。

函数原型：

int LSX\_API sox\_flow\_effects(

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \* chain,

LSX\_PARAM\_IN\_OPT sox\_flow\_effects\_callback callback,

LSX\_PARAM\_IN\_OPT void \* client\_data

);

参数说明：

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \* chain 运行效果器链指针

LSX\_PARAM\_IN\_OPT sox\_flow\_effects\_callback callback 回调

LSX\_PARAM\_IN\_OPT void \* client\_data 回调结果数据

返回值：

int类型。取值及说明如下：

0 ：成功

非0 ：失败

功能：

运行效果器。

函数原型：

void LSX\_API sox\_delete\_effects\_chain(

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \*ecp

);

参数说明：

LSX\_PARAM\_INOUT sox\_effects\_chain\_t \*ecp 效果器链指针

返回值：

功能：

释放效果器链资源。

函数原型：

int LSX\_API sox\_close(

LSX\_PARAM\_INOUT sox\_format\_t \* ft

);

参数说明：

LSX\_PARAM\_INOUT sox\_format\_t \* ft 效果器指针

返回值：

int类型。取值及说明如下：

0 ：成功

非0 ：失败

功能：

关闭效果器。

函数原型：

int LSX\_API sox\_quit(void);

参数说明：

返回值：

int类型。取值及说明如下：

0 ：成功

非0 ：失败

功能：

退出。

### 测试设计

#### 配置文件设计

设置配置文件，方便使用不同控制逻辑，支持不同的功能及结果输出，配置参数实现如下目的：

* 通过参数控制支持特殊功能

配置主要参数如下表：

|  |  |  |
| --- | --- | --- |
| 参数 | 取值 | 说明 |
| inwav | 字符串 | 待转码语音 |
| -b |  | 每个编码样本占用的数据位数 |
| -c |  | 音频文件包含的通道数 |
| -r |  | 音频文件的采样率 |
| outwav | 字符串 | 转码后语音 |

#### 测试功能点

根据设计需求，对服务必须满足的功能点进行验证测试。

|  |  |
| --- | --- |
| 测试功能 | 期望结果 |
| 使用不同的语音进行转码 | 正确配置参数，转码后语音正常。 |

### 可靠性设计

离线系统SOX设计考虑了实际应用场景的复杂性和输入的多样性的情况，内部加入了大部分实际情况中会出现的异常情况的处理方案，保证稳定性和可靠性。

### 扩展性设计

系统本身遵循模块化设计，可以随时添加定制化功能，满足不同应用场景的需要。

### 维护性设计

离线系统为了方便后期的引擎维护，采用标准c/c++函数同时在较低编译器上进行编译

### 易用性设计

系统安装部署时有一键安装部署脚本，只需要执行脚本即可安装,减少操作量。系统启停均有对应的脚本，实现自动启动、停止服务。

## 离线识别引擎SOX接口设计

### 概述

#### 编写目的

本文档主要描述sox内部接口设计及接口中参数的数据结构，本文档的读者对象为离线识别系统的开发人员、测试人员、系统维护人员及接入识别系统的第三方业务人员，通过本文档能够从总体上了解识别系统内部的数据流向及处理方式。

本说明给出SOX的接口设计说明，包括接口安全、版本兼容性、数据格式、服务端的异常处理等。

目的在于：

* 为开发人员提供依据；
* 为代码修改、维护提供条件；

### 总体设计

#### 总体需求

接口需求：

1，接口参数简单明了，参数有实际意义；

2，接口功能定义明确；

3，接口的数据特性明确；

4，接口的规格和技术要求明确；

5，接口程序的数据处理逻辑清晰；

### SOX接口设计

本节主要介绍soxAPI使用。

SoX 处理音频的基本流程如下：



#### 通用 API

1.sox\_init()：

在使用SOX库之前，必须初始化整个库的全局参数。函数返回一个整数，如果返回的是SOX\_SUCCESS这个枚举值，代码初始化成功；

2.sox\_init()：

退出SOX在。

3.sox\_open\_read()：

初始化输入文件。

4.sox\_open\_write()：

初始化输出文件。

5.sox\_create\_chain()：

构造效果器链。

1. sox\_create\_cffect()

构建一个输入数据的效果器。

1. sox\_effect\_options()

输入文件配置到这个效果器。

1. sox\_add\_effect()

效果器增加到效果器链。

1. sox\_flow\_effects()

运行效果器链。

### 代码目录

#### 代码目录结构

sox-14.4.2

├── 8svx.c

├── adpcm.c

├── adpcm.h

├── adpcms.c

├── adpcms.h

├── aifc-fmt.c

├── aiff.c

├── aiff-fmt.c

├── aiff.h

├── al-fmt.c

├── alsa.c

├── amr.h

├── amr-nb.c

├── amr-wb.c

├── ao.c

├── au.c

├── avr.c

├── band.h

├── bend.c

├── biquad.c

├── biquad.h

├── biquads.c

├── caf.c

├── cdr.c

├── chorus.c

├── CMakeLists.txt

├── compand.c

├── compandt.c

├── compandt.h

├── contrast.c

├── coreaudio.c

├── cvsd.c

├── cvsdfilt.h

├── cvsd-fmt.c

├── cvsd.h

├── dat.c

├── dcshift.c

├── delay.c

├── dft\_filter.c

├── dft\_filter.h

├── dither.c

├── dither.h

├── divide.c

├── downsample.c

├── dvms-fmt.c

├── earwax.c

├── echo.c

├── echos.c

├── effects.c

├── effects.h

├── effects\_i.c

├── effects\_i\_dsp.c

├── example0.c

├── example1.c

├── example2.c

├── example3.c

├── example4.c

├── example5.c

├── example6.c

├── f4-fmt.c

├── f8-fmt.c

├── fade.c

├── fap.c

├── fft4g.c

├── fft4g.h

├── fifo.h

├── fir.c

├── firfit.c

├── flac.c

├── flanger.c

├── formats.c

├── formats.h

├── formats\_i.c

├── g711.c

├── g711.h

├── g721.c

├── g723\_24.c

├── g723\_40.c

├── g72x.c

├── g72x.h

├── gain.c

├── getopt.c

├── gsm.c

├── gsrt.c

├── hcom.c

├── hilbert.c

├── htk.c

├── ignore-warning.h

├── ima-fmt.c

├── ima\_rw.c

├── ima\_rw.h

├── input.c

├── ladspa.c

├── ladspa.h

├── la-fmt.c

├── libsox.c

├── libsox\_i.c

├── loudness.c

├── lpc10.c

├── lu-fmt.c

├── Makefile.am

├── Makefile.in

├── mat4.c

├── mat5.c

├── maud.c

├── mcompand.c

├── mcompand\_xover.h

├── monkey.wav

├── mp3.c

├── mp3-util.h

├── noiseprof.c

├── noisered.c

├── noisered.h

├── nulfile.c

├── optional-fmts.am

├── opus.c

├── oss.c

├── output.c

├── overdrive.c

├── pad.c

├── paf.c

├── phaser.c

├── prc.c

├── pulseaudio.c

├── pvf.c

├── rate.c

├── rate\_filters.h

├── rate\_half\_fir.h

├── rate\_poly\_fir0.h

├── rate\_poly\_fir.h

├── raw.c

├── raw-fmt.c

├── raw.h

├── remix.c

├── repeat.c

├── reverb.c

├── reverse.c

├── s1-fmt.c

├── s2-fmt.c

├── s3-fmt.c

├── s4-fmt.c

├── sd2.c

├── sf.c

├── silence.c

├── sinc.c

├── skeleff.c

├── skelform.c

├── smp.c

├── sndfile.c

├── sndio.c

├── sounder.c

├── soundtool.c

├── sox.c

├── soxconfig.h.cmake

├── soxconfig.h.in

├── sox-fmt.c

├── sox.h

├── sox\_i.h

├── soxomp.h

├── sox\_sample\_test.c

├── sox\_sample\_test.h

├── spectrogram.c

├── speed.c

├── sphere.c

├── splice.c

├── stat.c

├── stats.c

├── stretch.c

├── sunaudio.c

├── swap.c

├── synth.c

├── tempo.c

├── testall.bat

├── testall.sh

├── test-comments

├── tests.bat

├── tests.sh

├── tremolo.c

├── trim.c

├── tx16w.c

├── u1-fmt.c

├── u2-fmt.c

├── u3-fmt.c

├── u4-fmt.c

├── ul-fmt.c

├── upsample.c

├── util.c

├── util.h

├── vad.c

├── voc.c

├── vol.c

├── vorbis.c

├── vox.c

├── vox-fmt.c

├── vox.h

├── w64.c

├── wav.c

├── waveaudio.c

├── wavpack.c

├── win32-glob.c

├── win32-glob.h

├── win32-ltdl.c

├── win32-ltdl.h

├── wve.c

├── xa.c

├── xi.c

├── xmalloc.c

└── xmalloc.h

#### 主要代码模块说明

##### 转码模块

g72x.c该文件主要实现g72x格式文件转换。

mp3.c该文件主要实现mp3格式文件转换。

##### 控制模块

sox.c该文件主要实现流程处理函数的实现。

## 离线sox数据结构设计说明

### 概述

#### 简介

Sox项目是由Lance Norskog创立的，后来被众多的开发者逐步完善，现在已经能够支持很多种声音文件格式和声音处理效果。基本上常见的声音格式都能够支持。更加有用的是，Sox能够进行声音滤波、采样频率转换。

Sox是Sound eXchange的简称，当ffmpeg不支持某些语音格式的转码时（如vox、V3格式），负责对这些特殊的语音进行转码。

#### 读者对象

本文档是对SOX代码流程和相关技术点的整体说明，因此读者最好满足如下几点条件后再读此文档：

* 有一定的C和C++语言基础

### 数据结构设计

本节主要介绍sox\_format\_t 、sox\_effect\_handler\_t、sox\_effect\_t结构体定义说明。

#### sox\_format\_t

struct sox\_format\_t {

char \* filename; /\*\*< File name \*/

sox\_signalinfo\_t signal;

sox\_encodinginfo\_t encoding;

char \* filetype; /\*\*< Type of file, as determined by header inspection or libmagic. \*/

sox\_oob\_t oob; /\*\*< comments, instrument info, loop info (out-of-band data) \*/

sox\_bool seekable; /\*\*< Can seek on this file \*/

char mode; /\*\*< Read or write mode ('r' or 'w') \*/

sox\_uint64\_t olength; /\*\*< Samples \* chans written to file \*/

sox\_uint64\_t clips; /\*\*< Incremented if clipping occurs \*/

int sox\_errno; /\*\*< Failure error code \*/

char sox\_errstr[256]; /\*\*< Failure error text \*/

void \* fp; /\*\*< File stream pointer \*/

lsx\_io\_type io\_type; /\*\*< Stores whether this is a file, pipe or URL \*/

sox\_uint64\_t tell\_off; /\*\*< Current offset within file \*/

sox\_uint64\_t data\_start; /\*\*< Offset at which headers end and sound data begins (set by lsx\_check\_read\_params) \*/

sox\_format\_handler\_t handler; /\*\*< Format handler for this file \*/

void \* priv; /\*\*< Format handler's private data area \*/

};

重要变量说明：

char \* filename; 文件名

sox\_signalinfo\_t signal; 信号

sox\_encodinginfo\_t encoding; 编码信息

char \* filetype; 文件类型，由头检查或libmagic确定

sox\_oob\_t oob; 循环信息

sox\_bool seekable; 是否查找

char mode; 模式

sox\_uint64\_t olength; 写入文件长度

sox\_uint64\_t clips; 递增clips

int sox\_errno; 错误码

char sox\_errstr[256]; 错误信息

void \* fp; 文件指针

lsx\_io\_type io\_type; 储存类型

sox\_uint64\_t tell\_off; 文件内当前偏移量

sox\_uint64\_t data\_start; 声音数据开始的偏移量

sox\_format\_handler\_t handler; 文件格式处理方式

void \* priv; 私有数据区域

#### sox\_effect\_handler\_t

struct sox\_effect\_handler\_t {

char const \* name;

char const \* usage;

unsigned int flags;

sox\_effect\_handler\_getopts getopts;

sox\_effect\_handler\_start start;

sox\_effect\_handler\_flow flow;

sox\_effect\_handler\_drain drain;

sox\_effect\_handler\_stop stop;

sox\_effect\_handler\_kill kill;

size\_t priv\_size;

};

重要变量说明：

char const \* name; Effect name

char const \* usage; effect参数接受效果

unsigned int flags; effect标志

sox\_effect\_handler\_getopts getopts; 调用命令解析参数

sox\_effect\_handler\_start start; 初始化效果器

sox\_effect\_handler\_flow flow; 调用处理结果

sox\_effect\_handler\_drain drain; 在输入完成后调用以完成输出

sox\_effect\_handler\_stop stop; 调用关闭effect

sox\_effect\_handler\_kill kill; 调用关闭effect

size\_t priv\_size; 预分配大小

#### sox\_effect\_t

struct sox\_effect\_t {

sox\_effects\_globals\_t \* global\_info;

sox\_signalinfo\_t in\_signal;

sox\_signalinfo\_t out\_signal;

sox\_encodinginfo\_t const \* in\_encoding;

sox\_encodinginfo\_t const \* out\_encoding;

sox\_effect\_handler\_t handler;

sox\_uint64\_t clips;

size\_t flows;

size\_t flow;

void \* priv;

sox\_sample\_t \* obuf;

size\_t obeg;

size\_t oend;

size\_t imin;

};

重要变量说明：

sox\_effects\_globals\_t \* global\_info; effect全局变量信息

sox\_signalinfo\_t in\_signal; 输入信号

sox\_signalinfo\_t out\_signal; 输出信号

sox\_encodinginfo\_t const \* in\_encoding; 输入数据编码信息

sox\_encodinginfo\_t const \* out\_encoding; 输出数据编码信息

sox\_effect\_handler\_t handler; effect句柄

sox\_uint64\_t clips; 递增

size\_t flows; MCHAN值

size\_t flow; flow数量

void \* priv; Effect的私有数据区域

sox\_sample\_t \* obuf; 输出buf

size\_t obeg; 输出缓冲区:有效数据段的开始

size\_t oend; 输出缓冲区:有效数据段的结束

size\_t imin; 最小输入缓冲区

sox\_error\_t一般用于返回错误码。

enum sox\_error\_t {

SOX\_SUCCESS = 0, /\*\*< Function succeeded = 0 \*/

SOX\_EOF = -1, /\*\*< End Of File or other error = -1 \*/

SOX\_EHDR = 2000, /\*\*< Invalid Audio Header = 2000 \*/

SOX\_EFMT, /\*\*< Unsupported data format = 2001 \*/

SOX\_ENOMEM, /\*\*< Can't alloc memory = 2002 \*/

SOX\_EPERM, /\*\*< Operation not permitted = 2003 \*/

SOX\_ENOTSUP, /\*\*< Operation not supported = 2004 \*/

SOX\_EINVAL /\*\*< Invalid argument = 2005 \*/

};

## 离线SOX开发测试环境说明

### 编译环境

#### 硬件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 最低配置 | 推荐配置 |
| CPU | 4C | 16 |
| 内存 | 8G | 32G |
| 磁盘 | 64G | 128G |

#### 软件环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.x |
| 数据库 | Redis | 3.2.0 |
| Mysql | 5.7.30 |
| 能力接口 | Tomcat | 9.0.35 |
| GCC | GCC | 4.8.5 |
| G++ | G++ | 4.8.5 |
| GUN | make | 3.82 |
| sox源代码 | sox-14.4.2 | 14.4.2 |

#### 编译方式

编译安装

tar -xvf sox-14.4.2.tar.gz

cd sox-14.4.2

./configure --prefix=~/local/sox

Make -j && make install

配置环境变量

echo "export LD\_LIBRARY\_PATH=$sox\_dir/lib/:\$LD\_LIBRARY\_PATH">>~/.bash\_profile

echo "export LIBRARY\_PATH=$sox\_dir/lib/:\$LIBRARY\_PATH">>~/.bash\_profile

echo "export PATH=$sox\_dir/bin/:\$PATH">>~/.bash\_profile

echo "export C\_INCLUDE\_PATH=$sox\_dir/include/:\$C\_INCLUDE\_PATH">>~/.bash\_profile

echo "export CPLUS\_INCLUDE\_PATH=$sox\_dir/include/:\$CPLUS\_INCLUDE\_PATH">>~/.bash\_profile

echo "export PKG\_CONFIG\_PATH=$sox\_dir/lib/pkgconfig/:\$PKG\_CONFIG\_PATH">>~/.bash\_profile

echo "export LD\_LIBRARY\_PATH=$sox\_dir/lib/:\$LD\_LIBRARY\_PATH">>~/.bashrc

echo "export LIBRARY\_PATH=$sox\_dir/lib/:\$LIBRARY\_PATH">>~/.bashrc

echo "export PATH=$sox\_dir/bin/:\$PATH">>~/.bashrc

echo "export C\_INCLUDE\_PATH=$sox\_dir/include/:\$C\_INCLUDE\_PATH">>~/.bashrc

echo "export CPLUS\_INCLUDE\_PATH=$sox\_dir/include/:\$CPLUS\_INCLUDE\_PATH">>~/.bashrc

echo "export PKG\_CONFIG\_PATH=$sox\_dir/lib/pkgconfig/:\$PKG\_CONFIG\_PATH">>~/.bashrc

#### 输出结果

编译输出四个目录

bin 可执行文件sox

include 头文件

lib libsox.so动态库

### 测试环境部署

#### 系统安装

如上1.3方式已经编译好SOX,无需在安装部署；

#### 运行环境

|  |  |  |
| --- | --- | --- |
| 分类 | 名称 | 版本 |
| 操作系统 | Centos | 7.x |
| GCC | GCC | 4.8.5 |
| G++ | G++ | 4.8.5 |
| GUN | make | 3.82 |

### 系统测试

#### 测试工具

sox可执行文件

#### 测试脚本

无

#### 测试数据

in.wav单声道语音。

#### 测试方法

执行命令：ffmpeg -i in.wav -b 16 -r 8000 out.wav

#### 测试结果

查看生成out.wav是否为8k16bitpcm。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模块 | 分类 | | 源码 | 简单说明 |
| 场景分割源码 | 离线 | 说话人聚类 | 说话人聚类 | 利用语音识别的分段结果，将不同说话人（坐席和用户）的语音和时间点进行提取 |
| 角色分类 | 角色分类源码+war包+文档 | 利用不同说话人的语音转写文本，利用角色分类模型确定其角色类型（坐席还是用户） |
| 端到端架构语音识别引擎(解码器)源码 | 离线 | 解码器内核 | 离线解码器内核 | 包含端到端声学模型推理计算和语音识别令牌传递，提供解码库实现输入语音特征，输出识别文本。 |
| 引擎封装TBNR | 离线语音识别引擎-总模块 | 语音识别引擎工程前端封装包括：端点检测、特征提取、后处理匹配和多线程封装等子模块。完成离线语音数据进行转写功能。 |
| 在线 | 解码器内核 | 在线解码器内核 | 包含端到端声学模型推理计算和语音识别令牌传递，提供解码库实现输入语音特征流，输出识别文本。 |
| 引擎封装TBNR | 在线语音识别引擎-总模块 | 语音识别引擎工程前端封装包括：端点检测、特征提取、后处理匹配和多线程封装等子模块。完成输入实时输入语音进行实时转写的功能。 |
| 叠音检测源码 | 离线 | 叠音检测 | 叠音检测源码 | 自动检测出用户和坐席同时说话的语音段（叠音），并自动给出其起始点位置。 |
|
| 离线转写结果融合源码 | 离线转写结果融合 | 转写结果融合源码 | 转写结果融合源码 | 基于离线语音识别引擎的结果，增加静音、语速、能量的统计结果，将结果进行合并，输出xml格式结果。 |
| xml格式文件转换工具 | XML格式文件转换Q1格式工具 | 将xml结果格式进行转换，得到Q1.xml格式的结果。 |