// stdafx.h : 标准系统包含文件的包含文件，

// 或是经常使用但不常更改的

// 特定于项目的包含文件

#pragma once

#ifndef \_SECURE\_ATL

#define \_SECURE\_ATL 1

#endif

#ifndef VC\_EXTRALEAN

#define VC\_EXTRALEAN // 从 Windows 头中排除极少使用的资料

#endif

#include "targetver.h"

#define \_ATL\_CSTRING\_EXPLICIT\_CONSTRUCTORS // 某些 CString 构造函数将是显式的

// 关闭 MFC 对某些常见但经常可放心忽略的警告消息的隐藏

#define \_AFX\_ALL\_WARNINGS

#include <afxwin.h> // MFC 核心组件和标准组件

#include <afxext.h> // MFC 扩展

#include <afxdisp.h> // MFC 自动化类

#include "Wav\_File\_Handle.h"

#include "Wav\_File\_Header.h"

#ifndef \_AFX\_NO\_OLE\_SUPPORT

#include <afxdtctl.h> // MFC 对 Internet Explorer 4 公共控件的支持

#endif

#ifndef \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcmn.h> // MFC 对 Windows 公共控件的支持

#endif // \_AFX\_NO\_AFXCMN\_SUPPORT

#include <afxcontrolbars.h> // 功能区和控件条的 MFC 支持

#ifdef \_UNICODE

#if defined \_M\_IX86

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#elif defined \_M\_X64

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#else

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#endif

#endif

// 端点检测波形展示.h : PROJECT\_NAME 应用程序的主头文件

//

#pragma once

#ifndef \_\_AFXWIN\_H\_\_

#error "在包含此文件之前包含“stdafx.h”以生成 PCH 文件"

#endif

#include "resource.h" // 主符号

// C端点检测波形展示App:

// 有关此类的实现，请参阅 端点检测波形展示.cpp

//

class C端点检测波形展示App : public CWinApp

{

public:

C端点检测波形展示App();

// 重写

public:

virtual BOOL InitInstance();

// 实现

DECLARE\_MESSAGE\_MAP()

};

extern C端点检测波形展示App theApp;

// 端点检测波形展示.cpp : 定义应用程序的类行为。

//

#include "stdafx.h"

#include "端点检测波形展示.h"

#include "端点检测波形展示Dlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// C端点检测波形展示App

BEGIN\_MESSAGE\_MAP(C端点检测波形展示App, CWinApp)

ON\_COMMAND(ID\_HELP, &CWinApp::OnHelp)

END\_MESSAGE\_MAP()

// C端点检测波形展示App 构造

C端点检测波形展示App::C端点检测波形展示App()

{

// 支持重新启动管理器

m\_dwRestartManagerSupportFlags = AFX\_RESTART\_MANAGER\_SUPPORT\_RESTART;

// TODO: 在此处添加构造代码，

// 将所有重要的初始化放置在 InitInstance 中

}

// 唯一的一个 C端点检测波形展示App 对象

C端点检测波形展示App theApp;

// C端点检测波形展示App 初始化

BOOL C端点检测波形展示App::InitInstance()

{

// 如果一个运行在 Windows XP 上的应用程序清单指定要

// 使用 ComCtl32.dll 版本 6 或更高版本来启用可视化方式，

//则需要 InitCommonControlsEx()。否则，将无法创建窗口。

INITCOMMONCONTROLSEX InitCtrls;

InitCtrls.dwSize = sizeof(InitCtrls);

// 将它设置为包括所有要在应用程序中使用的

// 公共控件类。

InitCtrls.dwICC = ICC\_WIN95\_CLASSES;

InitCommonControlsEx(&InitCtrls);

CWinApp::InitInstance();

AfxEnableControlContainer();

// 创建 shell 管理器，以防对话框包含

// 任何 shell 树视图控件或 shell 列表视图控件。

CShellManager \*pShellManager = new CShellManager;

// 标准初始化

// 如果未使用这些功能并希望减小

// 最终可执行文件的大小，则应移除下列

// 不需要的特定初始化例程

// 更改用于存储设置的注册表项

// TODO: 应适当修改该字符串，

// 例如修改为公司或组织名

SetRegistryKey(\_T("应用程序向导生成的本地应用程序"));

C端点检测波形展示Dlg dlg;

m\_pMainWnd = &dlg;

INT\_PTR nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: 在此放置处理何时用

// “确定”来关闭对话框的代码

}

else if (nResponse == IDCANCEL)

{

// TODO: 在此放置处理何时用

// “取消”来关闭对话框的代码

}

// 删除上面创建的 shell 管理器。

if (pShellManager != NULL)

{

delete pShellManager;

}

// 由于对话框已关闭，所以将返回 FALSE 以便退出应用程序，

// 而不是启动应用程序的消息泵。

return FALSE;

}

// Microsoft Visual C++ generated resource script.

//

#include "resource.h"

#define APSTUDIO\_READONLY\_SYMBOLS

/////////////////////////////////////////////////////////////////////////////

//

// Generated from the TEXTINCLUDE 2 resource.

//

#ifndef APSTUDIO\_INVOKED

#include "targetver.h"

#endif

#include "afxres.h"

#include "verrsrc.h"

/////////////////////////////////////////////////////////////////////////////

#undef APSTUDIO\_READONLY\_SYMBOLS

/////////////////////////////////////////////////////////////////////////////

// 中文(简体，中国) resources

#if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_CHS)

LANGUAGE LANG\_CHINESE, SUBLANG\_CHINESE\_SIMPLIFIED

#ifdef APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// TEXTINCLUDE

//

1 TEXTINCLUDE

BEGIN

"resource.h\0"

END

2 TEXTINCLUDE

BEGIN

"#ifndef APSTUDIO\_INVOKED\r\n"

"#include ""targetver.h""\r\n"

"#endif\r\n"

"#include ""afxres.h""\r\n"

"#include ""verrsrc.h""\r\n"

"\0"

END

3 TEXTINCLUDE

BEGIN

"#define \_AFX\_NO\_SPLITTER\_RESOURCES\r\n"

"#define \_AFX\_NO\_OLE\_RESOURCES\r\n"

"#define \_AFX\_NO\_TRACKER\_RESOURCES\r\n"

"#define \_AFX\_NO\_PROPERTY\_RESOURCES\r\n"

"\r\n"

"#if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_CHS)\r\n"

"LANGUAGE 4, 2\r\n"

"#include ""res\\My.rc2"" // 非 Microsoft Visual C++ 编辑的资源\r\n"

"#include ""l.CHS\\afxres.rc"" // 标准组件\r\n"

"#endif\r\n"

"\0"

END

#endif // APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// Icon

//

// Icon with lowest ID value placed first to ensure application icon

// remains consistent on all systems.

IDR\_MAINFRAME ICON "res\\端点检测波形展示.ico"

/////////////////////////////////////////////////////////////////////////////

//

// Dialog

//

IDD\_ABOUTBOX DIALOGEX 0, 0, 197, 90

STYLE DS\_SETFONT | DS\_MODALFRAME | DS\_FIXEDSYS | WS\_POPUP | WS\_CAPTION | WS\_SYSMENU

CAPTION "About Endpoint Detection"

FONT 9, "MS Shell Dlg", 0, 0, 0x1

BEGIN

ICON IDR\_MAINFRAME,IDC\_STATIC,14,14,18,18

LTEXT "Endpoint Detection waveform display，1.0 版",IDC\_STATIC,42,14,114,8,SS\_NOPREFIX

LTEXT "Copyright (C) 2016",IDC\_STATIC,42,26,114,8

DEFPUSHBUTTON "Ok",IDOK,140,69,50,14,WS\_GROUP

LTEXT "by leafspace",IDC\_STATIC,140,53,39,8

LTEXT "Contact me : 18852923073@163.com",IDC\_STATIC,42,40,122,8

END

IDD\_MY\_DIALOG DIALOGEX 0, 0, 753, 411

STYLE DS\_SETFONT | DS\_FIXEDSYS | WS\_MINIMIZEBOX | WS\_POPUP | WS\_VISIBLE | WS\_CAPTION | WS\_SYSMENU | WS\_THICKFRAME

EXSTYLE WS\_EX\_APPWINDOW

CAPTION "Endpoint Detection"

FONT 9, "MS Shell Dlg", 0, 0, 0x1

BEGIN

DEFPUSHBUTTON "Quit",IDOK,486,385,50,15

GROUPBOX "Voice",IDC\_STATIC1,10,5,535,70

GROUPBOX "CepsDistance",IDC\_STATIC2,10,80,535,70

GROUPBOX "EngChart",IDC\_STATIC3,10,155,535,70

GROUPBOX "EngDevia",IDC\_STATIC4,10,230,535,70

GROUPBOX "SpectrumEnt",IDC\_STATIC5,10,305,535,70

EDITTEXT IDC\_EDIT1,51,385,202,12,ES\_AUTOHSCROLL

LTEXT "File Name",IDC\_STATIC,15,387,31,8

CONTROL "Choose file",IDC\_COMMAND1,"Button",BS\_COMMANDLINK | WS\_TABSTOP,259,380,73,21

PUSHBUTTON "Show",IDC\_BUTTON1,347,382,44,19

GROUPBOX "Ctrl Feature Box",IDC\_STATIC,552,5,190,205

CONTROL "",IDC\_SLIDER1,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,15,100,15

CONTROL "",IDC\_SLIDER2,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,40,100,15

CONTROL "",IDC\_SLIDER3,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,65,100,15

CONTROL "",IDC\_SLIDER4,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,90,100,15

CONTROL "",IDC\_SLIDER5,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,115,100,15

CONTROL "",IDC\_SLIDER6,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,140,100,15

CONTROL "",IDC\_SLIDER7,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,165,100,15

CONTROL "",IDC\_SLIDER8,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,190,100,15

LTEXT "threshold\_h\_EC",IDC\_STATIC,560,17,50,10

LTEXT "threshold\_l\_EC",IDC\_STATIC,560,42,50,10

LTEXT "threshold\_h\_CD",IDC\_STATIC,560,67,50,10

LTEXT "threshold\_l\_CD",IDC\_STATIC,560,92,50,10

LTEXT "threshold\_h\_ED",IDC\_STATIC,560,117,50,10

LTEXT "threshold\_l\_ED",IDC\_STATIC,560,142,50,10

LTEXT "threshold\_h\_SE",IDC\_STATIC,560,167,50,10

LTEXT "threshold\_l\_SE",IDC\_STATIC,560,192,50,10

GROUPBOX "Ctrl Voice Length Box",IDC\_STATIC,552,214,190,61

CONTROL "",IDC\_SLIDER9,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,224,100,15

CONTROL "",IDC\_SLIDER10,"msctls\_trackbar32",TBS\_BOTH | TBS\_NOTICKS | WS\_TABSTOP,615,249,100,15

LTEXT "interim",IDC\_STATIC,560,226,50,10

LTEXT "burst",IDC\_STATIC,560,249,50,10

EDITTEXT IDC\_EDIT2,715,17,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT3,715,42,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT4,715,67,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT5,715,92,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT6,715,117,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT7,715,142,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT8,715,167,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT9,715,192,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT10,715,226,22,12,ES\_AUTOHSCROLL | ES\_READONLY

EDITTEXT IDC\_EDIT11,715,251,22,12,ES\_AUTOHSCROLL | ES\_READONLY

PUSHBUTTON "Create new voice",IDC\_BUTTON2,397,382,81,18

CONTROL "",IDC\_LIST1,"SysListView32",LVS\_REPORT | LVS\_ALIGNLEFT | WS\_BORDER | WS\_TABSTOP,557,294,180,89

GROUPBOX "Paragraph parameter",IDC\_STATIC,552,280,190,123

LTEXT "Paragraph：0",IDC\_STATIC6,680,388,55,8

PUSHBUTTON "Save",IDC\_BUTTON3,577,387,24,12

END

/////////////////////////////////////////////////////////////////////////////

//

// Version

//

VS\_VERSION\_INFO VERSIONINFO

FILEVERSION 1,0,0,1

PRODUCTVERSION 1,0,0,1

FILEFLAGSMASK 0x3fL

#ifdef \_DEBUG

FILEFLAGS 0x1L

#else

FILEFLAGS 0x0L

#endif

FILEOS 0x40004L

FILETYPE 0x1L

FILESUBTYPE 0x0L

BEGIN

BLOCK "StringFileInfo"

BEGIN

BLOCK "080404B0"

BEGIN

VALUE "CompanyName", "TODO: <公司名>"

VALUE "FileDescription", "端点检测波形展示"

VALUE "FileVersion", "1.0.0.1"

VALUE "InternalName", "端点检测波形展示.exe"

VALUE "LegalCopyright", "TODO: (C) <公司名>。保留所有权利。"

VALUE "OriginalFilename", "端点检测波形展示.exe"

VALUE "ProductName", "TODO: <产品名>"

VALUE "ProductVersion", "1.0.0.1"

END

END

BLOCK "VarFileInfo"

BEGIN

VALUE "Translation", 0x804, 1200

END

END

/////////////////////////////////////////////////////////////////////////////

//

// DESIGNINFO

//

#ifdef APSTUDIO\_INVOKED

GUIDELINES DESIGNINFO

BEGIN

IDD\_ABOUTBOX, DIALOG

BEGIN

LEFTMARGIN, 7

RIGHTMARGIN, 190

TOPMARGIN, 7

BOTTOMMARGIN, 83

END

IDD\_MY\_DIALOG, DIALOG

BEGIN

BOTTOMMARGIN, 410

END

END

#endif // APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// String Table

//

STRINGTABLE

BEGIN

IDS\_ABOUTBOX "About Endpoint Detection(&A)..."

END

#endif // 中文(简体，中国) resources

/////////////////////////////////////////////////////////////////////////////

#ifndef APSTUDIO\_INVOKED

/////////////////////////////////////////////////////////////////////////////

//

// Generated from the TEXTINCLUDE 3 resource.

//

#define \_AFX\_NO\_SPLITTER\_RESOURCES

#define \_AFX\_NO\_OLE\_RESOURCES

#define \_AFX\_NO\_TRACKER\_RESOURCES

#define \_AFX\_NO\_PROPERTY\_RESOURCES

#if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_CHS)

LANGUAGE 4, 2

#include "res\My.rc2" // 非 Microsoft Visual C++ 编辑的资源

#include "l.CHS\afxres.rc" // 标准组件

#endif

/////////////////////////////////////////////////////////////////////////////

#endif // not APSTUDIO\_INVOKED

// 端点检测波形展示Dlg.h : 头文件

//

#pragma once

#include "Wav\_File\_Handle.h"

// C端点检测波形展示Dlg 对话框

class C端点检测波形展示Dlg : public CDialogEx

{

// 构造

public:

C端点检测波形展示Dlg(CWnd\* pParent = NULL); // 标准构造函数

// 对话框数据

enum { IDD = IDD\_MY\_DIALOG };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

HICON m\_hIcon;

// 生成的消息映射函数

virtual BOOL OnInitDialog();

afx\_msg void OnSysCommand(UINT nID, LPARAM lParam);

afx\_msg void OnPaint();

afx\_msg HCURSOR OnQueryDragIcon();

DECLARE\_MESSAGE\_MAP()

private:

FILE \*fp;

Wav\_File\_Work \*File;

CPen newPen; //用于创建新画笔

CPen \*pOldPen; //用于存放旧画笔

CBrush newBrush; //用于创建新画刷

CBrush \*pOldBrush; //用于存放旧画刷

int width;

int height;

CSliderCtrl ctrlSlider\_threshold\_h\_EngChart;

CSliderCtrl ctrlSlider\_threshold\_l\_EngChart;

CSliderCtrl ctrlSlider\_threshold\_h\_CepsDistance;

CSliderCtrl ctrlSlider\_threshold\_l\_CepsDistance;

CSliderCtrl ctrlSlider\_threshold\_h\_EngDevia;

CSliderCtrl ctrlSlider\_threshold\_l\_EngDevia;

CSliderCtrl ctrlSlider\_threshold\_h\_SpectrumEnt;

CSliderCtrl ctrlSlider\_threshold\_l\_SpectrumEnt;

CSliderCtrl ctrlSlider\_interim\_StateLength;

CSliderCtrl ctrlSlider\_burst\_StateLength;

CListCtrl ctrlList\_paragraph;

bool repaint; //判断是否为重绘图标

CString fileName; //保存路径

HANDLE hdWrite; //用于保存DOS窗口的句柄

static const double DEVIATION; //定义的偏差值范围

vector<double> showMaxData, showMinData; //保存获取的数据

vector<bool> viabilityMaxData, viabilityMinData; //保存数据的可行性

void getTempVoiceData(void); //获取要处理的数据

void screenShowData(vector<double> showData, vector<bool> viabilityData); //输入:保存的数据，可行性数据，返回min的数值，用于计算范围显示用 ；这个函数用于筛选出偏差值在30%以内的数据

public:

void Repaint(void);

void StartDraw(CRect rc, CDC\* pdc, int flag, int f);

void Plot1(CRect rc, CDC\* pdc, int flag);

void Plot2(CRect rc, CDC\* pdc, int flag);

void Plot3(CRect rc, CDC\* pdc, int flag);

void Plot4(CRect rc, CDC\* pdc, int flag);

void Plot5(CRect rc, CDC\* pdc, int flag);

void ShowParagraphList(void);

afx\_msg void OnBnClickedCommand1();

afx\_msg void OnBnClickedButton1();

afx\_msg void OnNMCustomdrawSlider1(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider2(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider3(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider4(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider5(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider6(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider7(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider8(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider9(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnNMCustomdrawSlider10(NMHDR \*pNMHDR, LRESULT \*pResult);

afx\_msg void OnBnClickedButton2();

afx\_msg void OnBnClickedButton3();

};

// 端点检测波形展示Dlg.cpp : 实现文件

//

#include "stdafx.h"

#include "端点检测波形展示.h"

#include "端点检测波形展示Dlg.h"

#include "afxdialogex.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

const double C端点检测波形展示Dlg::DEVIATION = 0.3; //偏差值初始化(只能在Cpp中初始化)

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialogEx

{

public:

CAboutDlg();

// 对话框数据

enum { IDD = IDD\_ABOUTBOX };

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 支持

// 实现

protected:

DECLARE\_MESSAGE\_MAP()

};

CAboutDlg::CAboutDlg() : CDialogEx(CAboutDlg::IDD)

{

}

void CAboutDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

}

BEGIN\_MESSAGE\_MAP(CAboutDlg, CDialogEx)

END\_MESSAGE\_MAP()

// C端点检测波形展示Dlg 对话框

C端点检测波形展示Dlg::C端点检测波形展示Dlg(CWnd\* pParent /\*=NULL\*/)

: CDialogEx(C端点检测波形展示Dlg::IDD, pParent)

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void C端点检测波形展示Dlg::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

DDX\_Control(pDX, IDC\_SLIDER1, ctrlSlider\_threshold\_h\_EngChart); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER2, ctrlSlider\_threshold\_l\_EngChart); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER3, ctrlSlider\_threshold\_h\_CepsDistance); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER4, ctrlSlider\_threshold\_l\_CepsDistance); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER5, ctrlSlider\_threshold\_h\_EngDevia); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER6, ctrlSlider\_threshold\_l\_EngDevia); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER7, ctrlSlider\_threshold\_h\_SpectrumEnt); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER8, ctrlSlider\_threshold\_l\_SpectrumEnt); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER9, ctrlSlider\_interim\_StateLength); //初始化滑块

DDX\_Control(pDX, IDC\_SLIDER10, ctrlSlider\_burst\_StateLength); //初始化滑块

DDX\_Control(pDX, IDC\_LIST1, ctrlList\_paragraph); //初始化列表

}

BEGIN\_MESSAGE\_MAP(C端点检测波形展示Dlg, CDialogEx)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_COMMAND1, &C端点检测波形展示Dlg::OnBnClickedCommand1)

ON\_BN\_CLICKED(IDC\_BUTTON1, &C端点检测波形展示Dlg::OnBnClickedButton1)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER1, &C端点检测波形展示Dlg::OnNMCustomdrawSlider1)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER2, &C端点检测波形展示Dlg::OnNMCustomdrawSlider2)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER3, &C端点检测波形展示Dlg::OnNMCustomdrawSlider3)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER4, &C端点检测波形展示Dlg::OnNMCustomdrawSlider4)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER5, &C端点检测波形展示Dlg::OnNMCustomdrawSlider5)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER6, &C端点检测波形展示Dlg::OnNMCustomdrawSlider6)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER7, &C端点检测波形展示Dlg::OnNMCustomdrawSlider7)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER8, &C端点检测波形展示Dlg::OnNMCustomdrawSlider8)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER9, &C端点检测波形展示Dlg::OnNMCustomdrawSlider9)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER10, &C端点检测波形展示Dlg::OnNMCustomdrawSlider10)

ON\_BN\_CLICKED(IDC\_BUTTON2, &C端点检测波形展示Dlg::OnBnClickedButton2)

ON\_BN\_CLICKED(IDC\_BUTTON3, &C端点检测波形展示Dlg::OnBnClickedButton3)

END\_MESSAGE\_MAP()

// C端点检测波形展示Dlg 消息处理程序

BOOL C端点检测波形展示Dlg::OnInitDialog()

{

CDialogEx::OnInitDialog();

// 将“关于...”菜单项添加到系统菜单中。

// IDM\_ABOUTBOX 必须在系统命令范围内。

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

BOOL bNameValid;

CString strAboutMenu;

bNameValid = strAboutMenu.LoadString(IDS\_ABOUTBOX);

ASSERT(bNameValid);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// 设置此对话框的图标。当应用程序主窗口不是对话框时，框架将自动

// 执行此操作

SetIcon(m\_hIcon, TRUE); // 设置大图标

SetIcon(m\_hIcon, FALSE); // 设置小图标

//执行初始化操作

ctrlSlider\_threshold\_h\_EngChart.SetRange(0, 100);

ctrlSlider\_threshold\_l\_EngChart.SetRange(0, 100);

ctrlSlider\_threshold\_h\_CepsDistance.SetRange(0, 100);

ctrlSlider\_threshold\_l\_CepsDistance.SetRange(0, 100);

ctrlSlider\_threshold\_h\_EngDevia.SetRange(0, 100);

ctrlSlider\_threshold\_l\_EngDevia.SetRange(0, 100);

ctrlSlider\_threshold\_h\_SpectrumEnt.SetRange(0, 100);

ctrlSlider\_threshold\_l\_SpectrumEnt.SetRange(0, 100);

ctrlSlider\_interim\_StateLength.SetRange(5, 30);

ctrlSlider\_burst\_StateLength.SetRange(5, 30);

ctrlSlider\_threshold\_h\_EngChart.SetPos(80);

ctrlSlider\_threshold\_l\_EngChart.SetPos(20);

ctrlSlider\_threshold\_h\_CepsDistance.SetPos(80);

ctrlSlider\_threshold\_l\_CepsDistance.SetPos(20);

ctrlSlider\_threshold\_h\_EngDevia.SetPos(80);

ctrlSlider\_threshold\_l\_EngDevia.SetPos(20);

ctrlSlider\_threshold\_h\_SpectrumEnt.SetPos(80);

ctrlSlider\_threshold\_l\_SpectrumEnt.SetPos(20);

ctrlSlider\_interim\_StateLength.SetPos(10);

ctrlSlider\_burst\_StateLength.SetPos(5);

CString str1 = (CString)("0.80");

SetDlgItemText(IDC\_EDIT2, str1);

SetDlgItemText(IDC\_EDIT4, str1);

SetDlgItemText(IDC\_EDIT6, str1);

SetDlgItemText(IDC\_EDIT8, str1);

CString str2 = (CString)("0.20");

SetDlgItemText(IDC\_EDIT3, str2);

SetDlgItemText(IDC\_EDIT5, str2);

SetDlgItemText(IDC\_EDIT7, str2);

SetDlgItemText(IDC\_EDIT9, str2);

SetDlgItemText(IDC\_EDIT10, (CString)("10"));

SetDlgItemText(IDC\_EDIT11, (CString)("5"));

ShowWindow(SW\_FORCEMINIMIZE);

// TODO: 在此添加额外的初始化代码

repaint = false;

fp = NULL;

File = NULL;

CRect rect;

ctrlList\_paragraph.GetHeaderCtrl()->EnableWindow(false); //固定标题不被移动

ctrlList\_paragraph.GetClientRect(&rect); //获取编程语言列表视图控件的位置和大小

ctrlList\_paragraph.SetExtendedStyle(ctrlList\_paragraph.GetExtendedStyle()

| LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES); //为列表视图控件添加全行选中和栅格风格

ctrlList\_paragraph.InsertColumn(0, \_T("No"), LVCFMT\_CENTER, 30, 0);

ctrlList\_paragraph.InsertColumn(1, \_T("Begin"), LVCFMT\_CENTER, (rect.Width() - 30) / 3, 1);

ctrlList\_paragraph.InsertColumn(2, \_T("End"), LVCFMT\_CENTER, (rect.Width() - 30) / 3, 2);

ctrlList\_paragraph.InsertColumn(3, \_T("Length"), LVCFMT\_CENTER, (rect.Width() - 30) / 3, 3);

/\*

if(AllocConsole()){ //用于生成一个DOS窗口用于显示数据

SetConsoleTitleW(\_T("噪声环境下端点检测波形展示数据调试")); //设置DOS窗口的标题

hdWrite = GetStdHandle(STD\_OUTPUT\_HANDLE); //获取窗口句柄并保存下来

}

\*/

return TRUE; // 除非将焦点设置到控件，否则返回 TRUE

}

void C端点检测波形展示Dlg::OnSysCommand(UINT nID, LPARAM lParam)

{

if ((nID & 0xFFF0) == IDM\_ABOUTBOX)

{

CAboutDlg dlgAbout;

dlgAbout.DoModal();

}

else

{

CDialogEx::OnSysCommand(nID, lParam);

}

}

// 如果向对话框添加最小化按钮，则需要下面的代码

// 来绘制该图标。对于使用文档/视图模型的 MFC 应用程序，

// 这将由框架自动完成。

void C端点检测波形展示Dlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // 用于绘制的设备上下文

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// 使图标在工作区矩形中居中

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// 绘制图标

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialogEx::OnPaint();

}

if (repaint) { //repaint作为重绘标识符，首次执行时重绘触发器失败，等开始重绘时触发重绘

Repaint(); //重绘所需要的操作

}

}

//当用户拖动最小化窗口时系统调用此函数取得光标

//显示。

HCURSOR C端点检测波形展示Dlg::OnQueryDragIcon()

{

return static\_cast<HCURSOR>(m\_hIcon);

}

void C端点检测波形展示Dlg::Repaint()

{

CPaintDC dc(this); //准备参数

CWnd\* pWnd;

CRect rc;

CDC\* pdc;

for (int i = 0; i < 5; ++i) { //分别绘5个区域的图

pWnd = GetDlgItem(IDC\_STATIC1 + i);

pWnd->GetWindowRect(rc);

pdc = pWnd->GetDC();

StartDraw(rc, pdc, 1, i + 1); //绘制某一区域中的图像

}

}

void C端点检测波形展示Dlg::StartDraw(CRect rc, CDC\* pdc, int flag, int Flag)

{

width = rc.Width();

height = rc.Height();

CDC MemDC; //首先定义一个显示设备对象

CBitmap MemBitmap; //定义一个位图对象

MemDC.CreateCompatibleDC(NULL); //随后建立与屏幕显示兼容的内存显示设备

MemBitmap.CreateCompatibleBitmap(pdc, width, height); //下面建立一个与屏幕显示兼容的位图

CBitmap \*pOldBit = MemDC.SelectObject(&MemBitmap); //将位图选入到内存显示设备中//只有选入了位图的内存显示设备才有地方绘图，画到指定的位图上

MemDC.FillSolidRect(0, 0, width, height, RGB(240, 240, 240)); //先用背景色将位图清除干净

//MemDC.FillSolidRect(0, 17, width, height, RGB(255, 255, 255)); //先用背景色将位图清除干净

MemDC.Rectangle(1, 1, width - 1, height - 2);

newPen.CreatePen(PS\_SOLID, 1, RGB(255, 0, 0)); //创建实心画笔，粗度为1，颜色为红色

pOldPen = MemDC.SelectObject(&newPen); //选择新画笔，并将旧画笔的指针保存到pOldPen

switch (Flag) //绘制图形

{

case 1:

Plot1(rc, &MemDC, 1);

break;

case 2:

pdc->Rectangle(0, 0, width, height);

Plot2(rc, &MemDC, 1);

break;

case 3:

pdc->Rectangle(0, 0, width, height);

Plot3(rc, &MemDC, 1);

break;

case 4:

pdc->Rectangle(0, 0, width, height);

Plot4(rc, &MemDC, 1);

break;

case 5:

pdc->Rectangle(0, 0, width, height);

Plot5(rc, &MemDC, 1);

break;

default:

break;

}

MemDC.SelectObject(pOldPen); //恢复旧画笔

newPen.DeleteObject(); //删除新画笔

if (Flag == 1 || Flag == 3) { //绘制横坐标

pOldPen = MemDC.SelectObject(new CPen(PS\_SOLID, 1, RGB(0, 0, 0))); //选择新画笔，并将旧画笔的指针保存到pOldPen

MemDC.MoveTo(0, height / 2); //横轴开头

MemDC.LineTo(width, height / 2); //横轴结尾

for (int i = 0; i < width / 100; ++i) { //绘制标尺

MemDC.MoveTo((i + 1) \* 100, height / 2);

MemDC.LineTo((i + 1) \* 100, height / 2 - 10);

}

}

pdc->BitBlt(0, 0, width, height, &MemDC, 0, 0, SRCCOPY); //将内存中的图拷贝到屏幕上进行显示

MemBitmap.DeleteObject(); //绘图完成后的清理

MemDC.DeleteDC();

}

void C端点检测波形展示Dlg::Plot1(CRect rc, CDC\* pdc, int flag)

{

/\*

int max, min;

double highMax, highMin;

double tempNumberMax, tempNumberMin;

for (int i = 0; i < width; ++i) { //按像素逐个绘制图形

max = min = i\*File->Get\_dataNumber() / width;

for (double j = i\*File->Get\_dataNumber() / width; j < (i + 1)\*File->Get\_dataNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataDouble((int)j) > File->Get\_DataDouble(max)) {

max = (int)j;

}

if (File->Get\_DataDouble((int)j) < File->Get\_DataDouble(min)) {

min = (int)j;

}

}

tempNumberMax = File->Get\_DataDouble(max);

tempNumberMin = File->Get\_DataDouble(min);

highMax = height/2 - (height/2)\*tempNumberMax;

highMin = height/2 - (height/2)\*tempNumberMin;

pdc->MoveTo(i, (int)highMin);

pdc->LineTo(i, (int)highMax);

}

\*/

this->getTempVoiceData();

this->screenShowData(showMaxData, viabilityMaxData);

this->screenShowData(showMinData, viabilityMinData);

double highMax, highMin;

int topMax, topMin, bottomMax, bottomMin;

topMax = topMin = bottomMax = bottomMin = 0;

for (int i = 0; i < width; ++i) {

if (viabilityMaxData[i] && showMaxData[i] > showMaxData[topMax]) {

topMax = i;

}

if (viabilityMaxData[i] && showMaxData[i] < showMaxData[topMin]) {

topMin = i;

}

if (viabilityMinData[i] && showMinData[i] > showMinData[bottomMax]) {

bottomMax = i;

}

if (viabilityMinData[i] && showMinData[i] < showMinData[bottomMin]) {

bottomMin = i;

}

}

double unitTop = fabs(fabs(showMaxData[topMax]) - fabs(showMaxData[topMin]));

double unitBottom = fabs(fabs(showMinData[bottomMax]) - fabs(showMinData[bottomMin]));

vector<int> highTop, highBottom;

for (int i = 0; i < width; ++i) {

highMax = height / 2 - (fabs(showMaxData[i]) - fabs(showMaxData[topMin])) \* (height / 2 / unitTop);

highMin = height / 2 + (fabs(showMinData[i]) - fabs(showMinData[bottomMax])) \* (height / 2 / unitBottom);

if (!viabilityMaxData[i]) {

highMax = height / 2;

}

if (!viabilityMinData[i]) {

highMin = height / 2;

}

highTop.push\_back((int)highMin);

highBottom.push\_back((int)highMax);

}

for (int i = 0; i < width; ++i) {

pdc->MoveTo(i, highTop[i]);

pdc->LineTo(i, highBottom[i]);

}

for (unsigned long i = 0; i < File->Get\_voiceNumber(); ++i) {

VoiceParagraph tempPoint = File->Get\_dataVoicePoint(i);

int begin = (int)(tempPoint.begin / (File->Get\_FrameNumber() / width));

int end = (int)(tempPoint.end / (File->Get\_FrameNumber() / width));

pOldPen = pdc->SelectObject(new CPen(PS\_SOLID, 1, RGB(0, 0, 255))); //选择新画笔，并将旧画笔的指针保存到pOldPen

pdc->MoveTo(begin, 17);

pdc->LineTo(begin, height - 5);

pOldPen = pdc->SelectObject(new CPen(PS\_SOLID, 1, RGB(0, 255, 0))); //选择新画笔，并将旧画笔的指针保存到pOldPen

pdc->MoveTo(end, 17);

pdc->LineTo(end, height - 5);

pdc->DrawText((CString)("begin"), CRect(begin + 10, height - 20, begin + 50, height), 0);

pdc->DrawText((CString)("end"), CRect(end + 10, height - 20, end + 50, height), 0);

}

}

void C端点检测波形展示Dlg::Plot2(CRect rc, CDC\* pdc, int flag)

{

pdc->MoveTo(0, (int)height);

for (int i = 0; i < width; ++i) { //按像素逐个绘制图形

int max = 0;

for (double j = i\*File->Get\_FrameNumber() / width; j < (i + 1)\*File->Get\_FrameNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataCepsDistance((unsigned long)j) > File->Get\_DataCepsDistance(max)) {

max = (int)j;

}

}

double tempNumberMax = File->Get\_DataCepsDistance(max);

double highMax = height - height\*tempNumberMax;

pdc->LineTo(i, (int)highMax);

}

}

void C端点检测波形展示Dlg::Plot3(CRect rc, CDC\* pdc, int flag)

{

int max, min;

double highMax, highMin;

double tempNumberMax, tempNumberMin;

pdc->MoveTo(0, (int)height / 2);

for (int i = 0; i < width; ++i) { //按像素逐个绘制图形

max = min = 0;

for (double j = i\*File->Get\_FrameNumber() / width; j < (i + 1)\*File->Get\_FrameNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataEngChart((unsigned long)j) > File->Get\_DataEngChart(max)

&& File->Get\_DataEngChart((unsigned long)j) <= 1

&& File->Get\_DataEngChart((unsigned long)j) >= -1

) {

max = (int)j;

}

if (File->Get\_DataEngChart((unsigned long)j) < File->Get\_DataEngChart(min)

&& File->Get\_DataEngChart((unsigned long)j) <= 1

&& File->Get\_DataEngChart((unsigned long)j) >= -1

) {

min = (int)j;

}

}

tempNumberMax = File->Get\_DataEngChart(max);

tempNumberMin = File->Get\_DataEngChart(min);

highMax = height / 2 - (height / 2)\*tempNumberMax;

highMin = height / 2 - (height / 2)\*tempNumberMin;

pdc->MoveTo(i, (int)highMin);

pdc->LineTo(i, (int)highMax);

}

}

void C端点检测波形展示Dlg::Plot4(CRect rc, CDC\* pdc, int flag)

{

int max, min;

double highMax, highMin;

double tempNumberMax, tempNumberMin;

pdc->MoveTo(0, (int)height / 2);

for (int i = 0; i < width; ++i) { //按像素逐个绘制图形

max = min = 0;

for (double j = i\*File->Get\_FrameNumber() / width; j < (i + 1)\*File->Get\_FrameNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataEngDevia((unsigned long)j) > File->Get\_DataEngDevia(max)) {

max = (int)j;

}

if (File->Get\_DataEngDevia((unsigned long)j) < File->Get\_DataEngDevia(min)) {

min = (int)j;

}

}

tempNumberMax = File->Get\_DataEngDevia(max);

tempNumberMin = File->Get\_DataEngDevia(min);

tempNumberMax \*= 10;

tempNumberMin \*= 10;

tempNumberMax = tempNumberMax - (int)tempNumberMax;

tempNumberMin = tempNumberMin - (int)tempNumberMin;

highMax = height - height\*tempNumberMax;

highMin = height - height\*tempNumberMin;

pdc->MoveTo(i, (int)highMin);

pdc->LineTo(i, (int)highMax);

}

}

void C端点检测波形展示Dlg::Plot5(CRect rc, CDC\* pdc, int flag)

{

pdc->MoveTo(0, height);

for (int i = 0; i < width; ++i) { //按像素逐个绘制图形

int max = 0, min = 0;

max = min = (int)(i\*File->Get\_FrameNumber() / width);

for (double j = i\*File->Get\_FrameNumber() / width; j < (i + 1)\*File->Get\_FrameNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataSpectrumEnt((unsigned long)j) > File->Get\_DataSpectrumEnt(max)) {

max = (int)j;

}

if (File->Get\_DataSpectrumEnt((unsigned long)j) < File->Get\_DataSpectrumEnt(min)) {

min = (int)j;

}

}

double tempNumberMax = File->Get\_DataSpectrumEnt(max);

double tempNumberMin = File->Get\_DataSpectrumEnt(min);

double highMax = height / 2 - (height / 2)\*tempNumberMax;

double highMin = height / 2 - (height / 2) \* tempNumberMin;

pdc->MoveTo(i, (int)highMin);

pdc->LineTo(i, (int)highMax);

}

}

void C端点检测波形展示Dlg::ShowParagraphList()

{

ctrlList\_paragraph.DeleteAllItems();

//在列表视图控件中插入列表项，并设置列表子项文本

wchar\_t \*tempStr = new wchar\_t[50];

for (unsigned long i = 0; i < File->Get\_voiceNumber(); ++i) {

VoiceParagraph tempPoint = File->Get\_dataVoicePoint(i);

swprintf(tempStr, L"%d", (i + 1));

ctrlList\_paragraph.InsertItem(i, tempStr);

swprintf(tempStr, L"%d", tempPoint.begin);

ctrlList\_paragraph.SetItemText(i, 1, tempStr);

swprintf(tempStr, L"%d", tempPoint.end);

ctrlList\_paragraph.SetItemText(i, 2, tempStr);

swprintf(tempStr, L"%d", tempPoint.voiceLength);

ctrlList\_paragraph.SetItemText(i, 3, tempStr);

}

swprintf(tempStr, L"Paragraph：%d", (int)File->Get\_voiceNumber());

GetDlgItem(IDC\_STATIC6)->SetWindowText(tempStr);

}

void C端点检测波形展示Dlg::OnBnClickedCommand1()

{

// TODO: 在此添加控件通知处理程序代码

CFileDialog opendlg(TRUE, \_T("\*.wav"), \_T("\*.wav"), OFN\_OVERWRITEPROMPT, \_T("所有文件(\*.wav\*;)|\*.wav\*||"), NULL); //打开文件选择框

if (opendlg.DoModal() == IDOK)

{

fileName = opendlg.GetPathName(); //获取选择的文件名

}

SetDlgItemTextW(IDC\_EDIT1, fileName);

}

void C端点检测波形展示Dlg::OnBnClickedButton1()

{

// TODO: 在此添加控件通知处理程序代码

int nameLen = WideCharToMultiByte(CP\_ACP, 0, fileName, -1, NULL, 0, NULL, NULL);

char \*fileNameChar = new char[nameLen + 1];

WideCharToMultiByte(CP\_ACP, 0, fileName, -1, fileNameChar, nameLen, NULL, NULL); //将CString转为char\*

char \*fileNameTemp = new char[nameLen + 20];

int index = 0; //用于保存新文件名长度

for (int i = 0; i < nameLen + 1; ++i) { //处理'\'为'\\'，若文件中的路径分隔符为'\'则无法准确定位

fileNameTemp[index++] = fileNameChar[i];

if (fileNameChar[i] == '\\') {

fileNameTemp[index++] = '\\';

}

}

fileNameTemp[index] = 0; //防止原先没能保存完文件结束符

if ((fp = fopen(fileNameTemp, "rb")) == NULL) {

cout << "ERROR : File open failed !" << endl;

MessageBoxA(NULL, "ERROR : You have not selected the file !", "ERROR", MB\_ICONHAND);

return;

}

SetWindowText(\_T("端点检测波形展示 | Reading file | Please waiting a minute!"));

File = new Wav\_File\_Work(fp);

fclose(fp);

delete[] fileNameChar;

delete[] fileNameTemp;

SetWindowText(\_T("端点检测波形展示 | Beging painting | Please waiting a minute!"));

repaint = true; //表示即将开始重绘

InvalidateRect(NULL, true); //设置窗口无效

UpdateWindow(); //立即重绘，触发WM\_PAINT

showMaxData.clear();

showMinData.clear();

viabilityMaxData.clear();

viabilityMinData.clear();

this->ShowParagraphList();

SetWindowText(\_T("端点检测波形展示"));

}

void C端点检测波形展示Dlg::OnBnClickedButton2()

{

// TODO: 在此添加控件通知处理程序代码

if (this->fp == NULL) {

MessageBoxA(NULL, "ERROR : No File !", "ERROR", MB\_ICONHAND);

}

else {

CFileDialog opendlg(FALSE, \_T("\*.wav"), \_T("\*.wav"), OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, \_T("所有文件(\*.wav\*;)|\*.wav\*||"), NULL); //打开文件选择框

if (opendlg.DoModal() == IDOK)

{

CString fileName = opendlg.GetPathName(); //获取选择的文件名

int nameLen = WideCharToMultiByte(CP\_ACP, 0, fileName, -1, NULL, 0, NULL, NULL);

char \*fileNameChar = new char[nameLen + 1];

WideCharToMultiByte(CP\_ACP, 0, fileName, -1, fileNameChar, nameLen, NULL, NULL); //将CString转为char\*

char \*fileNameTemp = new char[nameLen + 20];

int index = 0; //用于保存新文件名长度

for (int i = 0; i < nameLen + 1; ++i) { //处理'\'为'\\'，若文件中的路径分隔符为'\'则无法准确定位

fileNameTemp[index++] = fileNameChar[i];

if (fileNameChar[i] == '\\') {

fileNameTemp[index++] = '\\';

}

}

fileNameTemp[index] = 0;

File->SaveNewWav(fileNameTemp);

MessageBoxA(NULL, "TIP : Voice created ok !", "TIP", MB\_ICONASTERISK);

}

}

}

void C端点检测波形展示Dlg::OnBnClickedButton3()

{

// TODO: 在此添加控件通知处理程序代码

if (this->fp == NULL) {

MessageBoxA(NULL, "ERROR : No File !", "ERROR", MB\_ICONHAND);

}

else {

CFileDialog opendlg(FALSE, \_T("\*.txt"), \_T("\*.txt"), OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, \_T("所有文件(\*.txt\*;)|\*.txt\*||"), NULL); //打开文件选择框

if (opendlg.DoModal() == IDOK)

{

CString fileName = opendlg.GetPathName(); //获取选择的文件名

int nameLen = WideCharToMultiByte(CP\_ACP, 0, fileName, -1, NULL, 0, NULL, NULL);

char \*fileNameChar = new char[nameLen + 1];

WideCharToMultiByte(CP\_ACP, 0, fileName, -1, fileNameChar, nameLen, NULL, NULL); //将CString转为char\*

char \*fileNameTemp = new char[nameLen + 20];

int index = 0; //用于保存新文件名长度

for (int i = 0; i < nameLen + 1; ++i) { //处理'\'为'\\'，若文件中的路径分隔符为'\'则无法准确定位

fileNameTemp[index++] = fileNameChar[i];

if (fileNameChar[i] == '\\') {

fileNameTemp[index++] = '\\';

}

}

fileNameTemp[index] = 0;

File->SaveParagraphInfo(fileNameTemp);

MessageBoxA(NULL, "TIP : Parahraph information saved ok !", "TIP", MB\_ICONASTERISK);

}

}

}

void C端点检测波形展示Dlg::getTempVoiceData(void) //获取要处理的数据

{

int max, min;

for (int i = 0; i < width; ++i) { //将数据范围分为宽度段，在段中找到Max，min

max = min = i\*File->Get\_dataNumber() / width;

for (double j = i\*File->Get\_dataNumber() / width;

j < (i + 1)\*File->Get\_dataNumber() / width; ++j) { //控制像素范围之内

if (File->Get\_DataDouble((int)j) > File->Get\_DataDouble(max)) {

max = (int)j;

}

if (File->Get\_DataDouble((int)j) < File->Get\_DataDouble(min)) {

min = (int)j;

}

}

this->showMaxData.push\_back(File->Get\_DataDouble(max));

this->showMinData.push\_back(File->Get\_DataDouble(min));

this->viabilityMaxData.push\_back(true);

this->viabilityMinData.push\_back(true);

}

}

void C端点检测波形展示Dlg::screenShowData(vector<double> showData, vector<bool> viabilityData) //输入:保存的数据，可行性数据，返回min的数值，用于计算范围显示用 ；这个函数用于筛选出偏差值在30%以内的数据

{

vector<double> deviationData; //用于保存计算的偏差值

bool flag = true; //作为一个标志，标志着没有筛选排除出新的数据

while (true) {

int viabilityNumber = 0; //保存当前可行的数据的个数，用于计算avg

double sum = 0;

for (int i = 0; i < (int)showData.size(); ++i) {

if (viabilityData[i] == false) { //如果此数为不为可行数据

continue;

}

else {

sum += showData[i]; //计算可行数据的和

++viabilityNumber;

}

}

double avg = sum / viabilityNumber;

for (int i = 0; i < (int)showData.size(); ++i) {

if (viabilityData[i] == false) {

deviationData.push\_back(0);

}

else { //如果数据为可行数据

deviationData.push\_back(fabs(showData[i] / avg - 1)); //计算偏差值公式

}

}

flag = true;

for (int i = 0; i < (int)showData.size(); ++i) {

if (viabilityData[i] == false) {

continue;

}

else {

if (deviationData[i] > DEVIATION) { //超出偏差值外的部分

viabilityData[i] = false; //被筛选排除了，可行性为false

flag = false;

}

}

}

deviationData.clear(); //清除偏差值，为下次筛选做准备

if (flag) { //没有筛选出数据，所以退出循环

break;

}

}

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider1(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_h\_EngChart.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT2, str);

Wav\_File\_Work::threshold\_h\_EngChart = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider2(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_l\_EngChart.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT3, str);

Wav\_File\_Work::threshold\_l\_EngChart = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider3(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_h\_CepsDistance.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT4, str);

Wav\_File\_Work::threshold\_h\_CepsDistance = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider4(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_l\_CepsDistance.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT5, str);

Wav\_File\_Work::threshold\_l\_CepsDistance = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider5(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_h\_EngDevia.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT6, str);

Wav\_File\_Work::threshold\_h\_EngDevia = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider6(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_l\_EngDevia.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT7, str);

Wav\_File\_Work::threshold\_l\_EngDevia = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider7(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_h\_SpectrumEnt.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT8, str);

Wav\_File\_Work::threshold\_h\_SpectrumEnt = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider8(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_threshold\_l\_SpectrumEnt.GetPos(); //获得滑块的当前位置

double trueData = (double)nPos / 100;

CString str = \_T("");

str.Format(\_T("%.2lf"), trueData);

SetDlgItemText(IDC\_EDIT9, str);

Wav\_File\_Work::threshold\_l\_SpectrumEnt = trueData;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider9(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_interim\_StateLength.GetPos(); //获得滑块的当前位置

CString str = \_T("");

str.Format(\_T("%d"), nPos);

SetDlgItemText(IDC\_EDIT10, str);

Wav\_File\_Work::interim\_StateLength = nPos;

\*pResult = 0;

}

void C端点检测波形展示Dlg::OnNMCustomdrawSlider10(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

// TODO: 在此添加控件通知处理程序代码

int nPos = ctrlSlider\_burst\_StateLength.GetPos(); //获得滑块的当前位置

CString str = \_T("");

str.Format(\_T("%d"), nPos);

SetDlgItemText(IDC\_EDIT11, str);

Wav\_File\_Work::burst\_StateLength = nPos;

\*pResult = 0;

}

#pragma once

#include <vector>

#include <iostream>

#include <windows.h>

using namespace std;

struct VoiceParagraph

{

unsigned long begin; //语音段落开始点

unsigned long end; //语音段落结束点

unsigned long voiceLength; //语音段落长度

VoiceParagraph(unsigned long be, unsigned long en, unsigned long vo) { //构造初始化

begin = be;

end = en;

voiceLength = vo;

}

};

class Wav\_File

{

private:

FILE \*fp; //文件指针

char \*fileName; //文件名或地址

char id\_RIFF[5]; //RIFF块标志

unsigned long fileSize; //文件的总字节数

char id\_WAVE[5]; //WAV标志

char id\_FMT[5]; //格式块标志

unsigned long formatLength; //格式块长度，16为正常，18说明有附加信息

short formatTag; //格式类别，值=1 表示编码方式为PCMu律编码

short channelsNumber; //声道数

unsigned long sampleRate; //每秒的样本数

unsigned long secondBytes; //每秒数据的字节数

short chunkAlign; //采样字节数

short sampleBits; //采样位数

short appendInformation; //附加信息，通过formatLength来判断

char id\_FACT[5]; //附加块标志

unsigned long appendLength; //附加块长度

unsigned long appendNone; //未知

char id\_DATA[5]; //附加块标志

unsigned long dataSize; //数据部分字节数

char \*data; //数据部分

long \*dataTuple; //每个样本的数据

unsigned long dataNumber; //样本的数据个数

long dataMax; //样本数据的最大值

long dataMin; //样本数据的最小值

bool Conversion\_Tuple(void); //将直接读取的数据转换为样本数据

long MakeWord(long NumberA, long NumberB); //合并字节

public:

Wav\_File(void) {

dataMax = 0;

dataMin = 1000000;

}

Wav\_File(FILE \*f) {

fp = f;

::Wav\_File();

}

void Give\_FP(FILE \*f) {

fp = f;

}

bool Read\_File(void); //读取文件

unsigned long Get\_SampleRate(void); //获取采样频率

short Get\_ChunkAlign(void); //获取样本字节数

long Get\_Data(unsigned long Number); //获取某个位置上的数据

unsigned long Get\_dataNumber(void); //获取样本数据个数

void ShowData(void); //输出数据

void SaveNewWav(unsigned long voiceNumber, vector<VoiceParagraph> voiceParagraph, int frameShift, char\* path); //保存去掉空白处的语音文件，参数1为段落的个数，参数2为各个语音段落的开始点与结束点信息

};

#pragma once

#include "Wav\_File\_Header.h"

#include <cmath>

#include <bitset>

#include <complex>

#include <fstream>

#define \_PARAGRAPH\_MUTE 0 //语音准备阶段

#define \_PARAGRAPH\_INTERIM 1 //语音假设阶段

#define \_PARAGRAPH\_VOICE 2 //语音进行阶段

#define \_PARAGRAPH\_BURST 3 //语音爆破阶段

struct FeatureParameter

{

double voice;

double engChart;

double cepsDistance;

double engDevia;

double spectrumEnt;

FeatureParameter(double voice, double engChart, double cepsDistance,

double engDevia, double spectrumEnt) {

this->voice = voice;

this->engChart = engChart;

this->cepsDistance = cepsDistance;

this->engDevia = engDevia;

this->spectrumEnt = spectrumEnt;

}

};

struct KalmanFilter

{

double x; //系统的状态量

double p; //估计误差协方差

double gain; //卡尔曼增益

double A; //x(n)=A\*x(n-1)+u(n),u(n)~N(0,q)

double H; //z(n)=H\*x(n)+w(n),w(n)~N(0,r)

double q; //预测过程噪声协方差

double r; //测量过程噪声协方差

static double predict\_q;

static double newMeasured\_q;

KalmanFilter(double init\_x, double init\_p) {

this->x = init\_x; //待测量的初始值，如有中值一般设成中值

this->p = init\_p; //后验状态估计值误差的方差的初始值（不要为0问题不大）

this->A = 1;

this->H = 1;

this->q = predict\_q; //预测（过程）噪声方差 影响收敛速率，可以根据实际需求给出

this->r = newMeasured\_q; //测量（观测）噪声方差 可以通过实验手段获得

}

double kalmanFilter\_filter(double newMeasured) {

this->x = this->A \* this->x; //%x的先验估计由上一个时间点的后验估计值和输入信息给出

this->p = this->A \* this->A \* this->p + this->q; //计算先验均方差 p(n|n-1)=A^2\*p(n-1|n-1)+q

this->gain = this->p \* this->H / (this->p \* this->H \* this->H + this->r);

this->x = this->x + this->gain \* (newMeasured - this->H \* this->x); //利用残余的信息改善对x(t)的估计，给出后验估计，这个值也就是输出

this->p = (1 - this->gain \* this->H) \* this->p; //%计算后验均方差

return this->x;

}

};

class Wav\_File\_Work :protected Wav\_File

{

private:

double \*dataDouble; //新转换的Double型数据，控制范围在[-1,1]

double \*\*dataFrameWindow; //保存分帧加窗后的数据

unsigned long frameNumber; //保存语音帧的个数

unsigned long voiceNumber; //语音段落个数

vector<VoiceParagraph> voiceParagraph; //保存这个语音文件中所有要处理的语音段落

unsigned long fftNumber; //傅里叶变换后的数据的个数

double \*\*dataCeps; //用于保存倒谱数据

int \*dataPitch; //用于保存在计算完倒谱后的基音周期

double \*\*dataMfcc; //用来保存mel倒谱系数

double \*dataCepsDistance; //用来保存倒谱距离

double \*dataEngChart; //用来保存短时能量

double \*dataEngDevia; //用来保存能量谱方差

double \*\*dataAutoFun; //用来保存自适应函数数据

double \*\*dataPSDF; //用来保存功率谱密度函数

double \*dataSpectrumEnt; //用来保存功率谱谱熵值

bool Conversion\_Double(void); //用来将新字节数据转换为Double数据

double Hamming\_window(double data); //汉明窗函数

bool Frame\_Window(void); //用来将新字节数据分帧加窗

bool kalmanFilter\_filter(void); //用于将数据进行卡尔曼滤波

double \*DistributionSpace(unsigned long col); //分配一维数组空间

double \*\*DistributionSpace(unsigned long row, unsigned long col); //分配二维数组空间

void ReleaseSpace(double \*\*data, unsigned long row); //释放二维数组空间

void Rader(double \*data, unsigned long dataNumber); //雷德算法，其中dataNumber一定是2^n

bool DFT(double \*data, unsigned long dataNumber); //正向离散傅立叶变换，无虚部

bool IDFT(double \*data, unsigned long dataNumber); //逆向离散傅立叶变换，无虚部

double\* FFT(double \*data, unsigned long dataNumber); //快速离散傅立叶变换，无虚部 (严重警告，因为可能出现扩展内存的情况，所以必须返回新地址)

bool IFFT(double \*data, unsigned long dataNumber); //快速离散逆傅立叶变换，无虚部

bool FrameFFT(double\*\* data, unsigned long frameNumber); //用于计算分帧后的整体FFT

bool FrameIFFT(double\*\* data, unsigned long frameNumber); //用于计算分帧后的整体IFFT

double Mel\_Filter(unsigned long data, int filterFlag); //mel 三角滤波器

double FilterFunction(int flag); //用于计算三角滤波器频率的辅助函数

bool AutocorrelativeFunction(void); //计算每一帧的自相关函数，是求功率谱谱熵的其中一部分

bool PSDF(void); //计算归一化的功率谱密度函数,在运行之前必须运行AutoCorrelativeFunction

bool NormallizationFun(double \*data, unsigned long dataNumber);

void Normalization(void); //将重点数据归一化操作

//五状态机操作 (输入为当前帧位置，返回的将是状态的转换情况)

void FrameVoice(double \*voice); //按照一定的计算方式计算每一帧的标准值

int Mute\_State\_Process(FeatureParameter framePoint); //无语音状态处理

int Interim\_State\_Process(FeatureParameter framePoint, unsigned long length); //语音假设状态处理

int Voice\_State\_Process(FeatureParameter framePoint); //语音状态处理

int Burst\_State\_Process(FeatureParameter framePoint, unsigned long length); //爆破音/无语音状态处理

void ShowEndpoint(); //使用这个函数获取所有的段落数据显示

public:

static const unsigned long N; //表示每个窗的帧长

static const int FrameShift; //窗函数的帧移

static const double PI; //数学圆周率PI

static const int FilterNumber; //mel 滤波器个数

static const int CoefficientOrder; //mfcc阶数

static double threshold\_h\_EngChart; //用来设置语音信号短时能量高门限

static double threshold\_l\_EngChart; //用来设置语音信号短时能量低门限

static double threshold\_h\_CepsDistance; //用来设置语音信号倒谱距离高门限

static double threshold\_l\_CepsDistance; //用来设置语音信号倒谱距离低门限

static double threshold\_h\_EngDevia; //用来设置语音信号能量方差高门限

static double threshold\_l\_EngDevia; //用来设置语音信号倒谱距离低门限

static double threshold\_h\_SpectrumEnt; //用来设置语音信号能量-熵高门限

static double threshold\_l\_SpectrumEnt; //用来设置语音信号能量-熵低门限

static unsigned long interim\_StateLength; //语音假设状态最短持续长度

static unsigned long burst\_StateLength; //爆破音\无语音状态最短持续长度

Wav\_File\_Work(void) {}

Wav\_File\_Work(FILE \*f) :Wav\_File(f) {

try

{

if (Wav\_File::Read\_File() == false) { //调用父函数读取文件

throw;

}

dataDouble = NULL;

dataFrameWindow = NULL;

dataCeps = NULL;

dataPitch = NULL;

dataMfcc = NULL;

dataCepsDistance = NULL;

dataEngChart = NULL;

dataEngDevia = NULL;

dataAutoFun = NULL;

dataPSDF = NULL;

dataSpectrumEnt = NULL;

dataDouble = (double\*)malloc(sizeof(double)\*Wav\_File::Get\_dataNumber()); //为Double型数据申请内存空间

if (dataDouble == NULL) { //分配空间未成功

throw invalid\_argument("ERROR : Memory failure !");

}

if (Conversion\_Double() == false) { //将新字节数据转换为Double数据

throw invalid\_argument("ERROR : Data change failure !");

}

if (kalmanFilter\_filter() == false) { //将新字节数据转换为Double数据

throw invalid\_argument("ERROR : Data kalman filter failure !");

}

if (Frame\_Window() == false) { //分帧加窗

throw invalid\_argument("ERROR : Frame windows failure !");

}

//this->Cepstrum(); //计算倒频谱

//this->Pitch(); //计算基音周期

this->MFCC(); //计算梅尔倒谱系数

this->CepstrumDistance(); //计算倒谱距离

this->EnergySpectrum(); //计算能量谱

this->EnergyDeviation(); //计算能量谱方差

this->SpectrumEntropy(); //计算谱熵

this->Normalization();

//this->ShowData();

this->Endpoint\_Detection();

this->ShowEndpoint();

}

catch (invalid\_argument &e) {

MessageBoxA(NULL, e.what(), "ERROR", MB\_ICONHAND);

exit(-1);

}

}

~Wav\_File\_Work() {

free(dataDouble);

}

double Get\_dataNumber(void); //获取Double数据的个数

double Get\_DataDouble(unsigned long Number); //获取转换后的Double数据

int Get\_WindowLength(void); //获取帧长（窗的大小）

unsigned long Get\_FrameNumber(void); //获取帧数

double Get\_DataCepsDistance(unsigned long Number); //获取指定位置上的倒谱距离

double Get\_DataEngChart(unsigned long Number); //获取指定位置上的能量谱

double Get\_DataEngDevia(unsigned long Number); //获取指定位置上的能量谱方差

double Get\_DataSpectrumEnt(unsigned long Number); //获取指定位置上的功率谱谱熵

unsigned long Get\_voiceNumber(void); //获取语音段落个数

VoiceParagraph Get\_dataVoicePoint(unsigned long Number); //获取某个语音点

void ShowData(void); //覆盖父类的展示数据函数

void SaveNewWav(char\* path); //保存去掉空白处的语音文件

void SaveParagraphInfo(char\* path); //保存段落信息

bool Cepstrum(void); //倒频谱

bool Pitch(void); //基音周期（严重注意-运行前必须先计算倒谱Cepstrum）

bool MFCC(void); //梅尔倒谱系数

bool CepstrumDistance(void); //计算倒谱距离（严重注意-运行前必须先计算mel倒谱系数MFCC）

bool EnergySpectrum(void); //计算短时能量

bool EnergyDeviation(void); //计算能量谱方差(严重注意-运行谦必须先计算能量谱)

bool SpectrumEntropy(void); //计算谱熵

bool Endpoint\_Detection(void); //端点检测函数

};

#include "stdafx.h"

#include "Wav\_File\_Header.h"

bool Wav\_File::Conversion\_Tuple(void)

{

try

{

cout << "TIP : Change to real format data ..." << endl;

dataNumber = dataSize / chunkAlign; //求出数据个数

dataTuple = (long\*)malloc(sizeof(long)\*dataNumber); //为新字节数据空间赋值

long tempNum = 0; //用来保存每个新字节数据

for (unsigned long i = 0; i < dataNumber; ++i) { //总共有dataNumber个数据

int Flag = 0;

for (short j = 0; j < chunkAlign; ++j) { //每个数据有两个字节

tempNum = MakeWord((long)data[i\*chunkAlign + j], tempNum); //每次将tempNum做为高字节数据，新字节为低字节数据制作新数据

}

dataTuple[i] = tempNum; //保存下新字节数据

tempNum = 0; //初始化缓冲区操作

if (dataMax < dataTuple[i]) { //求出最大值

dataMax = dataTuple[i];

}

if (dataMin > dataTuple[i]) { //求出最小值

dataMin = dataTuple[i];

}

}

}

catch (invalid\_argument &e) {

cerr << e.what() << endl;

MessageBoxA(NULL, e.what(), "ERROR", MB\_ICONHAND);

return false;

}

return true;

}

long Wav\_File::MakeWord(long NumberA, long NumberB) //合并字节，NumberA表示高字节位，NumberB表示低字节位

{

if (NumberA == -128) { //补码情况

NumberA = 0; //实际上此时就是超出范围，强制赋0

}

int Flag = 0; //标志，用来表示最后的数的正负，1为负，0为正

if (NumberA >= 0) {

if (NumberB >= 0) {

return NumberA \* 256 + NumberB; //高低字节都是正数，则直接移位合并

}

else {

return NumberA \* 256 + abs(NumberB) + 128; //高字节为正数，低字节为负数

}

}

else {

Flag = 1;

if (NumberB > 0) { //高字节为负数，低字节为正数

return -1 \* (abs(NumberA) \* 256 + NumberB);

}

else { //高低字节都是负数

return -1 \* (abs(NumberA) \* 256 + abs(NumberB) + 128);

}

}

}

bool Wav\_File::Read\_File(void)

{

try

{

cout << "TIP : Reading file ..." << endl;

fread(id\_RIFF, sizeof(char), 4, fp); //读取'RIFF'

id\_RIFF[4] = 0; //末尾添零

if (strcmp(id\_RIFF, "RIFF")) {

throw invalid\_argument("ERROR : File not RIFF file !");

}

fread(&fileSize, sizeof(unsigned long), 1, fp); //读取文件的大小

fread(id\_WAVE, sizeof(char), 4, fp); //读取'WAVE'

id\_WAVE[4] = 0;

if (strcmp(id\_WAVE, "WAVE")) {

throw invalid\_argument("ERROR : File not WAVE file !");

}

fread(id\_FMT, sizeof(char), 4, fp); //读取'FMT'

id\_FMT[3] = 32;

id\_FMT[4] = 0;

fread(&formatLength, sizeof(unsigned long), 1, fp); //采样频率，单位khz

fread(&formatTag, sizeof(short), 1, fp); //读取文件标签

fread(&channelsNumber, sizeof(short), 1, fp); //读取通道数目

fread(&sampleRate, sizeof(unsigned long), 1, fp); //读取采样频率，单位hz

fread(&secondBytes, sizeof(unsigned long), 1, fp); //读取每秒数据量

fread(&chunkAlign, sizeof(short), 1, fp); //读取块对其

fread(&sampleBits, sizeof(short), 1, fp); //读取样本大小

if (formatLength > 16) {

fread(&appendInformation, sizeof(short), 1, fp); //读取附加信息

}

fread(id\_DATA, sizeof(char), 4, fp); //读取'DATA'

id\_DATA[4] = 0;

fread(&dataSize, sizeof(unsigned long), 1, fp); //读取数据大小

data = (char\*)malloc(sizeof(char)\*dataSize); //申请数据的存储空间

fread(data, sizeof(char), dataSize, fp); //读取数据

fclose(fp);

}

catch (invalid\_argument &e)

{

MessageBoxA(NULL, e.what(), "ERROR", MB\_ICONHAND);

return false;

}

if (Conversion\_Tuple() == true) { //转换格式成功

MessageBoxA(NULL, "TIP : File read ok !", "TIP", MB\_ICONASTERISK);

}

else {

MessageBoxA(NULL, "ERROR : Data change failure !", "ERROR", 0);

return false;

}

return true;

}

unsigned long Wav\_File::Get\_SampleRate(void) //获取采样频率

{

return sampleRate;

}

short Wav\_File::Get\_ChunkAlign(void) //获取样本字节数

{

return chunkAlign;

}

long Wav\_File::Get\_Data(unsigned long Number) //获取某个位置上的数据

{

if (Number >= dataNumber) { //如果所需要的数超过了数据个数

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

return -1;

}

else {

return dataTuple[Number];

}

}

unsigned long Wav\_File::Get\_dataNumber(void) //获取样本数据个数

{

return dataNumber;

}

void Wav\_File::ShowData(void)

{

cout << "TIP : Show data ..." << endl;

cout << id\_RIFF << endl;

cout << fileSize << endl;

cout << id\_WAVE << endl << endl;

cout << id\_FMT << endl;

cout << formatLength << endl;

cout << formatTag << endl;

cout << channelsNumber << endl;

cout << sampleRate << endl;

cout << secondBytes << endl;

cout << chunkAlign << endl;

cout << sampleBits << endl;

cout << appendInformation << endl << endl;

cout << id\_FACT << endl;

cout << appendLength << endl;

cout << appendNone << endl << endl;

cout << id\_DATA << endl;

cout << dataSize << endl;

for (unsigned int i = 0; i < dataNumber; ++i) {

cout << dataTuple[i] << "\t";

}

cout << endl;

cout << "MAX :" << dataMax << endl;

cout << "MIN :" << dataMin << endl;

}

void Wav\_File::SaveNewWav(unsigned long voiceNumber, vector<VoiceParagraph> voiceParagraph, int frameShift, char\* path) //保存去掉空白处的语音文件，参数1为段落的个数，参数2为各个语音段落的开始点与结束点信息

{

unsigned long endPointLength = 0;

for (unsigned long i = 0; i < voiceNumber; ++i) {

VoiceParagraph temp = voiceParagraph[i];

endPointLength += temp.voiceLength;

}

unsigned long removerLength = dataNumber - endPointLength \*frameShift;

unsigned long removerSize = removerLength\*chunkAlign;

//更改数据操作较多，暂时无法完成

//Wav\_File\* newWav = (Wav\_File\*)malloc(sizeof(Wav\_File));

//newWav = this;

FILE \*fp;

if ((fp = fopen(path, "wb")) == NULL) {

cout << "ERROR : File open failed !" << endl;

exit(-1);

}

try

{

unsigned long tempLength;

cout << "TIP : Writing file ..." << endl;

fwrite(id\_RIFF, sizeof(char), 4, fp); //写入'RIFF'

tempLength = fileSize - removerSize;

fwrite(&tempLength, sizeof(unsigned long), 1, fp); //写入文件的大小

fwrite(id\_WAVE, sizeof(char), 4, fp); //写入'WAVE'

fwrite(id\_FMT, sizeof(char), 4, fp); //写入'FMT'

fwrite(&formatLength, sizeof(unsigned long), 1, fp); //写入格式块长度

fwrite(&formatTag, sizeof(short), 1, fp); //写入文件标签

fwrite(&channelsNumber, sizeof(short), 1, fp); //写入通道数目

fwrite(&sampleRate, sizeof(unsigned long), 1, fp); //写入采样频率

fwrite(&secondBytes, sizeof(unsigned long), 1, fp); //写入每秒数据量

fwrite(&chunkAlign, sizeof(short), 1, fp); //写入块对其

fwrite(&sampleBits, sizeof(short), 1, fp); //写入样本大小

if (formatLength > 16) {

fwrite(&appendInformation, sizeof(short), 1, fp); //写入附加信息

}

fwrite(id\_DATA, sizeof(char), 4, fp); //写入'DATA'

tempLength = dataSize - removerSize;

fwrite(&tempLength, sizeof(unsigned long), 1, fp); //写入数据大小

unsigned long n = 0;

for (unsigned long i = 0; i < voiceNumber; ++i) { //逐段落写入语音端点信息

VoiceParagraph tempParagraph = voiceParagraph[i];

tempParagraph.begin \*= frameShift;

tempParagraph.end \*= frameShift;

for (unsigned long j = tempParagraph.begin; j < tempParagraph.end; ++j, n += 2) {

fwrite(&data[j \* 2], sizeof(char), 1, fp);

fwrite(&data[j \* 2 + 1], sizeof(char), 1, fp);

}

}

fclose(fp);

}

catch (invalid\_argument &e)

{

MessageBoxA(NULL, e.what(), "ERROR", MB\_ICONHAND);

}

}

#include "stdafx.h"

#include "Wav\_File\_Handle.h"

double KalmanFilter::predict\_q = 0.7; //预测过程中的噪声协方差 （严重注明：此处的数值需要经过历史经验与多次实验才能获得最佳数据）

double KalmanFilter::newMeasured\_q = 0.7; //测量过程中的噪声协方差 （严重注明：此处的数值需要经过历史经验与多次实验才能获得最佳数据）

const unsigned long Wav\_File\_Work::N = 256; //初始化每个窗的窗长

const int Wav\_File\_Work::FrameShift = 10; //初始化窗函数的帧移

const double Wav\_File\_Work::PI = 3.14159; //初始化圆周率

const int Wav\_File\_Work::FilterNumber = 24; //mel 滤波器个数

const int Wav\_File\_Work::CoefficientOrder = 12; //mfcc阶数

double Wav\_File\_Work::threshold\_h\_CepsDistance = 0.8;

double Wav\_File\_Work::threshold\_h\_EngChart = 0.8;

double Wav\_File\_Work::threshold\_h\_EngDevia = 0.8;

double Wav\_File\_Work::threshold\_h\_SpectrumEnt = 0.8;

double Wav\_File\_Work::threshold\_l\_CepsDistance = 0.2;

double Wav\_File\_Work::threshold\_l\_EngChart = 0.2;

double Wav\_File\_Work::threshold\_l\_EngDevia = 0.2;

double Wav\_File\_Work::threshold\_l\_SpectrumEnt = 0.2;

unsigned long Wav\_File\_Work::interim\_StateLength = 10; //语音假设状态最短持续长度

unsigned long Wav\_File\_Work::burst\_StateLength = 5; //爆破音\无语音状态最短持续长度

bool Wav\_File\_Work::Conversion\_Double(void) //用来将新字节数据转换为Double数据

{

try

{

cout << "TIP : Change to double data ..." << endl;

const double Flag = pow((double)2, Wav\_File::Get\_ChunkAlign() \* 8); //表示的是原数据最大值的一半

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) { //遍历每个数据

dataDouble[i] = (double)(Wav\_File::Get\_Data(i) / Flag); //控制每个数据在[-1,1]之间

}

}

catch (invalid\_argument &e) {

cerr << e.what() << endl;

MessageBoxA(NULL, e.what(), "ERROR", MB\_ICONHAND);

return false;

}

return true;

}

double Wav\_File\_Work::Hamming\_window(double data) //汉明窗函数

{

if (data >= 0 && data <= (N - 1)) { //0<= n <= N-1的情况

return 0.54 - 0.46\*cos(2 \* PI\*data / (N - 1)); //返回数值

}

else {

return 0;

}

}

bool Wav\_File\_Work::Frame\_Window(void) //用来将新字节数据分帧加窗

{

cout << "TIP : Frame window ..." << endl;

if (dataFrameWindow) { //如果原来有数据

ReleaseSpace(dataFrameWindow, frameNumber);

}

frameNumber = (unsigned long)(Wav\_File\_Work::Get\_dataNumber() - Wav\_File\_Work::N) / Wav\_File\_Work::FrameShift + 1; //求帧数

if (frameNumber < 1) {

return false;

}

dataFrameWindow = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //这是帧的个数

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) { //遍历窗中的每一个数据

if ((j + i\*Wav\_File\_Work::FrameShift) > this->Get\_dataNumber()) {

dataFrameWindow[i][j] = 0;

}

dataFrameWindow[i][j] = dataDouble[j + i\*Wav\_File\_Work::FrameShift] \* Hamming\_window(j); //拷贝原数据列中的数据

}

}

return true;

}

bool Wav\_File\_Work::kalmanFilter\_filter(void) //用于将数据进行卡尔曼滤波

{

double sumData = 0;

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) {

sumData += dataDouble[i];

}

KalmanFilter \*kalman = new KalmanFilter(sumData / Wav\_File::Get\_dataNumber(), 0);

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) { //每个数据进入卡尔曼滤波器处理

dataDouble[i] = kalman->kalmanFilter\_filter(dataDouble[i]);

}

return true;

}

double\* Wav\_File\_Work::DistributionSpace(unsigned long col) //分配一维数组空间

{

double \*tempspace = (double\*)malloc(sizeof(double)\*col);

if (tempspace == NULL) {

throw invalid\_argument("ERROR : Memory failure !");

return false;

}

return tempspace;

}

double\*\* Wav\_File\_Work::DistributionSpace(unsigned long row, unsigned long col) //分配二维数组空间

{

double \*\*tempspace = (double\*\*)malloc(sizeof(double\*)\*row);

if (tempspace == NULL) {

throw invalid\_argument("ERROR : Memory failure !");

return false;

}

for (unsigned long i = 0; i < row; ++i) {

tempspace[i] = (double\*)malloc(sizeof(double)\*col);

if (tempspace[i] == NULL) {

throw invalid\_argument("ERROR : Memory failure !");

return false;

}

}

return tempspace;

}

void Wav\_File\_Work::ReleaseSpace(double \*\*data, unsigned long row) //释放二维数组空间

{

for (unsigned long i = 0; i < row; ++i) {

free(data[i]);

}

free(data);

}

void Wav\_File\_Work::Rader(double \*data, unsigned long dataNumber) //雷德算法，其中dataNumber一定是2^n

{

unsigned long endNum;

double temp;

endNum = 0;

for (unsigned long i = 0; i < dataNumber - 1; i++) { //因为第一个数永远是0不变，所以查找次数为N-1

if (i < endNum) { //如果顺序数小于逆序数则交换，这是防止重复交换的一种手段，改为大于号也行

temp = data[i]; //交换过程

data[i] = data[endNum];

data[endNum] = temp;

}

unsigned long k = dataNumber >> 1; //100 代表窗长的一半，用来判断最高位是否为1

while (k <= endNum) { //如果逆序数的最高位为1

endNum = endNum - k; //则当前位置0

k >>= 1; //基标/2

}

endNum = endNum + k; //当前位为0，则逆转为1

}

}

bool Wav\_File\_Work::DFT(double \*data, unsigned long dataNumber) //正向离散傅立叶变换，无虚部

{

double exp; //这是指数部分

double \*dataDFT;

dataDFT = DistributionSpace(dataNumber);

for (unsigned long i = 0; i < dataNumber; ++i) { //一个一个建立新的数据

dataDFT[i] = 0; //预防假数据

exp = -2 \* PI\*i / dataNumber; //指数部分

for (unsigned long j = 0; j < dataNumber; ++j) { //做累加和

dataDFT[i] += data[j] \* cos(j\*exp); //纯实数部分的做法

}

}

for (unsigned long i = 0; i < dataNumber; ++i) {

data[i] = dataDFT[i]; //拷贝数据

}

free(dataDFT);

return true;

}

bool Wav\_File\_Work::IDFT(double \*data, unsigned long dataNumber) //逆向离散傅立叶变换，无虚部

{

double exp; //这是指数部分

double \*dataIDFT;

dataIDFT = DistributionSpace(dataNumber);

for (unsigned long i = 0; i < dataNumber; ++i) { //一个一个建立新的数据

dataIDFT[i] = 0; //预防假数据

exp = -2 \* PI\*i / dataNumber; //指数部分

for (unsigned long j = 0; j < dataNumber; ++j) { //做累加和

dataIDFT[i] += data[j] \* cos(j\*exp); //纯实数部分的做法

}

dataIDFT[i] /= dataNumber; //逆向回归

}

for (unsigned long i = 0; i < dataNumber; ++i) {

data[i] = dataIDFT[i]; //拷贝数据

}

free(dataIDFT);

return true;

}

double\* Wav\_File\_Work::FFT(double \*data, unsigned long dataNumber) //快速离散傅立叶变换，无虚部

{

short power = 0;

for (int i = dataNumber - 1; i > 0; ++power, i /= 2);

fftNumber = (unsigned long)pow((float)2, (int)power);

double \*dataFFT = DistributionSpace(fftNumber);

for (unsigned long i = 0; i < dataNumber; i++) {

dataFFT[i] = data[i]; //初始化快速傅立叶变换数据

}

double \*W = DistributionSpace(fftNumber / 2); //计算旋转因子

for (unsigned long i = 0; i < fftNumber / 2; i++) {

W[i] = cos(2 \* i \* PI / fftNumber);

}

unsigned long ulGroupLength = 1; //段的长度

unsigned long ulHalfLength = 0; //段长度的一半

unsigned long ulGroupCount = 0; //段的数量

double cw, c1, c2;

for (short b = 0; b < power; b++) //计算FFT

{

ulHalfLength = ulGroupLength;

ulGroupLength \*= 2;

for (unsigned long j = 0; j < fftNumber; j += ulGroupLength)

{

for (unsigned long k = 0; k < ulHalfLength; k++)

{

cw = W[k \* fftNumber / ulGroupLength], dataFFT[j + k + ulHalfLength];

c1 = dataFFT[j + k] + cw;

c2 = dataFFT[j + k] - cw;

dataFFT[j + k] = c1;

dataFFT[j + k + ulHalfLength] = c2;

}

}

}

free(W);

if (fftNumber == dataNumber) { //如何FFT运算时个数正好，没有扩展，则需要拷贝数据

for (unsigned long i = 0; i < dataNumber; ++i) {

data[i] = dataFFT[i];

}

free(dataFFT);

return NULL;

}

else { //否则需要返回新的数据地址

free(data);

return dataFFT;

}

}

bool Wav\_File\_Work::IFFT(double \*data, unsigned long dataNumber) //快速离散逆傅立叶变换，无虚部

{

short power = 0;

double \*dataIFFT;

for (unsigned long i = dataNumber - 1; i > 0; ++power, i /= 2); //计算次幂

for (unsigned long i = 1, j = dataNumber / 2; i < N - 1; ++i) { //雷德算法重排位置

if (i < j) {

double temp = data[i];

data[i] = data[j];

data[j] = temp;

}

unsigned long k = dataNumber / 2;

while (k <= j) {

j = j - k;

k = k / 2;

}

j = j + k;

}

dataIFFT = DistributionSpace(dataNumber);

for (unsigned long i = 0; i < dataNumber; ++i) { //逆操作

dataIFFT[i] = data[i] / dataNumber;

}

for (short p = 0; p < power; ++p) { //IFFT计算

for (unsigned long i = 0; i < pow((long double)2, (long double)p); ++i) {

double cw = i\*pow((long double)2, (long double)power - (p + 1)); //计算旋转因子

for (unsigned long j = i; j < dataNumber - 1; j += (unsigned long)pow((long double)2, (long double)(p + 1))) {

unsigned long index = j + (unsigned long)pow((long double)2, (long double)p);

cw = dataIFFT[index] \* cos(2 \* Wav\_File\_Work::PI\*cw / dataNumber);

dataIFFT[index] = dataIFFT[j] - cw;

dataIFFT[j] = dataIFFT[j] + cw;

}

}

}

for (unsigned long i = 0; i < dataNumber; ++i) { //拷贝数据到自身

data[i] = dataIFFT[i];

}

free(dataIFFT);

return true;

}

bool Wav\_File\_Work::FrameFFT(double\*\* data, unsigned long frameNumber) //用于计算分帧后的整体FFT

{

cout << "TIP : Calculate FrameFFT ..." << endl;

for (unsigned long i = 0; i < frameNumber; ++i) { //逐帧计算FFT

double \*space = this->FFT(data[i], Wav\_File\_Work::N); //保存下返回的地址，因为这有可能是新数据的地址

if (space != NULL) {

data[i] = space;

}

}

return true;

}

bool Wav\_File\_Work::FrameIFFT(double\*\* data, unsigned long frameNumber) //用于计算分帧后的整体IFFT

{

cout << "TIP : Calculate FrameIFFT ..." << endl;

for (unsigned long i = 0; i < frameNumber; ++i) { //逐帧计算IFFT

this->IFFT(data[i], Wav\_File\_Work::N);

}

return true;

}

double Wav\_File\_Work::Mel\_Filter(unsigned long data, int filterFlag) //mel 三角滤波器 公式来自网络

{

if (data < FilterFunction(filterFlag - 1) || data >= FilterFunction(filterFlag + 1)) {

return 0;

}

else if (data >= FilterFunction(filterFlag - 1) && data <= FilterFunction(filterFlag)) {

return 2 \* (data - FilterFunction(filterFlag - 1)) /

((FilterFunction(filterFlag + 1) - FilterFunction(filterFlag - 1))\*

(FilterFunction(filterFlag) - FilterFunction(filterFlag - 1)));

}

else if (data >= FilterFunction(filterFlag) && data <= FilterFunction(filterFlag + 1)) {

return 2 \* (FilterFunction(filterFlag - 1) - data) /

((FilterFunction(filterFlag + 1) - FilterFunction(filterFlag - 1))\*

(FilterFunction(filterFlag) - FilterFunction(filterFlag - 1)));

}

else {

return 0;

}

}

double Wav\_File\_Work::FilterFunction(int flag) //用于计算三角滤波器频率的辅助函数 参数来自网络

{

return 700 \* (exp(log((long double)1 + 20000 / 700)\*flag / (Wav\_File\_Work::FilterNumber + 1)) - 1);

}

bool Wav\_File\_Work::AutocorrelativeFunction(void) //计算每一帧的自相关函数，是求功率谱谱熵的其中一部分

{

if (dataAutoFun) {

ReleaseSpace(dataAutoFun, frameNumber);

}

dataAutoFun = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) {

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

dataAutoFun[i][j] = 0;

for (unsigned long k = 0; k < Wav\_File\_Work::N - j; ++k) {

dataAutoFun[i][j] += dataFrameWindow[i][k] \* dataFrameWindow[i][k + j];

}

}

}

return true;

}

bool Wav\_File\_Work::PSDF(void) //计算归一化的功率谱密度函数

{

if (dataPSDF) {

ReleaseSpace(dataPSDF, frameNumber);

}

dataPSDF = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) {

double sum = 0;

for (unsigned long j = 0; j < Wav\_File\_Work::N / 2; ++j) {

sum += dataAutoFun[i][j];

}

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

dataPSDF[i][j] = dataAutoFun[i][j] / sum;

}

}

return true;

}

bool Wav\_File\_Work::NormallizationFun(double \*data, unsigned long dataNumber)

{

double max = 0;

for (unsigned long i = 0; i < dataNumber; ++i) {

double tempdata = data[i] > 0 ? data[i] : -data[i];

if (tempdata<pow((double)10, -3) || tempdata>pow((double)10, 3)) {

data[i] = 0;

}

if (tempdata > max) {

max = tempdata;

}

}

for (unsigned long i = 0; i < dataNumber; ++i) {

data[i] /= max;

}

return true;

}

void Wav\_File\_Work::Normalization(void) //将重点数据归一化操作

{

if (dataDouble) {

this->NormallizationFun(dataDouble, Wav\_File::Get\_dataNumber());

}

if (dataCepsDistance) {

this->NormallizationFun(dataCepsDistance, frameNumber);

}

if (dataEngChart) {

this->NormallizationFun(dataEngChart, frameNumber);

}

if (dataEngDevia) {

this->NormallizationFun(dataEngDevia, frameNumber);

}

if (dataSpectrumEnt) {

this->NormallizationFun(dataSpectrumEnt, frameNumber);

for (unsigned long i = 0; i < frameNumber; ++i) {

dataSpectrumEnt[i] = dataSpectrumEnt[i] \* 100 - (unsigned long)(dataSpectrumEnt[i] \* 100);

if (\_finite(dataSpectrumEnt[i]) == 0) {

dataSpectrumEnt[i] = 0;

}

}

}

}

void Wav\_File\_Work::FrameVoice(double \*voice) //按照一定的计算方式计算每一帧的标准值, 先使用平均值

{

for (unsigned long i = 0; i < frameNumber; ++i) {

voice[i] = 0;

for (unsigned long j = 0; j < this->N; ++j) {

voice[i] += this->dataFrameWindow[i][j];

}

voice[i] /= this->N;

}

}

int Wav\_File\_Work::Mute\_State\_Process(FeatureParameter framePoint) //无语音状态处理

{

bool logic\_A = fabs(framePoint.engChart) > this->threshold\_l\_EngChart;

bool logic\_B = framePoint.cepsDistance > this->threshold\_l\_CepsDistance;

bool logic\_C = framePoint.engDevia > this->threshold\_l\_EngDevia;

bool logic\_D = framePoint.spectrumEnt > this->threshold\_l\_SpectrumEnt;

if ((logic\_A && logic\_B) || (logic\_A && logic\_C) || (logic\_B && logic\_C) ||

(logic\_A && logic\_D) || (logic\_C && logic\_D) || (logic\_B && logic\_D)) { //这是一个逻辑表达式，表示其中满足条件达到两个或以上则条件成功

return \_PARAGRAPH\_INTERIM;

}

else {

return \_PARAGRAPH\_MUTE;

}

}

int Wav\_File\_Work::Interim\_State\_Process(FeatureParameter framePoint, unsigned long length) //语音假设状态处理

{

//1） to \_PARAGRAPH\_MUTE

if (this->Mute\_State\_Process(framePoint) == \_PARAGRAPH\_MUTE) {

return \_PARAGRAPH\_MUTE;

}

else {

//2） to \_PARAGRAPH\_INTERIM

if (length < this->interim\_StateLength) {

return \_PARAGRAPH\_INTERIM;

}

else {

//3） to \_PARAGRAPH\_VOICE

return \_PARAGRAPH\_VOICE;

}

}

}

int Wav\_File\_Work::Voice\_State\_Process(FeatureParameter framePoint) //语音状态处理

{

bool logic\_A = (fabs(framePoint.engChart) > this->threshold\_l\_EngChart) &&

(framePoint.engChart < this->threshold\_h\_EngChart);

bool logic\_B = (framePoint.cepsDistance > this->threshold\_l\_CepsDistance) &&

(framePoint.cepsDistance < this->threshold\_h\_CepsDistance);

bool logic\_C = (framePoint.engDevia > this->threshold\_l\_EngDevia) &&

(framePoint.engDevia < this->threshold\_h\_EngDevia);

bool logic\_D = (framePoint.spectrumEnt > this->threshold\_l\_SpectrumEnt) &&

(framePoint.spectrumEnt < this->threshold\_h\_SpectrumEnt);

if ((logic\_A && logic\_B) || (logic\_A && logic\_C) || (logic\_B && logic\_C) ||

(logic\_A && logic\_D) || (logic\_C && logic\_D) || (logic\_B && logic\_D)) { //这是一个逻辑表达式，表示其中满足条件达到两个或以上则条件成功

//满足条件，则继续保持语音状态

return \_PARAGRAPH\_VOICE;

}

else {

//不在条件之中

return \_PARAGRAPH\_BURST;

}

}

int Wav\_File\_Work::Burst\_State\_Process(FeatureParameter framePoint, unsigned long length) //爆破音/无语音状态处理

{

//1) to \_PARAGRAPH\_MUTE

if (length > this->burst\_StateLength) {

return \_PARAGRAPH\_MUTE;

}

else {

//2) to \_PARAGRAPH\_VOICE

if (this->Voice\_State\_Process(framePoint) == \_PARAGRAPH\_VOICE) {

return \_PARAGRAPH\_VOICE;

}

else {

//3) to \_PARAGRAPH\_BURST

return \_PARAGRAPH\_BURST;

}

}

}

void Wav\_File\_Work::ShowEndpoint() //使用这个函数获取所有的段落数据显示

{

for (unsigned long i = 0; i < this->voiceNumber; ++i) {

VoiceParagraph tempParagraph = this->Get\_dataVoicePoint(i);

cout << tempParagraph.begin << " " << tempParagraph.end << " " << tempParagraph.voiceLength << endl;

}

}

double Wav\_File\_Work::Get\_dataNumber(void) //获取Double数据的个数

{

return Wav\_File::Get\_dataNumber();

}

double Wav\_File\_Work::Get\_DataDouble(unsigned long Number) //获取转换后的Double数据

{

if (Number >= (Wav\_File::Get\_dataNumber()) || Number < 0) { //如果所需要的数超过了数据个数

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

return -1;

}

else {

return dataDouble[Number];

}

}

int Wav\_File\_Work::Get\_WindowLength(void) //获取帧长（窗的大小）

{

return N;

}

unsigned long Wav\_File\_Work::Get\_FrameNumber(void) //获取帧数

{

return frameNumber;

}

double Wav\_File\_Work::Get\_DataCepsDistance(unsigned long Number) //获取指定位置上的倒谱距离

{

if (Number >= frameNumber || Number < 0) {

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

return -1;

}

else {

return dataCepsDistance[Number];

}

}

double Wav\_File\_Work::Get\_DataEngChart(unsigned long Number) //获取指定位置上的能量谱

{

if (Number >= frameNumber || Number < 0) {

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

return -1;

}

else {

return dataEngChart[Number];

}

}

double Wav\_File\_Work::Get\_DataEngDevia(unsigned long Number) //获取指定位置上的能量谱方差

{

if (Number >= frameNumber || Number < 0) {

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

return -1;

}

else {

return dataEngDevia[Number];

}

}

double Wav\_File\_Work::Get\_DataSpectrumEnt(unsigned long Number) //获取指定位置上的功率谱谱熵

{

if (Number >= frameNumber || Number < 0) {

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

return -1;

}

else {

return dataSpectrumEnt[Number];

}

}

unsigned long Wav\_File\_Work::Get\_voiceNumber(void) //获取语音段落个数

{

return voiceNumber;

}

VoiceParagraph Wav\_File\_Work::Get\_dataVoicePoint(unsigned long Number) //获取某个语音点

{

if (Number >= voiceNumber || Number < 0) { //如果所需要的数超过了数据个数

MessageBoxA(NULL, "ERROR : Over list !", "ERROR", MB\_ICONHAND);

throw invalid\_argument("ERROR : Over list !");

}

else {

return voiceParagraph[Number];

}

}

void Wav\_File\_Work::ShowData(void) //覆盖父类的展示数据函数

{

/\*

int max = 0, min = 0;

cout << "TIP : Double data " << endl;

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) {

cout << dataDouble[i] << "\t";

if (dataDouble[i] > dataDouble[max]) {

max = i;

}

if (dataDouble[i] < dataDouble[min]) {

min = i;

}

}

cout << endl;

\*/

/\*

cout << "Max " << dataDouble[max] << endl;

cout << "Min " << dataDouble[min] << endl;

cout << "TIP : Parameter " << endl;

cout << "Max energy " << Get\_maxEnergy() << endl;

cout << "Min energy " << Get\_minEnergy() << endl;

cout << "Max ZCR " << Get\_maxZCR() << endl;

cout << "Min ZCR " << Get\_minZCR() << endl;

cout << "TIP : Energy " << endl;

for (auto i : dataEnergy) {

cout << i << "\t";

}

cout << endl;

cout << "TIP : ZCR " << endl;

for (auto i : dataZCR) {

cout << i << "\t";

}

cout << endl;

for (int i = 0; i < (Wav\_File::Get\_dataNumber() - N); ++i) {

cout << Get\_DataEnergy(i) << endl;

}

for (int i = 0; i < (Wav\_File::Get\_dataNumber() - N); ++i) {

cout << Get\_DataZCR(i) << endl;

}

\*/

/\*

cout << "TIP : DFT " << endl;

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) {

cout << dataDFT[i] << "\t";

}

cout << endl;

\*/

/\*

cout << "TIP : FFT " << endl;

for (unsigned long i = 0; i < fftNumber; ++i) {

cout << dataFFT[i] << "\t";

}

cout << endl;

\*/

/\*

cout << "TIP : Cepstrum " << endl;

for (unsigned long i = 0; i < Wav\_File::Get\_dataNumber(); ++i) {

cout << dataCeps[i] << "\t";

}

cout << endl;

\*/

/\*

for (unsigned long i = 0; i < frameNumber; ++i) {

for (int j = 0; j < Wav\_File\_Work::CoefficientOrder; ++j) {

cout << dataMfcc[i][j] << "\t";

}

cout << endl;

}

\*/

/\*

cout << "倒谱距离数据：" << endl;

for (unsigned long i = 0; i < frameNumber; ++i) {

cout<< dataCepsDistance[i] << "\t";

}

cout << endl;

cout << "能量谱数据：" << endl;

for (unsigned long i = 0; i < frameNumber; ++i) {

cout << dataEngChart[i] << "\t";

}

cout << endl;

\*/

/\*

cout << "能量方差数据：" << endl;

for (unsigned long i = 0; i < frameNumber; ++i) {

dataEngDevia[i] \*= 10;

dataEngDevia[i] = dataEngDevia[i] - (int)dataEngDevia[i];

cout << dataEngDevia[i] << "\t";

}

cout << endl;

\*/

cout << "谱熵数据：" << endl;

for (unsigned long i = 0; i < frameNumber; ++i) {

cout << dataSpectrumEnt[i] << "\t";

}

cout << endl;

}

void Wav\_File\_Work::SaveNewWav(char\* path) //保存去掉空白处的语音文件

{

Wav\_File::SaveNewWav(voiceNumber, voiceParagraph, Wav\_File\_Work::FrameShift, path); //调用父类的生成函数

}

void Wav\_File\_Work::SaveParagraphInfo(char\* path) //保存检测信息到文件

{

ofstream paragraphData(path, ios::out);

if (paragraphData.is\_open()) {

unsigned long sumLength = 0;

paragraphData << "No | Begin | End | Length" << endl;

for (unsigned long i = 0; i < this->Get\_voiceNumber(); ++i) {

VoiceParagraph tempPoint = this->Get\_dataVoicePoint(i);

sumLength += tempPoint.voiceLength;

paragraphData << (i + 1) << " | " << tempPoint.begin << " | " << tempPoint.end << " | " << tempPoint.voiceLength << endl;

}

paragraphData << endl;

paragraphData << "Paragraph number : " << this->Get\_voiceNumber() << endl;

paragraphData << "Paragraph data number : " << this->Get\_FrameNumber() << endl;

paragraphData << "Paragraph useful length : " << sumLength << endl << endl;

paragraphData << "Settings" << endl;

paragraphData << "----------------------------" << endl;

paragraphData << "predict\_q : " << KalmanFilter::predict\_q << endl;

paragraphData << "newMeasured\_q : " << KalmanFilter::newMeasured\_q << endl;

paragraphData << "N : " << Wav\_File\_Work::N << endl;

paragraphData << "FrameShift : " << Wav\_File\_Work::FrameShift << endl;

paragraphData << "PI : " << Wav\_File\_Work::PI << endl;

paragraphData << "FilterNumber : " << Wav\_File\_Work::FilterNumber << endl;

paragraphData << "CoefficientOrder : " << Wav\_File\_Work::CoefficientOrder << endl;

paragraphData << "threshold\_h\_CepsDistance : " << Wav\_File\_Work::threshold\_h\_CepsDistance << endl;

paragraphData << "threshold\_h\_EngChart : " << Wav\_File\_Work::threshold\_h\_EngChart << endl;

paragraphData << "threshold\_h\_EngDevia : " << Wav\_File\_Work::threshold\_h\_EngDevia << endl;

paragraphData << "threshold\_h\_SpectrumEnt : " << Wav\_File\_Work::threshold\_h\_SpectrumEnt << endl;

paragraphData << "threshold\_l\_CepsDistance : " << Wav\_File\_Work::threshold\_l\_CepsDistance << endl;

paragraphData << "threshold\_l\_EngChart : " << Wav\_File\_Work::threshold\_l\_EngChart << endl;

paragraphData << "threshold\_l\_EngDevia : " << Wav\_File\_Work::threshold\_l\_EngDevia << endl;

paragraphData << "threshold\_l\_SpectrumEnt : " << Wav\_File\_Work::threshold\_l\_SpectrumEnt << endl;

paragraphData << "interim\_StateLength : " << Wav\_File\_Work::interim\_StateLength << endl;

paragraphData << "burst\_StateLength : " << Wav\_File\_Work::burst\_StateLength << endl;

}

else {

cout << "ERROR : Paragraph write open fail !" << endl;

}

}

bool Wav\_File\_Work::Cepstrum(void) //倒频谱

{

cout << "TIP : Calculate CepsTrum ..." << endl;

if (frameNumber < 1) { //如果检测无帧的处理

throw invalid\_argument("ERROR : No frame !");

return false;

}

double \*\*tempData = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //拷贝数据到处理数组，保护原有数据

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = dataFrameWindow[i][j];

}

}

this->FrameFFT(tempData, frameNumber); //对帧数据进行FFT

for (unsigned long i = 0; i < frameNumber; ++i) {

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = log(fabs(tempData[i][j])); //求模对数

}

}

dataCeps = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //拷贝数据到结果数组

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

dataCeps[i][j] = tempData[i][j];

}

}

ReleaseSpace(tempData, frameNumber);

this->FrameIFFT(dataCeps, frameNumber); //对帧数据进行IFFT

return true;

}

bool Wav\_File\_Work::Pitch(void) //基音周期

{

cout << "TIP : Calculate Pitch ..." << endl;

int Pmax = Wav\_File::Get\_SampleRate() / 60; //检测的最大限

int Pmin = Wav\_File::Get\_SampleRate() / 500; //检测的最低限

if (dataPitch) { //如果原来有数据则此为重做计划

free(dataPitch); //擦除原来的数据

}

dataPitch = (int\*)malloc(sizeof(int)\*frameNumber); //为数据分配空间

if (dataPitch == NULL) {

throw invalid\_argument("ERROR : Memory failure !");

return false;

}

for (unsigned long i = 0; i < frameNumber; ++i) { //每一帧都要找到一个最大值

unsigned long max = 0;

for (int j = Pmin; j < Pmax; ++j) { //限制查找范围

if (dataCeps[i][j] >= dataCeps[i][max]) { //找到数据的最大值

max = j;

}

}

dataPitch[i] = max;

}

return true;

}

bool Wav\_File\_Work::MFCC(void) //梅尔倒谱系数

{

cout << "TIP : Calculate MFCC ..." << endl;

if (frameNumber < 1) { //无帧时的紧急处理

throw invalid\_argument("ERROR : No frame !");

return false;

}

double \*\*tempData = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //拷贝数据到处理数组，保护原有数据

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = dataFrameWindow[i][j];

}

}

this->FrameFFT(tempData, frameNumber); //对帧数据进行FFT

for (unsigned long i = 0; i < frameNumber; ++i) {

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = pow(tempData[i][j], 2); //平方操作

}

}

double \*\*temp\_dataMfcc = DistributionSpace(frameNumber, Wav\_File\_Work::FilterNumber); //为数据分配空间

for (unsigned long i = 0; i < frameNumber; ++i) {

for (int m = 0; m < Wav\_File\_Work::FilterNumber; ++m) { //每个数据通过FilterNumber个三角滤波器

temp\_dataMfcc[i][m] = 0;

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) { //累加和计算数据

temp\_dataMfcc[i][m] += tempData[i][j] \* this->Mel\_Filter(j, m);

}

}

}

ReleaseSpace(tempData, frameNumber);

dataMfcc = DistributionSpace(frameNumber, Wav\_File\_Work::CoefficientOrder); //为数据分配空间

for (unsigned long i = 0; i < frameNumber; ++i) {

for (int n = 0; n < Wav\_File\_Work::CoefficientOrder; ++n) { //分为CoefficientOrder个阶数

dataMfcc[i][n] = 0;

for (int j = 0; j < Wav\_File\_Work::FilterNumber; ++j) { //公式求取mfcc

double tempdataA = fabs(temp\_dataMfcc[i][j]);

double tempdataB = cos((n + 1)\*Wav\_File\_Work::PI\*(j - 0.5) / Wav\_File\_Work::FilterNumber);

if (\_finite(log(tempdataA)\* tempdataB) == 0) {

continue;

}

dataMfcc[i][n] += log(tempdataA)\* tempdataB;

}

dataMfcc[i][n] \*= sqrt((long double)2 / Wav\_File\_Work::FilterNumber);

}

}

ReleaseSpace(temp\_dataMfcc, frameNumber);

return true;

}

bool Wav\_File\_Work::CepstrumDistance(void) //计算倒谱距离（严重注意-运行前必须先计算mel倒谱系数MFCC）

{

cout << "TIP : Calculate Cepstrum Distance ..." << endl;

dataCepsDistance = DistributionSpace(frameNumber); //为倒谱距离数据分配空间

double \*tempCeps = DistributionSpace(Wav\_File\_Work::CoefficientOrder); //为噪音数据分配空间

for (int i = 0; i < Wav\_File\_Work::CoefficientOrder; ++i) { //取前5帧计算背景噪音系数

tempCeps[i] = 0;

for (int j = 0; j < 5; ++j) {

tempCeps[i] += dataMfcc[j][i];

}

tempCeps[i] /= 5;

}

for (unsigned long i = 0; i < frameNumber; ++i) { //逐帧计算倒谱距离

dataCepsDistance[i] = 0;

for (int j = 0; j < Wav\_File\_Work::CoefficientOrder; ++j) {

dataCepsDistance[i] += pow(dataMfcc[i][j] - tempCeps[j], 2);

}

dataCepsDistance[i] = 4.3429\*sqrt(pow(dataMfcc[i][0] - tempCeps[0], 2) + 2 \* dataCepsDistance[i]);

}

return true;

}

bool Wav\_File\_Work::EnergySpectrum(void) //计算短时能量

{

cout << "TIP : Calculate Energy Spectrum ..." << endl;

dataEngChart = DistributionSpace(frameNumber);

double \*\*tempData = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //拷贝数据到处理数组，保护原有数据

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = dataFrameWindow[i][j];

}

}

this->FrameFFT(tempData, frameNumber); //对帧数据进行FFT

for (unsigned long i = 0; i < frameNumber; ++i) {

dataEngChart[i] = 0;

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

dataEngChart[i] += tempData[i][j];

}

}

ReleaseSpace(tempData, frameNumber);

return true;

}

bool Wav\_File\_Work::EnergyDeviation(void) //计算能量谱方差

{

cout << "TIP : Calculate Energy Deviation ..." << endl;

dataEngDevia = DistributionSpace(frameNumber);

double \*\*tempData = DistributionSpace(frameNumber, Wav\_File\_Work::N);

for (unsigned long i = 0; i < frameNumber; ++i) { //拷贝数据到处理数组，保护原有数据

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

tempData[i][j] = dataFrameWindow[i][j];

}

}

this->FrameFFT(tempData, frameNumber);

for (unsigned long i = 0; i < frameNumber; ++i) {

dataEngDevia[i] = 0;

double tempEngChart = 0;

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) { //重新计算EngChart，因为如果不加任何处理，EngChart数值将会很小

tempEngChart += fabs(tempData[i][j]);

}

for (unsigned long j = 0; j < Wav\_File\_Work::N; ++j) {

dataEngDevia[i] += pow(fabs(tempData[i][j]) - tempEngChart / Wav\_File\_Work::N, 2);

}

}

ReleaseSpace(tempData, frameNumber);

return true;

}

bool Wav\_File\_Work::SpectrumEntropy(void) //计算谱熵

{

cout << "TIP : Calculate Spectrum Entropy ..." << endl;

this->AutocorrelativeFunction(); //求每一帧的短时自相关函数

this->FrameFFT(dataAutoFun, frameNumber); //对短时自相关函数进行快速傅里叶变换，得到语音帧的短时功率谱密度

this->PSDF(); //计算这一帧中的每个频率分量的归一化功率谱密度函数

if (dataSpectrumEnt) {

free(dataSpectrumEnt);

}

dataSpectrumEnt = DistributionSpace(frameNumber);

for (unsigned long i = 0; i < frameNumber; ++i) {

dataSpectrumEnt[i] = 0;

for (unsigned long j = 0; j < Wav\_File\_Work::N / 2; ++j) {

dataSpectrumEnt[i] += dataPSDF[i][j] \* log(dataPSDF[i][j]); //依据归一化的功率谱密度，计算语音信息熵

}

dataSpectrumEnt[i] \*= -1;

}

return true;

}

bool Wav\_File\_Work::Endpoint\_Detection(void) //is a voise end point check function

{

int statusFlag = 0; //设置语音状态标志

unsigned long begin = 0; //语音段落的起点

unsigned long end = 0; //语音段落的终点

unsigned long voiceLength = 0; //语音段落的长度

unsigned long suppose\_V\_Length = 0; //假设状态中语音持续的长度

unsigned long burse\_V\_Length = 0; //爆破状态中语音持续的长度

bool voiceJudge = true; //如果为true则语音段落正常结束，如果为false，则说明语音段落还没有结束，用于终点判断

voiceNumber = 0;

statusFlag = \_PARAGRAPH\_MUTE; //初始时语音处于静音段

//计算voice的标准量

double \*voice = this->DistributionSpace(frameNumber); //为voice分配空间

this->FrameVoice(voice);

for (unsigned long i = 0; i < frameNumber; ++i) {

FeatureParameter feature(voice[i], this->dataEngChart[i], this->dataCepsDistance[i],

this->dataEngDevia[i], this->dataSpectrumEnt[i]);

switch (statusFlag)

{

case \_PARAGRAPH\_MUTE:

statusFlag = this->Mute\_State\_Process(feature);

//1) => \_PARAGRAPH\_MUTE

//2) => \_PARAGRAPH\_INTERIM

if (statusFlag == \_PARAGRAPH\_INTERIM) { //即将转换到准备阶段

suppose\_V\_Length = 0; //初始化准备阶段的持续时间长度

}

break;

case \_PARAGRAPH\_INTERIM:

statusFlag = this->Interim\_State\_Process(feature, suppose\_V\_Length);

//1) => \_PARAGRAPH\_MUTE

//2) => \_PARAGRAPH\_INTERIM

++suppose\_V\_Length; //若持续此状态，则添加准备时间的长度

//3) => \_PARAGRAPH\_VOICE

if (statusFlag == \_PARAGRAPH\_VOICE) { //即将转换到语音阶段

begin = i - suppose\_V\_Length; //则从准备阶段开始时的帧数则为起点

}

break;

case \_PARAGRAPH\_VOICE:

voiceJudge = false; //语音开始了

statusFlag = this->Voice\_State\_Process(feature);

//1) => \_PARAGRAPH\_VOICE

++voiceLength;

//2) => \_PARAGRAPH\_BURST

if (statusFlag == \_PARAGRAPH\_BURST) { //即将跳转到爆破音处，为计算爆破音的长度做准备

burse\_V\_Length = 0;

}

break;

case \_PARAGRAPH\_BURST:

statusFlag = this->Burst\_State\_Process(feature, burse\_V\_Length);

//1） => \_PARAGRAPH\_BURST

++burse\_V\_Length;

//2） => \_PARAGRAPH\_VOICE

//3） => \_PARAGRAPH\_MUTE

//this is a end point

if (statusFlag == \_PARAGRAPH\_MUTE) {

end = i - burse\_V\_Length;

voiceLength = end - begin;

VoiceParagraph voiceTemp(begin, end, voiceLength);

this->voiceParagraph.push\_back(voiceTemp); //保存这一段点

this->voiceNumber++;

voiceJudge = true;

}

break;

default:break; //不可能的情况,出错

}

}

if (!voiceJudge) {

VoiceParagraph voiceTemp(begin, frameNumber, frameNumber - begin);

this->voiceParagraph.push\_back(voiceTemp);

this->voiceNumber++;

}

return true;

}