# 实验三　实现简单的语音朗读

## 一、实验目的

1. 熟悉和掌握语音合成的基本方法，加深和巩固课本所学的知识。
2. 学会基本的编程方法。

## 二、实验内容

基本要求：利用微软的 speech sdk 或 Java speech api(jsapi)等实现简单的语音（中英文均可）朗读

更高要求：实现简单的语音识别。

## 三、实验环境

硬件：耳机（音箱）；麦克风（识别）

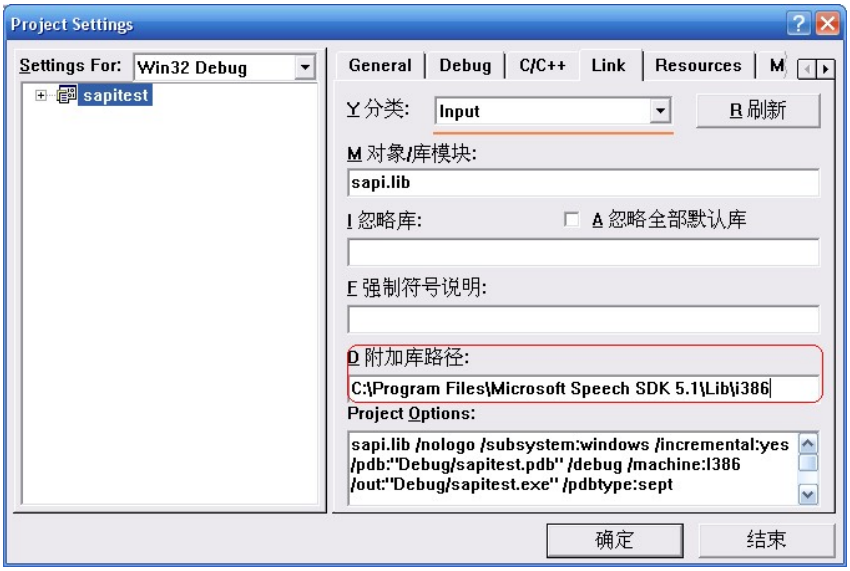
软件：microsoft speech sdk 或实现 Java speech api(jsapi)的工具包如 freetts（http://freetts.sourceforge.net/）或 sphinx4（http://cmusphinx.sourceforge.net/sphinx4/）

## 四、实验基本步骤和要点

调用 Speech sdk 的方法简述如下：


首先在 visual c++主窗口的菜单栏中选择“工程”—〉 “设置” 。在弹出对话框中选择“c/c++”选项卡，在分类中选择“preprocessor”,在附加包含路径中，输入 speech sdk 安装路径下的 include 目录，如：

C:\Program Files\Microsoft Speech SDK 5.1\Include




然后选择“link”选项卡，在分类中选择“Input”，在附加库路径中，输入 speech sdk 安装路径下的 lib 目录，如：

C:\Program Files\Microsoft Speech SDK 5.1\Lib\i386

程序中的关键代码如下：

```
#include <sapi.h>
#include <sphelper.h> //引入头文件，以调用 SDK 中的 API
    UpdateData();
     ISpVoice * pVoice = NULL;
     if (FAILED(CoInitialize(NULL)))
     {
          AfxMessageBox("Error to intiliaze COM");
          return;
     }//初始化 COM
    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice,
(void    **)&pVoice);
    if( SUCCEEDED( hr ) )
    {
    hr = pVoice->SetVolume(80);//音量控制 0～100
    hr = pVoice->SetRate(0);//语速控制-5～5
    hr = pVoice->Speak(L“hello world”, SPF_DEFAULT, NULL);//语音的内容
    ISpObjectToken * spToken = NULL;
     if (SUCCEEDED(SpFindBestToken(SPCAT_VOICES, L"language=804", NULL,
&SpToken)))
    {  pVoice->SetVoice(SpToken);//切换至中文语音模式，language=804 代表中文，409 代表
英文
      pVoice->Speak(L"世界你好", SPF_DEFAULT, NULL);
       SpToken->Release();
    }
    if (SUCCEEDED(SpFindBestToken(SPCAT_VOICES, L"language=409", L"Name=Microsoft
Mike", &SpToken)))//说话人控制
```

```
{
 pVoice->SetVoice(SpToken);
 pVoice->Speak(L"This is Mike speaking", SPF_DEFAULT, NULL);
 SpToken->Release();

}
 pVoice->Speak(L"<LANG   LANGID='804'>你好</LANG> ", SPF_IS_XML, NULL);//也可直接写成 xml 标签的格式
 pVoice->Speak(L"This sounds normal <pitch middle = '-10'/> but the pitch drops half way through", SPF_IS_XML, NULL ); //声调控制，-代表低音，+代表高音
 pVoice->Release();
pVoice = NULL;
}
CoUninitialize();
```

Speech sdk 还有很多功能，详见安装目录下的帮助文档。

## 五、参考文献

（1）补充资料：Microsoft Speech SDK 介绍；
（2）Microsoft Speech SDK 联机帮助和示例程序；
（3）MSDN 联机帮助；
（4）网络上相关介绍；

## 补充资料：Microsoft Speech SDK 介绍

**主要内容：**

**文本语音转换入门（p4）**



**Microsoft speech SDK TTS 编程介绍（p8）**

# 文本语音转换入门

## 一、内容简介

文本语音（Text-to-Speech，以下简称 TTS），它的作用就是把通过 TTS 引擎把文本转化为语音输出。本文不是讲述如何建立自己的 TTS 引擎，而是简单介绍如何运用 Microsoft Speech SDK 建立自己的文本语音转换应用程序。

## 二、Microsoft Speech SDK 简介

Microsoft Speech SDK 是微软提供的软件开发包，提供的 Speech API （SAPI）主要包含两大方面：

- 1. API for Text-to-Speech
- 2. API for Speech Recognition

其中 API for Text-to-Speech，就是微软 TTS 引擎的接口，通过它我们可以很容易地建立功能强大的文本语音程序，金山词霸的单词朗读功能就用到了这写 API，而目前几乎所有的文本朗读工具都是用这个 SDK 开发的。至于 API for Speech Recognition 就是与 TTS 相对应的语音识别，语音技术是一种令人振奋的技术，但由于目前语音识别技术准确度和识别速度不太理想，还未达到广泛应用的要求。

Microsoft Speech SDK 可以在微软的网站免费下载，目前的版本是 5.1，为了支持中文，还要把附加的语言包（LangPack）一起下载。

为了在 VC 中使用这 SDK，必需在工程中添加 SDK 的 include 和 lib 目录，为免每个工程都添加目录，最好的办法是在 VC 的
Option->Directoris 立加上 SDK 的 include 和 lib 目录。

## 三、一个最简单的例子

先看一个入门的例子：

```
#include <sapi.h>

#pragma comment(lib,"ole32.lib")  //CoInitialize CoCreateInstance 需要调用 ole32.dll
#pragma comment(lib,"sapi.lib")   //sapi.lib 在 SDK 的 lib 目录,必需正确配置
```

```
int main(int argc, char* argv[])
{
        ISpVoice * pVoice = NULL;

//COM 初始化：
        if (FAILED(::CoInitialize(NULL)))
                return FALSE;

//获取 ISpVoice 接口：
        HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice,
(void **)&pVoice);
        if( SUCCEEDED( hr ) )
        {
                hr = pVoice->Speak(L"Hello world", 0, NULL);
                pVoice->Release();
                pVoice = NULL;
        }

//千万不要忘记：
        ::CoUninitialize();
        return TRUE;
}
```

短短 20 几行代码就实现了文本语音转换，够神奇吧。SDK 提供的 SAPI 是基于 COM 封装的，无论你是否熟悉 COM，只要按部就班地用 CoInitialize(), CoCreateInstance()获取 IspVoice 接口就够了，需要注意的是初始化 COM 后，程序结束前一定要用 CoUninitialize()释放资源。

## 四、IspVoice 接口主要函数

上述程序的流程是获取 IspVoice 接口，然后用 ISpVoice::Speak（）把文本输出为语音，可见，程序的核心就是 IspVoice 接口。除了 Speak 外 IspVoice 接口还有许多成员函数，具体用法请参考 SDK 的文档。下面择要说一下几个主要函数的用法：

```
HRESULT Speak(const WCHAR *pwcs,DWORD dwFlags,ULONG *pulStreamNumber);
```

功能：就是 speak 了
参数：

*pwcs 输入的文本字符串，必需为 Unicode，如果是 ansi 字符串必需先转换为 Unicode。

dwFlags 用来标志 Speak 的方式，其中 SPF_IS_XML 表示输入文本含有 XML 标签，这个下文会讲到。

PulStreamNumber 输出，用来获取去当前文本输入的等候播放队列的位置，只有在异步模式才有用。

```
HRESULT Pause ( void );
HRESULT Resume ( void );
```

功能：一看就知道了。

```
HRESULT SetRate(long RateAdjust );
HRESULT GetRate(long *pRateAdjust);
```

功能：设置/获取播放速度，范围：-10 to 10

```
HRESULT SetVolume(USHORT usVolume);
HRESULT GetVolume(USHORT *pusVolume);
```

功能：设置/获取播放音量，范围：0 to 100

```
HRESULT SetSyncSpeakTimeout(ULONG msTimeout);
HRESULT GetSyncSpeakTimeout(ULONG *pmsTimeout);
```

功能：设置/获取同步超时时间。由于在同步模式中，电泳 Speak 后程序就会进入阻塞状态等待 Speak 返回，为免程序长时间没相应，应该设置超时时间，msTimeout 单位为毫秒。

```
HRESULT SetOutput(IUnknown *pUnkOutput,BOOL fAllowFormatChanges);
```

功能：设置输出，下文会讲到用 SetOutput 把 Speak 输出问 WAV 文件。
这些函数的返回类型都是 HRESULT，如果成功则返回 S_OK，错误有各自不同的错误码。

## 五、使用 XML

个人认为这个 TTS api 功能最强大之处在于能够分析 XML 标签，通过 XML 标签设置音量、音调、延长、停顿，几乎可以使输出达到自然语音效果。前面已经提过，把 Speak 参数 dwFlags 设为 SPF_IS_XML,TTS 引擎就会分析 XML 文本，输入文本并不需要严格遵守 W3C 的标准，只要含有 XML 标签就行了，下面举个例子：

……

```
pVoice->Speak(L"<VOICE REQUIRED=''NAME=Microsoft Mary''/>volume<VOLUME
LEVEL=''100''>turn up</VOLUME>", SPF_IS_XML, NULL);
```

……

```
<VOICE REQUIRED=''NAME=Microsoft Mary''/>
```

标签把声音设为 Microsoft Mary，英文版 SDK 中一共含有 3 种声音，另外两种是 Microsoft Sam 和 Microsoft Mike。

……

```
<VOLUME LEVEL=''100''>
```

把音量设为 100，音量范围是 0～100。

另外：标志音调（-10～10）：

```
<PITCH MIDDLE="10">text</PITCH>
```

注意：" 号在 C/C++中前面要加 \ ，否则会出错。 标志语速（-10～10）：

```
<RATE SPEED="-10">text</RATE>
```

逐个字母读：

```
<SPELL>text</SPELL>
```

强调：

<EMPH>text</EMPH>
停顿 200 毫秒（最长为 65,536 毫秒）：
<SILENCE MSEC="200" />
控制发音：
<PRON SYM = ''h eh - l ow 1''/>

这个标签的功能比较强，重点讲一下：所有的语言发音都是由基本的音素组成，拿中文发音来说，拼音是组成发音的最基本的元素，只要知道汉字的拼音，即使不知道怎么写，我们可知道这个字怎么都，对于 TTS 引擎来说，它不一定认识所有字，但是你把拼音对应的符号（SYM）给它，它就一定能够读出来，而英语发音则可以用音标表示，"h eh - l ow 1"就是 hello 这个单词对应的语素。至于发音与符号 SYM 具体对应关系请看 SDK 文档中的 Phoneme Table。

再另外，数字、日期、时间的读法也有一套规则，SDK 中有详细的说明，这里不说了（懒得翻译了），下面随便抛个例子：
<context ID = "date_ ymd">1999.12.21</context>
会读成
"December twenty first nineteen ninety nine"
XML 标签可以嵌套使用，但是一定要遵守 XML 标准。XML 标签确实好用，效果也不错，但是……缺点：一个字———"烦"，如果给一大段文字加标签，简直痛不欲生。

## 六、把文本语音输出为 WAV 文件

```
#include <sapi.h>
#include <sphelper.h>

#pragma comment(lib,"ole32.lib")
#pragma comment(lib,"sapi.lib")

int main(int argc, char* argv[])
{
	ISpVoice * pVoice = NULL;
	if (FAILED(::CoInitialize(NULL)))
		return FALSE;

	HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL,
		IID_ISpVoice, (void **)&pVoice);
	if( SUCCEEDED( hr ) )
	{
		CComPtr<ISpStream>                    cpWavStream;
		CComPtr<ISpStreamFormat>cpOldStream;
		CSpStreamFormat OriginalFmt;
		pVoice->GetOutputStream( &cpOldStream );
		OriginalFmt.AssignFormat(cpOldStream);
		hr = SPBindToFile( L"D:\\output.wav",SPFM_CREATE_ALWAYS,
			&cpWavStream,&OriginalFmt.FormatId(),
			OriginalFmt.WaveFormatExPtr() );
		if( SUCCEEDED( hr ) )
		{
			pVoice->SetOutput(cpWavStream,TRUE);
```

```
                        WCHAR WTX[] = L"<VOICE REQUIRED=''NAME=Microsoft Mary''/>text
to wave";
                        pVoice->Speak(WTX, SPF_IS_XML, NULL);
                        pVoice->Release();
                        pVoice = NULL;
                }
        }

        ::CoUninitialize();
        return TRUE;
}
```

SPBindToFile 把文件绑定到输出流上，而 SetOutput 把输出设为绑定文件的流上。

## Microsoft speech SDK TTS 编程介绍

## 第一部分 Speech SDK 概述

Microsoft Speech SDK 提供关于语音（Speech）处理的一套应用程序编程接口 SAPI（Speech Application Programming Interface）。SAPI 提供了实现文字–语音转换（Text-to-Speech）和语音识别（Speech Recognition）程序的基本函数，大大简化了语音编程的难度，降低了语音编程的工作量。Speech SDK 可以免费从如下网址下载：http://www.microsoft.com/speech。

由于 Speech SDK 是以 COM 接口的方式提供服务的，所以首先介绍 COM 的有关基础知识。

### （一）COM 基础

Speech SDK 提供了完善的 COM 接口，所以具备一定的 COM 编程基础对进行 Speech SDK 编程来说是非常必要的。笔者将简要介绍 COM 编程的基础知识。虽然这些知识对阅读本书来说是足够了，但是如果你没有进行过任何的 COM 编程实践，笔者还是建议你先阅读一本 COM 的教科书。

### 1．什么是 COM

组件对象模型（Component Object Model，COM）对象是符合 COM 规范的可重用的软件组件。符合 COM 规范的 COM 对象相互之间可以很好地工作，并且可以很容易地集成到应用程序中。从应用的观点来看，一个 COM 对象就是一个黑箱，应用程序可以使用它来创建一项或多项任务。

COM 对象常常用动态链接库（Dynamic Link Libraries，DLLs）的形式来实现。与传统的 DLL 一样，COM 对象暴露其方法，应用程序能调用这些方法来实现对象所支持的功能。应用程序与 COM 对象的关系就像应用程序与 C++对象的关系，但其中也存在一些区别。

1）COM 对象执行严格的封装。你不能简单地创建一个对象就调用其中的公用方法。COM 对象的公用方法聚合为一个或多个接口。为了使用一个方法，必须先创建一个对象，并从对

象中获得相应的接口。一个接口一般包含一组方法，通过它们能使用对象的特定功能。不能通过接口来调用不属于该接口的方法。

2） 创建 COM 对象的方法与创建 C++对象的方法不同。有多种方法可以创建 COM 对象，但所有的方法都需要使用 COM 的细节技术。Speech SDK 应用程序编程接口（API）包括许多的帮助函数和方法，它们简化了创建大部分 Speech 对象的工作。

3） 必须使用 COM 的细节技术来控制对象的生命期。

4） COM 对象不需要明确地装载。COM 对象一般包含在 DLL 中。然而，与使用普通 DLL 中的方法不同，使用 COM 对象时，不需要明确地装载 DLL 或链接静态库。每一个 COM 对象都具有一个惟一的注册标识。用该标识来创建对象时，COM 将自动地装载正确的 DLL。

5）COM 是一种二进制规范。COM 对象可用许多种编程语言来编写和调用。对于使用者来说，并不需要了解对象的源程序的任何信息。比如，Microsoft Visual Basic 编写的应用程序可以很好地调用 C++编写的 COM 对象。

下面先介绍 COM 的几个基本概念，包括对象与接口、GUID、HRESULT 类型和指针地址等。

（1）对象与接口

理解对象与接口的区别是非常重要的。通常用其主要接口的名称来称呼对象。但是，严格地说，这两个术语是不能互换使用的。

一个对象能暴露任何数量的接口。例如，所有的对象都必须暴露 IUnknown 接口，它们一般还暴露至少一个其他的接口，它们也可能暴露更多的接口。为了使用一个特定的方法，首先必须获得正确的接口指针。

多个不同的接口可以暴露同一个接口。一个接口就是一组执行特定操作的方法。接口定义只是指定了方法的调用语法和它们的一般功能。任何需要支持一组特定操作的 COM 对象都可通过暴露一个合适的接口来实现这些特定的操作。有些接口非常专业，仅仅由单一的对象来暴露。有些接口在许多场合下都非常有用，它们可由许多的对象来暴露。最极端的例子是 IUnknown 接口，所有的 COM 对象都需要暴露它。

如果一个对象暴露一个接口，它必须支持接口定义的每一个方法。但是，不同的对象实现一个特定方法的方式是不同的。不同的对象可能使用不同的算法来实现最后的结果。有时一个对象暴露一组通用的方法，但往往只需要支持其中的一部分方法。可是其他的未被支持的方法也需要能被成功调用，只是它们都只返回 E_NOTIMPL。

COM 标准要求接口一旦发布，其定义就不能再改变。例如，不能在一个已有的接口中增加一个新的方法，而必须创建一个新的接口。如果没有限制接口中必须有什么样的方法，通常的做法是在其下一代接口中包括原有接口的所有方法和其他的新方法。

但是一个接口有几代版本的情况并不常见。通常所有的版本实现完全相同的功能，只是在具体实现细节上不同。一个对象经常暴露所有版本的接口。这样做使较老版本的应用程序能继续使用对象的较老版本的接口，而较新版本的应用程序能使用较新版本接口的强大功能。一般来说，同一家族的接口具有相同的名称，在名称后加一个整数来表示不同的版本。比如，原来的接口叫做 IMyInterface，接下来的两代版本就分别叫做 IMyInterface2 和 IMyInterface3。

（2）GUID

全球惟一标识符（Globally Unique Identifiers，GUIDs）是 COM 编程模型的关键部分。从本质上来说， GUID 是一个 128 位的结构。然而，GUID 在创建时必须保证不能出现两个相同的 GUID。COM 在如下的两个方面广泛地使用 GUID：

1）惟一地标识一个特定的 COM 对象。赋予一个 COM 对象的 GUID 叫做一个类标识（class ID，CLSID）。当需要创建一个相关的 COM 对象的实例时，需要使用 CLSID。

2）惟一地标识一个特定的 COM 接口。与一个 COM 接口相关的 GUID 叫做接口标识（interface ID，IID）。当从一个对象中请求一个特定接口时，需要使用 IID。不管哪个对象暴露接口，该接口的 IID 都是相同的。

为了叙述方便，在文档中经常使用对象和接口的描述名称来引用对象和接口。虽然这样做并不会在文档中引起混乱，但是，严格地说，并不能保证不会有两个或多个不同的对象或接口具有相同的描述名。惟一的不能引起歧义的方法是用 GUID 来引用对象或接口。

虽然 GUID 是一种结构，但却经常表示为对应的字符串。最常用的 GUID 字符串形式是“{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}”，其中 x 为一个十六进制整数。

由于实际的 GUID 很长，并且很容易写错，所以一般还提供一个等价的名称。在调用 CoCreateInstance 之类的函数时，可以使用这个名称而不使用实际的 GUID 结构。通常的命名惯例是分别在对象或接口的描述名称前加上 CLSID_或 IID_作为前缀。例如，ISpVoice 接口的 CLSID 的名称是 CLSID_ISpVoice。

（3）HRESULT 类型值

所有的 COM 方法都返回一个 32 位的 HRESULT 类型的值。对大部分方法而言，HRESULT 本质上是一个包含独立的两部分信息的结构：

1）方法调用成功了还是失败了；

2）关于方法所支持操作的结果的更详细信息。

有些方法仅仅返回在 Winerror.h 中定义的标准的 HRESULT 类型值。然而，方法可以返回自己定义的 HRESULT 类型值，以提供更专用的信息。当然这样的自定义值一般会在方法的参考文档中说明。需要注意的是，参考文档可能不会列出标准的 HRESULT 类型值，但方法却可能返回标准值。

虽然 HRESULT 类型值经常用来返回错误信息，但不要将它们看成错误码。由于说明成功或失败的位在包含详细信息的数据中是分别存储的，所以 HRESULT 类型值可以包含任何数量的成功和失败代码。作为一种惯例，成功码的名称具有 S_前缀，错误码的名称具有 E_前缀。比如，最常用的成功码和错误码分别是 S_OK 和 E_FAIL。

由于 COM 方法可能返回许多不同的成功码或错误码，因此必须很小心地测试 HRESULT 类型的值。例如，假设一个方法在文档中说明成功，返回 S_OK，不成功则返回 E_FAIL。这时，请注意，该方法可能还会返回其他的成功码或错误码。下面的代码段说明了使用简单测试的危险。

```
// hr 是该方法返回的 HRESULT 类型值

if(hr == E_FAIL)
  {
    // 处理错误
  }
else
  {
    // 处理成功
  }
```

只有在该方法只返回 E_FAIL 来指示失败时，上面的测试才能正常工作。然而，该方法还可能返回一个 E_NOTIMPL 或 E_INVALIDARG 之类的错误值。上面的代码将会将它们解释为成功，这将可能导致程序的失败。

如果需要关于方法运行结果的更详细的信息，必须测试每一个相关的 HRESULT 值。但经常只关心方法是成功的还是失败的。一种可靠的测试 HRESULT 类型值说明成功还是失败的方法是利用如下的宏来判断，这些宏定义在 Winerror.h 中。

1）宏 SUCCEEDED 返回 TRUE 作为成功码，返回 FALSE 作为失败码；

2）宏 FAILED 返回 TRUE 作为失败码，返回 FALSE 作为成功码；

可以使用宏 FAILED 来修改上面的代码段：

```
// hr 是该方法返回的 HRESULT 类型值
```

```
if(FAILED(hr))
  {
    // 处理错误
  }
else
  {
    // 处理成功
  }
```

这段代码能合理地处理 E_NOTIMPL 和 E_INVALIDARG 之类的失败码。

大多数的 COM 方法返回结构化的 HRESULT 类型值，只有很少数量的方法使用 HRESULT 来返回简单的整数值，这类方法经常是成功的。如果将这类整数值传给宏 SUCCESS，该宏将总是返回 TRUE。常用的例子是 IUnknown::Release 方法，它减少一次对象的引用计数并返回当前的引用计数。将在管理对象的生命期一节中讨论引用计数的问题。

（4）指针地址

阅读一些 COM 方法的参考文档时，经常看到如下的说明：

```
HRESULT CreateDevice(
  .
  .
  .
  IDirect3DDevice8 **ppReturnedDeviceInterface
);
```

C 或 C++开发人员熟悉普通的指针，但是 COM 经常使用另外的间接层。这种间接的第二层用两个星号（**）跟着类型声明来表示。变量名一般使用“pp”作为前缀。在上面的例子中，参数 ppReturnedDeviceInterface 表示指向 IDirect3DDevice8 接口的指针的地址。

与 C++不同，不需要直接访问 COM 对象的方法，而必须获取指向方法的接口的指针。然后像调用指向 C++方法的指针一样来调用方法。例如，使用如下的语法来调用方法 IMyInterface::DoSomething method：

```
IMyInterface *pMyIface;
.
.
.
pMyIface->DoSomething(...);
```

这样做的原因是我们并不直接创建接口指针，而是必须调用不同的方法（例如上述的 CreateDevice）来创建接口指针。为了使用这种方法来获取接口指针，应声明一个指向需要的接口的指针变量，并将该指针变量的地址，即一个指针的地址，传递给该方法。当该方法返回时，该变量将指向你要求的接口，可以使用该指针来调用接口的任何方法。将在使用 COM 接口一节中更详细地讨论接口指针的问题。

## 2．创建 COM 对象

有多种方法可以创建 COM 对象。以下是在编程中最常用的两种方法。

1）直接法： 将对象的 CLSID 传给 CoCreateInstance 函数。该函数将创建对象的一个实例，并返回指向你所指定接口的指针。

2）间接法： 调用一个特殊的方法或函数来创建对象。这类方法创建对象并返回该对象的接口。使用这种方式来创建对象时，通常并不能指定需要返回的接口。

创建对象之前，必须调用 CoInitialize 函数来初始化 COM。如果使用间接法来创建对象，对象的创建方法将自动完成 COM 初始化。如果使用 CoCreateInstance 来创建对象，则必须明确地调用 CoInitialize。当完成了所有的 COM 工作后，必须调用 CoUninitialize 来卸载 COM。如果调用了 CoInitialize，则必须对应地调用一次 CoUninitialize。一般地说，需要明确，初始化 COM 的应用程序在其启动过程中初始化 COM，在其清除过程中卸载 COM。

用 CoCreateInstance 来创建一个 COM 对象的实例需要使用该对象的 CLSID。如果其 CLSID 是公开的，可以在其参考手册或相应的头文件中找到。如果其 CLSID 不是公开的，则不能使用直接法来创建该对象。

CoCreateInstance 函数有 5 个参数，一般可以按如下方式来设置其参数。

1）rclsid：将该参数设为需要创建的对象的 CLSID。

2）pUnkOuter：将该参数设为 NULL。只有在聚合对象时才需要使用该参数。参见关于聚合的讨论。

3）dwClsContext：将该参数设为 CLSCTX_INPROC_SERVER。该值说明对象是在 DLL 中实现的，将作为你的应用程序进程的一部分来运行。

4）riid：将该参数设为需要返回的接口的 IID。该函数将创建指定的对象，并通过参数 ppv 返回所请求的接口指针。

5）ppv：将该参数设为 riid 所指定的接口的指针地址。该变量应该声明为一个指向请求的接口的指针。在参数表中，该参数应被强制为（LPVOID *）类型。

例如，下面的代码段创建了 ISpVoice 对象的一个新的实例，函数返回时，m_IpVoice 变量是指向 ISpVoice 接口的指针。如果发生错误，程序将终止，并显示一个消息框。

```
CComPtr<ISpVoice> m_IpVoice = NULL;

 HRESULT hr;
hr = m_IpVoice.CoCreateInstance(CLSID_SpVoice);

if (FAILED(hr))
{
    AfxMessageBox("Error creating voice");
    return FALSE;
}
```

用间接法创建对象往往简单得多。只要将接口指针的地址传给对象的创建方法，该方法就会创建对象并返回接口指针。但间接地创建对象时，一般不能选择返回哪个接口。但是可以指定如何来创建对象。

## 3．IUnknown 接口

所有的 COM 对象都支持一个叫做 IUnknown 的接口。该接口提供了对对象的生命期的控制和检索对象的其他接口的方法。IUnknown 接口有以下 3 个方法。

1）AddRef：当一个接口或另一个应用程序与对象绑定时，对对象的引用计数加 1。

2）QueryInterface：查询对象所支持的功能，并请求指向指定的接口的指针。

3）Release：对对象的引用计数减 1。当引用计数变为 0 时，对象将被释放。

AddRef 和 Release 方法维护对象的引用计数。例如，当创建一个对象时，该对象的引用计数变为 1。每次一个函数返回一个指向该对象的接口的指针时，该函数都必须调用 AddRef 来增加其引用计数。AddRef 的每一次调用都必须与 Release 的调用相匹配。在指针被释放前必须对该指针调用 Release。当一个对象的引用计数变为 0 时，该对象将被释放，它的所有接口都将变为无效接口。

QueryInterface 方法用来确定一个对象是否支持指定的接口。如果一个对象支持一个接口，QueryInterface 返回一个指向该接口的指针。然后就可以使用该接口的方法来与对象进行通信。如果 QueryInterface 成功地返回一个指向接口的指针，它将明确地调用 AddRef 来增加引用计数。因此在释放该接口的指针之前，应用程序必须调用 Release 来减少引用计数。

## 4．使用 COM 接口

获得接口指针后，可以用该指针来访问接口的任何方法。

在许多情形中，从创建方法接收到的接口指针就是所需要的。实际上，只暴露一个除 IUnknown 之外的接口的对象是很不常见的。相反，许多的对象暴露多个接口，需要指向这些接口的多个指针。如果需要创建方法所返回的接口之外的更多的接口，则并不需要再创建一个新的对象。可以使用对象的 IUnknown::QueryInterface 方法来请求其他接口的指针。

如果使用 CoCreateInstance 来创建对象，则可以请求一个 IUnknown 接口指针，然后调用 IUnknown::QueryInterface 方法来请求需要的每一个接口。然而，当只需要一个接口时，这种方法显得很不方便。而且，如果使用不允许指定哪个接口应该返回的创建方法时，这种方法更不能工作。在实践中，经常不需要获得一个明确的 IUnknown 指针，因为所有的 COM 接口都是从 IUnknown 接口继承或扩展而来的。

扩展一个接口很像继承一个 C++类。子接口暴露父接口的所有方法，并增加一个或多个自己的方法。事实上，经常看见“继承”而不是“扩展”。需要记住的是，继承只能出现在对象的内部。应用程序不能继承或扩展另一个对象的接口，但是可以使用子接口来调用它自己或其父接口的方法。

由于所有接口都是 IUnknown 的子接口，因此可以使用已经获得的任何该对象接口的指针来调用 QueryInterface。这样做时，需要提供你请求的接口的 IID 和当方法返回时存放接口指针的指针。

## 5．管理 COM 对象的生命期

当对象被创建时，系统将分配必需的内存资源。当一个对象不再需要时，应该删除它。系统将收回它所占有的内存，以用于其他目的。对于 C++对象，应直接使用 new 和 delete 操作符来控制对象的生命期。COM 不允许直接创建或删除对象。其原因是同一对象可能被多个应用程序所使用。如果其中的一个应用程序要删除该对象，其他的应用程序就可能失败。实际上，COM 采用引用计数系统来控制对象的生命期。

对象的引用计数就是其中的接口被请求的次数。接口每被请求一次，其引用计数都将增加。当不再需要接口时，应用程序将释放该接口，并减少其引用计数。只要引用计数大于 0，对象将保留在内存中。当引用计数变为 0 时，对象将释放自己。我们不必关心对象的引用计数，只需要正确地获取和释放对象的接口，对象将具有适当的生命期。

合理地处理引用计数对 COM 编程来说是非常重要的。处理不当将导致内存泄漏。COM 编程人员最常见的错误是不释放接口。当出现这样的错误时，引用计数将永远不能变为 0，对象将不确定地保留在内存中。

只要获得一个新的接口指针，引用计数就必须调用 IUnknown::AddRef 来增加。但是，应用程序通常不需要调用该方法。如果通过调用一个对象创建方法或 IUnknown::QueryInterface 来获得接口指针，对象将自动地增加引用计数。如果用其他的方法来创建接口指针，比如拷贝已有的指针，就必须明确地调用 IUnknown::AddRef。否则，当释放原来的接口指针时，你得到的对象可能被破坏，即使你还需要使用该指针的拷贝。

不论明确地或对象自动地增加了引用计数，都必须释放接口指针。当不再需要接口指针时，调用 IUnknown::Release 来减少引用计数。一种常用的方法是，将所有的接口指针初始化

为 NULL，并在释放接口后将它们重新设为 NULL。这样可以在清除代码中测试所有的接口指针。那些不为 NULL 的接口指针就是仍然活动的，需要在退出应用程序之前释放它们。

### 6．用 C 来操作 COM 对象

虽然 C++是最常用的 COM 编程语言，有些时候也需要使用 C 语言来访问 COM 对象。这样做更加直截了当，但也要用到更加复杂的语法。

所有的方法都需要一个额外的参数，放在参数表的最前面。这个参数必须设为接口指针。而且，必须明确地引用对象的 vtable。

每个 COM 对象都有一个包含指向对象所暴露的方法的指针的列表。接口指针指向 vtable 的相应位置，该位置包含了指向你要调用的特定方法的指针。使用 C++时，不必关心 vtable，因为它在 C++中是不可见的。当然，如果需要使用 C 来调用 COM 方法，就必须包括一个明确引用 vtable 的间接的额外层。

有些组件在它们的头文件中定义了一些宏，它们能很好地解决如何正确地调用 COM。

### 7．用 ATL 来处理 COM 接口

如果使用活动模板库（Active Template Library，ATL）来处理 Microsoft Speech，为了兼容 ATL，必须首先重新声明接口。这将允许你合理地使用 CComQIPtr 类来获取指向接口的指针。

如果没有为 ATL 重新声明接口，则会得到下面的错误信息：

```
C:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE\atlbase.h(566) :
 error C2787: 'ISpVoice' : no GUID has been associated with this object
```

## （二） SAPI 接口

SAPI 提供应用程序和语音引擎之间的高层接口，它实现并隐藏了控制和管理不同语音引擎的实时操作的底层技术细节。SAPI 的结构如图 3-1 所示。

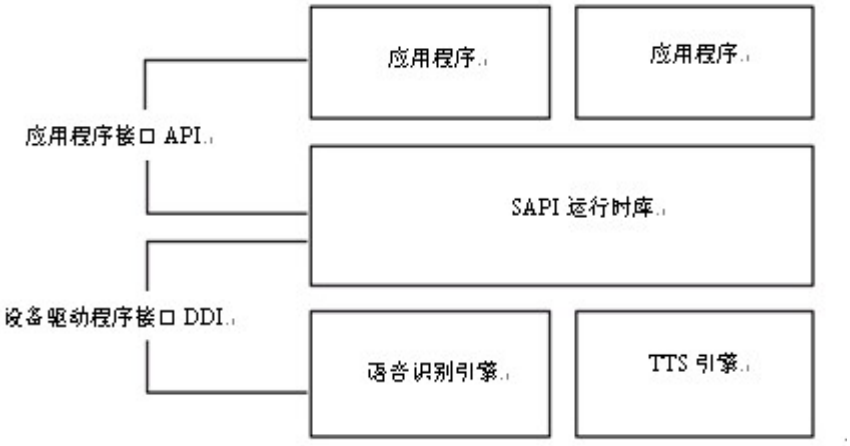



图 3-1　SAPI 的结构

最基本的语音引擎包括 Text-To-Speech（TTS）和语音识别器。 TTS 通过合成声音来朗读文本字符串和文本文件。语音识别器将人类的语音转换成可阅读的字符串和文件。

### 1．Text-To-Speech API

应用程序通过 ISpVoice 组件对象接口（Component Object Model Interface）来控制 Text-To-Speech (TTS)。一旦应用程序创建了 IspVoice 对象，调用接口 IspVoice 的方法 ISpVoice::Speak，就能产生朗读指定的文字的声音。IspVoice 接口还提供了其他一系列的方法来改变声音和其合成特征，比如控制语速的 ISpVoice::SetRate，控制输出音量的 ISpVoice::SetVolume 和改变当前语音的 ISpVoice::SetVoice。

在输入用于朗读的文字中还可插入一系列的特殊 SAPI 控制符，用来控制输出声音的实时合成特性，如语音、语调、重音、语速和音量等。合成标志文件 sapi.xsd 用来说明声音的合成特性。sapi.xsd 是一种标准 XML 格式文件，它与特定的引擎或当前正在使用的语音无关，是一种简单而功能强大的定制 TTS 语音的方法。

IspVoice::Speak 方法既能同步地（在语音播放完之后才返回）也能异步地（语音开始播放就返回，语音播放在后台处理）操作语音。 指定 SPF_ASYNC 作为播放方式时，语音异步播放。这时可调用 ISpVoice::GetStatus 方法来获取实时状态信息，如播放状态、当前播放的文字位置等。同时，既可以打断当前的播放而立即播放新的文字（需指定 SPF_PURGEBEFORESPEAK 参数），也可以在播放完当前文字之后再自动播放新的文字。

除了 IspVoice 接口以外， SAPI 还提供了许多有用的 COM 接口来实现高级的 TTS 应用程序。

（1）事件（Events）

SAPI 通过使用标准的回调机制（窗口消息、回调函数或 Win32 事件）来与应用程序传送事件。对于 TTS，事件主要用来同步语音输出。应用程序能同步处理语音输出和实时动作，比如词语分界、音位或嘴形动画分界及应用程序定制的分界等。应用程序可通过调用 IspNotifySource，IspNotifySink，IspNotifyTranslator，IspEventSink，IspEventSource 和 ISpNotifyCallbackcan 来初始化和处理这些实时事件。

（2）词典（Lexicons）

通过调用 IspContainerLexicon，IspLexicon 和 IspPhoneConverter 接口提供的方法，应用程序能为语音合成引擎设置定制的词汇发音。

（3）资源（Resources）

下面的 COM 接口用于处理 SAPI 语音数据（比如声音文件和发音词典）：

IspDataKey，IspRegDataKey，IspObjectTokenInit，IspObjectTokenCategory，IspObjectToken，IenumSpObjectTokens，IspObjectWithToken，IspResourceManager 和 IspTask。

（4）声音（Audio）

SAPI 还提供了定制声音输出到特定目标（如电话和客户硬件）的接口，包括 IspAudio，ISpMMSysAudio，IspStream，IspStreamFormat 和 IspStreamFormatConverter。

### 2．语音识别 API

正如 IspVoice 是主要的语音合成接口一样，IspRecoContext 是语音识别的主要接口。与 IspVoice 一样，它也是一种 ISpEventSource 接口，提供了为请求的语音识别事件接收通知消息的基本载体。

有两种不同的语音识别引擎（ISpRecognizer），即共享语音识别引擎（shared speech recognition engine）和进程内语音识别引擎（InProc speech recognition engine）。应用程序可以选择其中的一种。

一般推荐使用共享语音识别引擎，这种引擎能被多个应用程序共享。创建共享 IspRecognizer 的 IspRecoContext 接口很简单，应用程序只需指定参数为组件的 CLSID_SpSharedRecoContext 并调用 COM 的 CoCreateInstance 函数即可。这时，SAPI 将设置音频输入流为 SAPI 的默认音频输入流。

对于单独运行于一个系统中的大型服务器应用程序，其运行效率是很重要的。这时使用进程内语音识别引擎更合适。使用进程内语音识别引擎有 3 个步骤：首先，应用程序需指定参数为组件的 CLSID_ SpInprocRecoInstance 并调用 COM 的 CoCreateInstance 函数来创建其自己的进程内语音识别 IspRecognizer；其次，应用程序需调用 ISpRecognizer::SetInput 方法（参见 IspObjectToken 接口的说明）来设置音频输入流；最后，应用程序可调用 ISpRecognizer::CreateRecoContext 来获取 IspRecoContext 接口。

下一步需要为应用程序感兴趣的事件设置通知消息。IspRecognizer 也是一种 IspEventSource 接口，自然是一种 IspNotifySource 接口，因此，应用程序能够从其 IspRecoContext 接口中调用 IspNotifySource 的方法来指定 IspRecoContext 所需的消息应通知到何处。调用 ISpEventSource::SetInterest 方法可以设定什么样的事件需要被通知。最重要的事件是 SPEI_RECOGNITION，它标识了 IspRecognizer 已从 IspRecoContext 中识别了一些语音。Speech SDK 文档中 SPEVENTENUM 的说明提供了其他语音识别事件的详细说明。

最后需要说明的是，应用程序必须创建、装载并激活一个 IspRecoGrammar 接口。该接口从本质上说明了什么语音类型，即口述或命令和控制语法。应用程序首先应调用 ISpRecoContext::CreateGrammar 方法创建一个 IspRecoGrammar 接口。然后装载合适的语法，调用 ISpRecoGrammar::LoadDictation 方法可装载口述语法，调用 ISpRecoGrammar::LoadCmdxxx 方法可装载命令和控制语法。最后，为了激活语法并启动识别，应用程序应该调用 ISpRecoGrammar::SetDictationState 方法设置口述状态，或者调用 ISpRecoGrammar::SetRuleState 方法或 ISpRecoGrammar::SetRuleIdState 方法设置命令和控制状态。

当应用程序通过请求的通知机制得到通知消息时，SPEVENT 结构的 lParam 成员包含了一个 IspRecoResult 接口，应用程序能从中确定用 IspRecoContext 中的哪个 IspRecoGrammar 接口已识别了什么语音。

无论共享的还是进程内的 IspRecognizer 接口都能拥有多个与其关联的 IspRecoContexts 接口，并且每一个接口都能通过自己的事件通知方式得到相应的消息。可以从一个 IspRecoContext 接口中创建多个 IspRecoGrammars 接口，不同的接口可用于识别不同的语音类型。

### （三）安装 Speech SDK

进行 Text-To-Speech 编程之前，必须先下载 Miscrosoft Speech SDK，并将它安装到你的系统中。

Miscrosoft Speech SDK 的下载网址是 http://www.microsoft.com/speech。至笔者编写本章时为止，最新的 Speech SDK 版本是 5.1 版。下载的 speechsdk51.exe 是一个可执行的文件包压缩文件。运行它，将安装文件释放到一个临时目录中，执行其中的 Microsoft Speech SDK 5.1.msi，将 Speech SDK 安装到相应的目录中。一般选用默认的安装目录（C:\Program Files\Microsoft Speech SDK 5.1）。

Speech SDK 支持的默认语言是英语，即安装 Speech SDK 后，系统还只能支持英语的语音。要使系统支持中文和日文语音，还需要下载安装相应的语言包。从相同的网址中下载语言包 speechsdk51LangPack.exe。运行它，将安装文件释放到一个临时目录中，执行其中的 Microsoft Speech SDK 5.1 Language Pack.msi，将中、日文支持安装到系统中。

安装好 Speech SDK 后，语音控制程序将被添加到系统的控制面板中。利用该控制程序可以设置语音识别和文字–语音转换的各项属性，包括语言/语音、语速和输入设备等，如图 3-2 所示。


图 3-2　控制面板中的语音控制程序

## 第二部分 Text-To-Speech 编程技术

至此已做好了编写语音程序的准备工作，可以开始编写语音程序了。下面首先介绍文本–语音转换的编程技术。

### （一）构造 CText2Speech 类

为了便于使用 Speech SDK 提供的文本–语音转换 COM 接口，笔者编写了一个类 CText2Speech，其中封装了文本–语音转换 COM 接口的基本方法。借助该类来编写文本–语音转换程序非常方便。

先来讨论该 CText2Speech 类的设计，其定义文件列举如下：

```
//////////////////////////////////////////////////////////////
// active speech engine
//
#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include "sapi.h"
#include <sphelper.h>

//////////////////////////////////////////////////////////////
// speech message
//
#define WM_TTSEVENT  WM_USER+101

//////////////////////////////////////////////////////////////
// text-to-speech class
//
class CText2Speech
{
public:
    CText2Speech();
    virtual ~CText2Speech();

    // initialize
    BOOL Initialize(HWND hWnd = NULL);
    void Destroy();

    // speak
    HRESULT Speak(const WCHAR *pwcs, DWORD dwFlags = SPF_DEFAULT);
    HRESULT Pause();
    HRESULT Resume();
```

```
    // rate
    HRESULT SetRate(long lRateAdjust);
    HRESULT GetRate(long* plRateAdjust);

    // volume
    HRESULT SetVolume(USHORT usVolume);
    HRESULT GetVolume(USHORT* pusVolume);

    // voice
    ULONG  GetVoiceCount();
    HRESULT GetVoice(WCHAR **ppszDescription, ULONG lIndex = -1);
    HRESULT SetVoice(WCHAR **ppszDescription);

    // error string
    CString GetErrorString()
    {
        return m_sError;
    }

    // interface
    CComPtr<ISpVoice> m_IpVoice;

private:
    CString m_sError;
};
```

文件的开始几行语句：

```
#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include "sapi.h"
#include <sphelper.h>
```

用于使我们的代码能操作 Speech SDK 中的相关的接口、函数和常量。

Speech SDK 支持事件。为了与窗口交互，这里在类中定义了消息 WM_TTSEVENT。当发生 Speech 事件时，向相应的窗口发送 WM_TTSEVENT 消息。在窗口中响应该消息就响应了相应的事件。

CText2Speech 类中定义了一个操作 Text-To-Speech 引擎的接口指针 m_IpVoice，作为数据成员，其定义如下：

```
CComPtr<ISpVoice> m_IpVoice;
```

几乎所有的 Text-To-Speech 操作都是借助该指针来调用 IspVoice 接口的方法而实现的。CText2Speech 类实现了如下的方法：

```
// 初始化和释放函数
BOOL Initialize(HWND hWnd = NULL);
void Destroy();
```

```
// 语音操作函数
HRESULT Speak(const WCHAR *pwcs, DWORD dwFlags = SPF_DEFAULT);
HRESULT Pause();
HRESULT Resume();

// 语速函数
HRESULT SetRate(long lRateAdjust);
HRESULT GetRate(long* plRateAdjust);

// 音量函数
HRESULT SetVolume(USHORT usVolume);
HRESULT GetVolume(USHORT* pusVolume);

// 语言函数
ULONG  GetVoiceCount();
HRESULT GetVoice(WCHAR **ppszDescription, ULONG lIndex = -1);
HRESULT SetVoice(WCHAR **ppszDesc);

// 获取错误信息函数
CString GetErrorString()
```

CText2Speech 类的构造函数用于初始化 Text-To-Speech 引擎接口指针 m_IpVoice 和错误字符串；析构函数则调用释放引擎的 Destroy()函数释放语音引擎，其代码如下：

```
CText2Speech::CText2Speech()
{
    m_IpVoice = NULL;
    m_sError=_T("");
}
CText2Speech::~CText2Speech()
{
    Destroy();
}
```

初始化函数 Initialize 首先初始化 COM 库，并调用 CoCreateInstance 方法初始化语音引擎。然后设置必须响应的引擎事件，并指定响应事件消息的窗口句柄。该窗口句柄是作为函数的参数传入的。Initialize 函数的代码如下：

```
BOOL CText2Speech::Initialize(HWND hWnd)
{
  if (FAILED(CoInitialize(NULL)))
  {
        m_sError=_T("Error intialization COM");
        return FALSE;
  }

  HRESULT hr;
  hr = m_IpVoice.CoCreateInstance(CLSID_SpVoice);

  if (FAILED(hr))
```

```
    {
          m_sError=_T("Error creating voice");
          return FALSE;
    }

    hr = m_IpVoice->SetInterest(SPFEI(SPEI_VISEME), SPFEI(SPEI_VISEME));
    if (FAILED(hr))
    {
         m_sError=_T("Error creating interest...seriously");
         return FALSE;
    }

    if (::IsWindow(hWnd))
    {
       hr = m_IpVoice->SetNotifyWindowMessage(hWnd, WM_TTSEVENT, 0, 0);
       if (FAILED(hr))
       {
        m_sError=_T("Error setting notification window");
        return FALSE;
       }
    }

    return TRUE;
}
```

释放函数则释放语音引擎接口和 COM 库，其代码如下：

```
void CText2Speech::Destroy()
{
    if (m_IpVoice)
           m_IpVoice.Release();
    CoUninitialize();
}
```

语音、语速、音量函数都是通过 m_IpVoice 成员直接调用 ISpVoice 接口的相关方法来实现的：

```
HRESULT CText2Speech::Speak(const WCHAR *pwcs, DWORD dwFlags)
{
    return m_IpVoice->Speak(pwcs, dwFlags, NULL);
}

HRESULT CText2Speech::Pause()
{
    return m_IpVoice->Pause();
}

HRESULT CText2Speech::Resume()
{
    return m_IpVoice->Resume();
```

```
}

// rate
HRESULT CText2Speech::SetRate(long lRateAdjust)
{
    return m_IpVoice->SetRate(lRateAdjust);
}

HRESULT CText2Speech::GetRate(long* plRateAdjust)
{
    return m_IpVoice->GetRate(plRateAdjust);
}

// volume
HRESULT CText2Speech::SetVolume(USHORT usVolume)
{
    return m_IpVoice->SetVolume(usVolume);
}

HRESULT CText2Speech::GetVolume(USHORT* pusVolume)
{
    return m_IpVoice->GetVolume(pusVolume);
}
```

语言函数的实现比较复杂。由于 IspVoice 接口提供的语言函数，都只与抽象的语音语言接口 ISpObjectToken 相关，而我们能看到的却是语音语言的描述，比如，通过控制面板的语音程序所能见到的就是语音语言的描述。因此，笔者设计了直接对语音语言进行操作的语言函数，包括获取系统中已安装的语音语言数目，设置指定的语音语言，获取指定的语音语言描述（包括当前设定的语音语言）。它们的代码如下：

```
ULONG CText2Speech::GetVoiceCount()
{
    HRESULT                             hr = S_OK;
    CComPtr<ISpObjectToken>             cpVoiceToken;
    CComPtr<IEnumSpObjectTokens>        cpEnum;
    ULONG                               ulCount = -1;

    //Enumerate the available voices
    hr = SpEnumTokens(SPCAT_VOICES, NULL, NULL, &cpEnum);
    if(FAILED(hr))
    {
        m_sError = _T("Error to enumerate voices");
        return -1;
    }

    //Get the number of voices
    hr = cpEnum->GetCount(&ulCount);
    if(FAILED(hr))
```

```
    {
        m_sError = _T("Error to get voice count");
        return -1;
    }

    return ulCount;
}

HRESULT CText2Speech::GetVoice(WCHAR **ppszDescription, ULONG lIndex)
{
    HRESULT                             hr = S_OK;
    CComPtr<ISpObjectToken>             cpVoiceToken;
    CComPtr<IEnumSpObjectTokens>        cpEnum;
    ULONG                               ulCount = 0;

    if (lIndex == -1)
    {
        // current voice
        //

        hr = m_IpVoice->GetVoice(&cpVoiceToken);
        if(FAILED(hr))
        {
            m_sError = _T("Error to get current voice");
            return hr;
        }

        SpGetDescription(cpVoiceToken, ppszDescription);
        if(FAILED(hr))
        {
            m_sError = _T("Error to get current voice description");
            return hr;
        }
    }
    else
    {
        // else other voices, we should enumerate the voice list first
        //

        //Enumerate the available voices
        hr = SpEnumTokens(SPCAT_VOICES, NULL, NULL, &cpEnum);
        if(FAILED(hr))
        {
            m_sError = _T("Error to enumerate voices");
            return hr;
        }
```

```
        //Get the number of voices
        hr = cpEnum->GetCount(&ulCount);
        if(FAILED(hr))
        {
            m_sError = _T("Error to voice count");
            return hr;
        }

        // range control
        ASSERT(lIndex >= 0);
        ASSERT(lIndex < ulCount);

        // Obtain specified voice id
        ULONG l = 0;
        while (SUCCEEDED(hr))
        {
            cpVoiceToken.Release();
            hr = cpEnum->Next( 1, &cpVoiceToken, NULL );
            if(FAILED(hr))
            {
                m_sError = _T("Error to get voice token");
                return hr;
            }

            if (l == lIndex)
            {
                hr = SpGetDescription(cpVoiceToken, ppszDescription);
                if(FAILED(hr))
                {
                    m_sError = _T("Error to get voice description");
                    return hr;
                }
                break;
            }

            l++;
        }
    }

    return hr;
}

HRESULT CText2Speech::SetVoice(WCHAR **ppszDescription)
{
    HRESULT                             hr = S_OK;
    CComPtr<ISpObjectToken>             cpVoiceToken;
    CComPtr<IEnumSpObjectTokens>        cpEnum;
```

```
ULONG                                   ulCount = 0;

//Enumerate the available voices
hr = SpEnumTokens(SPCAT_VOICES, NULL, NULL, &cpEnum);
if(FAILED(hr))
{
    m_sError = _T("Error to enumerate voices");
    return hr;
}

//Get the number of voices
hr = cpEnum->GetCount(&ulCount);
if(FAILED(hr))
{
    m_sError = _T("Error to voice count");
    return hr;
}

// Obtain specified voice id
while (SUCCEEDED(hr) && ulCount--)
{
    cpVoiceToken.Release();
    hr = cpEnum->Next( 1, &cpVoiceToken, NULL );
    if(FAILED(hr))
    {
        m_sError = _T("Error to voice token");
        return hr;
    }

    WCHAR *pszDescription1;
    hr = SpGetDescription(cpVoiceToken, &pszDescription1);
    if(FAILED(hr))
    {
        m_sError = _T("Error to get voice description");
        return hr;
    }

    if (! wcsicmp(pszDescription1, *ppszDescription))
    {
        hr = m_IpVoice->SetVoice(cpVoiceToken);
        if(FAILED(hr))
        {
            m_sError = _T("Error to set voice");
            return hr;
        }

        break;
```

```
        }
    }

    return hr;
}
```

如果不想处理有关语言的细节问题，只想显示和选择系统可提供的语音引擎，则可以直接调用 Speech SDK 提供的两个帮助函数 SpInitTokenComboBox 和 SpInitTokenListBox 来实现语音语言的显示和选择，其代码如下：

```
HRESULT SpInitTokenComboBox(
   HWND    hwnd,
   const WCHAR* pszCatName,
   const WCHAR* pszRequiredAttrib = NULL,
   const WCHAR* pszOptionalAttrib = NULL
);
HRESULT SpInitTokenListBox(
   HWND    hwnd,
   const WCHAR* pszCatName,
   const WCHAR* pszRequiredAttrib = NULL,
   const WCHAR* pszOptionalAttrib = NULL
);
```

CText2Speech 类具有很好的错误处理机制。一旦调用某个函数发生了错误，响应的错误信息都将存放在 m_sError 数据成员中。可通过 GetErrorString 函数来获得错误描述。

### （二）示例：用 CText2Speech 类编制文字朗读程序

下面使用 CText2Speech 类来编写一个文字朗读程序 Reciter，其界面如图 3-3 所示。

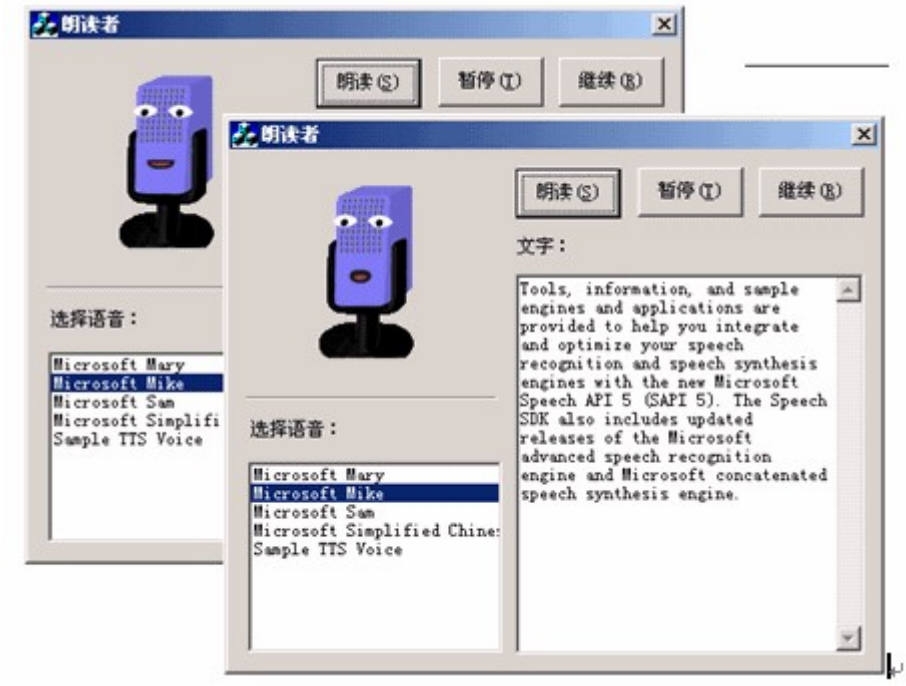



图 3-3　朗读者程序界面

用 Visual C++编制 Reciter 的步骤和要点如下：

1）使用 AppWizard 生成一个基于对话框的项目 Reciter。

2）将 Text2Speech.H，Text2Speech.CPP 增加到 Reciter 项目中。

3）在资源编辑器中编辑好响应的控件。

4）用 ClassWizard 为控件在 CReciterDlg 类中生成相应的成员。

5）修改 ReciterDlg.h 文件，为类 CReciterDlg 增加相应的变量和函数。

6）用 ClassWizard 为 CReciterDlg 类添加对控件和消息的响应函数。ReciterDlg.h 的代码如下所示：

```
#include "Text2Speech.h"

// CONTANTS OF MOUTH
#define CHARACTER_WIDTH     128
#define CHARACTER_HEIGHT    128
#define WEYESNAR            14            // eye positions
#define WEYESCLO            15

/////////////////////////////////////////////////////////////////////////
// Mouth Mapping Array (from Microsoft's TTSApp Example)
//
const int g_iMapVisemeToImage[22] =
{
 0, // SP_VISEME_0 = 0,// Silence
11, // SP_VISEME_1,    // AE, AX, AH
11, // SP_VISEME_2,    // AA
11, // SP_VISEME_3,    // AO
10, // SP_VISEME_4,    // EY, EH, UH
11, // SP_VISEME_5,    // ER
9,  // SP_VISEME_6,    // y, IY, IH, IX
2,  // SP_VISEME_7,    // w, UW
13, // SP_VISEME_8,    // OW
9,  // SP_VISEME_9,    // AW
12, // SP_VISEME_10,   // OY
11, // SP_VISEME_11,   // AY
9,  // SP_VISEME_12,   // h
3,  // SP_VISEME_13,   // r
6,  // SP_VISEME_14,   // l
7,  // SP_VISEME_15,   // s, z
8,  // SP_VISEME_16,   // SH, CH, JH, ZH
5,  // SP_VISEME_17,   // TH, DH
4,  // SP_VISEME_18,    // f, v
7,  // SP_VISEME_19,   // d, t, n
9,  // SP_VISEME_20,   // k, g, NG
1   // SP_VISEME_21,   // p, b, m
};

/////////////////////////////////////////////////////////////////////////
// CReciterDlg dialog
```

```
class CReciterDlg : public CDialog
{
// Construction
public:
    CReciterDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CReciterDlg)
    enum { IDD = IDD_RECITER_DIALOG };
    CStatic       m_cMouth;
    CListBox   m_ListVoices;
    CString       m_strText;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CReciterDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

    CText2Speech m_Text2Speech;

    void InitText2Speech();
    void InitMouthImageList();

private:
    CImageList m_cMouthList;
    Int m_iMouthBmp;
     CRect m_cMouthRect;

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CReciterDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnButtonSpeak();
    afx_msg void OnSelchangeList1();
    afx_msg void OnButtonStop();
    afx_msg void OnButtonResume();
    //}}AFX_MSG
    afx_msg LRESULT OnMouthEvent(WPARAM, LPARAM);
```

```
    DECLARE_MESSAGE_MAP()
};
```

注意，在 CReciterDlg 类中定义了一个 CText2Speech 类的对象。

7）在 ReciterDlg.cpp 中编写各成员函数的代码。成员函数 InitText2Speech 用于初始化语音引擎并找出系统中的所有语音语言，显示在语音列表中，其代码如下所示：

```
void CReciterDlg::InitText2Speech()
{
    if (! m_Text2Speech.Initialize(m_hWnd))
            AfxMessageBox(m_Text2Speech.GetErrorString());

    long lCount = m_Text2Speech.GetVoiceCount();
    WCHAR* pszID;
    for (long l=0; l<lCount; ++l)
    {
            m_Text2Speech.GetVoice(&pszID, l);
            m_ListVoices.AddString(CString(pszID));
    }
    m_Text2Speech.GetVoice(&pszID, -1);
    m_ListVoices.SelectString(0, CString(pszID));
}
```

成员函数 InitMouthImageList 用于初始化朗读者图像列表，其代码如下：

```
void CReciterDlg::InitMouthImageList()
{
    m_cMouth.GetClientRect(&m_cMouthRect);
    m_cMouth.ClientToScreen(&m_cMouthRect);
    ScreenToClient(&m_cMouthRect);
    m_cMouth.ShowWindow(SW_HIDE);

    CBitmap bmp;
    m_cMouthList.Create(CHARACTER_WIDTH, CHARACTER_HEIGHT, ILC_COLOR32 |
                        ILC_MASK, 1, 0);

    bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICFULL));
        m_cMouthList.Add(&bmp, RGB(255,0,255));
        bmp.Detach();

        bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH2));
    m_cMouthList.Add(&bmp, RGB(255,0,255));
    bmp.Detach();

    bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH3));
    m_cMouthList.Add(&bmp, RGB(255,0,255));
    bmp.Detach();

    bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH4));
    m_cMouthList.Add(&bmp, RGB(255,0,255));
```

```
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH5));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH6));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH7));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH8));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH9));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH10));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH11));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH12));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICMOUTH13));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICEYESNAR));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

bmp.LoadBitmap(MAKEINTRESOURCE(IDB_MICEYESCLO));
m_cMouthList.Add(&bmp, RGB(255,0,255));
bmp.Detach();

m_cMouthList.SetOverlayImage(1, 1);
```

```
    m_cMouthList.SetOverlayImage(2, 2);
    m_cMouthList.SetOverlayImage(3, 3);
    m_cMouthList.SetOverlayImage(4, 4);
    m_cMouthList.SetOverlayImage(5, 5);
    m_cMouthList.SetOverlayImage(6, 6);
    m_cMouthList.SetOverlayImage(7, 7);
    m_cMouthList.SetOverlayImage(8, 8);
    m_cMouthList.SetOverlayImage(9, 9);
    m_cMouthList.SetOverlayImage(10, 10);
    m_cMouthList.SetOverlayImage(11, 11);
    m_cMouthList.SetOverlayImage(12, 12);
    m_cMouthList.SetOverlayImage(13, 13);
    m_cMouthList.SetOverlayImage(14, WEYESNAR);
    m_cMouthList.SetOverlayImage(15, WEYESCLO);
}
```

成员函数 InitText2Speech 和 InitMouthImageList 都在 OnInitDialog 函数中被调用。响应朗读、暂停和继续的函数较简单，其代码如下：

```
void CReciterDlg::OnButtonSpeak()
{
    // TODO: Add your control notification handler code here
    UpdateData();

    if (FAILED(m_Text2Speech.Speak(m_strText.AllocSysString(), SPF_ASYNC)))
        AfxMessageBox(m_Text2Speech.GetErrorString());
}

void CReciterDlg::OnButtonStop()
{
    // TODO: Add your control notification handler code here
    m_Text2Speech.Pause();

}

void CReciterDlg::OnButtonResume()
{
    // TODO: Add your control notification handler code here
    m_Text2Speech.Resume();

}
```

选择列表中的语音语言并设为当前的语音设置是通过响应该列表的 LBN_SELCHANGE 消息来实现的。其消息响应函数为：

```
void CReciterDlg::OnSelchangeList1()
{
    CString sVoice;
    int nIndex = m_ListVoices.GetCurSel();
    m_ListVoices.GetText(nIndex, sVoice);
```

```
    BSTR bstr = sVoice.AllocSysString();
    if (FAILED(m_Text2Speech.SetVoice(&bstr)))
        AfxMessageBox(m_Text2Speech.GetErrorString());
}
```

为了显示朗读者肖像，需要将其肖像序列在 OnPaint 函数中画出来：

```
void CReciterDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        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;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CPaintDC dc(this); // device context for painting
        CDialog::OnPaint();

        // Draw into memory DC
        m_cMouthList.Draw(&dc, 0, m_cMouthRect.TopLeft(),
                          INDEXTOOVERLAYMASK(m_iMouthBmp));

        if (m_iMouthBmp % 6 == 2)
        {
            m_cMouthList.Draw(&dc, WEYESNAR, m_cMouthRect.TopLeft(), 0 );
        }
        else if (m_iMouthBmp % 6 == 5)
        {
            m_cMouthList.Draw(&dc, WEYESCLO, m_cMouthRect.TopLeft(), 0 );
        }
    }
}
```

使朗读者的嘴型随着所朗读的文字变化的关键是响应 CText2Speech 类定义的消息 WM_TTSEVENT。该消息指示出一个语音朗读事件已完成，即已发出某个音节。通过 CSpEvent

的 GetFrom 函数可以获得当前的事件信息，eEventId 成员中记录了朗读的音节的代号。数组 g_iMapVisemeToImage 定义了音节代码和对应嘴形位图序列号的对应关系：

```
const int g_iMapVisemeToImage[22] =
{
 0,  // SP_VISEME_0 = 0, // Silence
11, // SP_VISEME_1,       // AE, AX, AH
11, // SP_VISEME_2,       // AA
11, // SP_VISEME_3,       // AO
10, // SP_VISEME_4,       // EY, EH, UH
11, // SP_VISEME_5,       // ER
9,  // SP_VISEME_6,       // y, IY, IH, IX
2,  // SP_VISEME_7,       // w, UW
13, // SP_VISEME_8,       // OW
9,  // SP_VISEME_9,       // AW
12, // SP_VISEME_10,      // OY
11, // SP_VISEME_11,      // AY
9,  // SP_VISEME_12,      // h
3,  // SP_VISEME_13,      // r
6,  // SP_VISEME_14,      // l
7,  // SP_VISEME_15,      // s, z
8,  // SP_VISEME_16,      // SH, CH, JH, ZH
5,  // SP_VISEME_17,      // TH, DH
4,  // SP_VISEME_18,      // f, v
7,  // SP_VISEME_19,      // d, t, n
9,  // SP_VISEME_20,      // k, g, NG
1   // SP_VISEME_21,      // p, b, m
};
```

为了响应消息 WM_TTSEVENT，需要添加相应的消息响应函数：

```
BEGIN_MESSAGE_MAP(CReciterDlg, CDialog)
    //{{AFX_MSG_MAP(CReciterDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_SPEAK, OnButtonSpeak)
    ON_LBN_SELCHANGE(IDC_LIST1, OnSelchangeList1)
    ON_BN_CLICKED(IDC_BUTTON_STOP, OnButtonStop)
    ON_BN_CLICKED(IDC_BUTTON_RESUME, OnButtonResume)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_TTSEVENT, OnMouthEvent)
END_MESSAGE_MAP()

LRESULT CReciterDlg::OnMouthEvent(WPARAM wParam, LPARAM lParam)
{
    CSpEvent event;

    while (event.GetFrom(m_Text2Speech.m_IpVoice) == S_OK) {
```

```
        switch (event.eEventId) {
            case SPEI_VISEME:
                m_iMouthBmp = g_iMapVisemeToImage[event.Viseme()];
                InvalidateRect(m_cMouthRect, false);
                break;
        }
    }

    return 0;
}
```

函数 OnMouthEvent 分析出当前事件，如果是 SPEI_VISEME 事件，则取得所读音节的嘴形位图序号，并重画朗读者的嘴部位置。

8）为了调用 Speech 引擎，应该在 Microsoft Visual C++编程环境中设置好相应的 include 和 lib 设置，如下所述。

① 设置 include 路径

- 通过 Project→Settings 菜单项打开 Project Settings 对话框；
- 点击 C/C++项；
- 在 Category 下拉列表中选取 Preprocessor；
- 在“Additional include directories”编辑框中输入安装 Speech SDK 的 include 的路径，默认的路径是 C:\Program Files\Microsoft Speech SDK 5.1\Include。

② 设置 lib 信息

- 通过 Project→Settings 菜单项打开 Project Settings 对话框；
- 选择 Link 项；
- 在 Category 下拉列表中选取 Input 项；
- 在“Additional library path”编辑框中输入安装 Speech SDK 的 lib 的路径，默认的路径是 C:\Program Files\Microsoft Speech SDK 5.1\ Lib\i386；
- 将“sapi.lib”输入“Object/library modules”所标识的编辑框中。

9）编译连接该项目，你就可欣赏朗读者的风采了。

# Microsoft speech SDK SR 编程介绍

## Speech Recognition 编程技术

语音识别编程涉及 IspRecognizer，IspRecoContext 和 ISpRecoGrammar 等多个语音识别引擎接口。下面先来设计一个操作语音识别的类 CSpeechRecognition，然后基于该类来实现一个实例。

### （一）构造 CSpeechRecognition 类

CSpeechRecognition 类封装了语音识别操作所需调用的几个接口，使用它进行语音识别编程很方便，也很简洁。

CSpeechRecognition 类的定义如下：

```
//////////////////////////////////////////////////////////////
// active speech engine
//
#include <atlbase.h>
extern CComModule _Module;
#include <atlcom.h>
#include <sapi.h>
#include <sphelper.h>
#include <spuihelp.h>

//////////////////////////////////////////////////////////////
// speech message
//
#define WM_SREVENT   WM_USER+102

class CSpeechRecognition
{
public:
    CSpeechRecognition();
    virtual ~CSpeechRecognition();

    // initialize
    BOOL Initialize(HWND hWnd = NULL, BOOL bIsShared = TRUE);
    void Destroy();

    // start and stop
    BOOL Start();
    BOOL Stop();
    BOOL IsDictationOn()
    {
        return m_bOnDictation;
    }
```

```
    // event handler
    void GetText(WCHAR **ppszCoMemText, ULONG ulStart = 0, ULONG nlCount = -1);

    // voice training
    HRESULT VoiceTraining(HWND hWndParent);

    // microphone setup
    HRESULT MicrophoneSetup(HWND hWndParent);

    // token list
    HRESULT InitTokenList(HWND hWnd, BOOL bIsComboBox = FALSE);

    // error string
    CString GetErrorString()
    {
        return m_sError;
    }

    // interface
      CComPtr<ISpRecognizer> m_cpRecoEngine;   // SR engine
      CComPtr<ISpRecoContext> m_cpRecoCtxt;    //Recognition contextfor dictation
      CComPtr<ISpRecoGrammar> m_cpDictationGrammar;  // Dictation grammar

private:
    CString m_sError;
     BOOL    m_bOnDictation;
};
```

其中定义的消息 WM_SREVENT 用于指示语音识别事件，该消息将通知到初始化函数指定的响应窗口。

类中定义了 3 个接口指针 m_cpRecoEngine，m_cpRecoCtxt 和 m_cpDictationGrammar，分别用于引用语音识别引擎的 3 个重要接口 IspRecognizer，ISpRecoContext 和 IspRecoGrammar。

初始化函数 Initialize 设定了语音识别引擎的基本工作环境，包括引擎、识别上下文、语法、音频和事件等的初始化：

```
BOOL CSpeechRecognition::Initialize(HWND hWnd, BOOL bIsShared)
{
    // com library
    if (FAILED(CoInitialize(NULL)))
    {
        m_sError=_T("Error intialization COM");
        return FALSE;
    }

    // SR engine
    HRESULT hr = S_OK;
    if (bIsShared)
```

```
{
    // Shared reco engine.
    // For a shared reco engine, the audio gets setup automatically
    hr = m_cpRecoEngine.CoCreateInstance( CLSID_SpSharedRecognizer );
}
else
{
    hr = m_cpRecoEngine.CoCreateInstance(CLSID_SpInprocRecognizer);

}

// RecoContext
if( SUCCEEDED( hr ) )
{
    hr = m_cpRecoEngine->CreateRecoContext( &m_cpRecoCtxt );
}

// Set recognition notification for dictation
if (SUCCEEDED(hr))
{
    hr = m_cpRecoCtxt->SetNotifyWindowMessage( hWnd, WM_SREVENT, 0, 0 );
}

if (SUCCEEDED(hr))
{
    // when the engine has recognized something
    const ULONGLONG ullInterest = SPFEI(SPEI_RECOGNITION);
    hr = m_cpRecoCtxt->SetInterest(ullInterest, ullInterest);
}

// create default audio object
CComPtr<ISpAudio> cpAudio;
hr = SpCreateDefaultObjectFromCategoryId(SPCAT_AUDIOIN, &cpAudio);

// set the input for the engine
hr = m_cpRecoEngine->SetInput(cpAudio, TRUE);
hr = m_cpRecoEngine->SetRecoState( SPRST_ACTIVE );

// grammar
if (SUCCEEDED(hr))
{
    // Specifies that the grammar we want is a dictation grammar.
    // Initializes the grammar (m_cpDictationGrammar)
    hr = m_cpRecoCtxt->CreateGrammar( 0, &m_cpDictationGrammar );
}
if  (SUCCEEDED(hr))
{
```

```
        hr = m_cpDictationGrammar->LoadDictation(NULL, SPLO_STATIC);
    }
    if (SUCCEEDED(hr))
    {
        hr = m_cpDictationGrammar->SetDictationState( SPRS_ACTIVE );
    }
    if (FAILED(hr))
    {
        m_cpDictationGrammar.Release();
    }

    return (hr == S_OK);
}
```

释放函数 Destroy 被类的析构函数调用，释放了类所引用的所有接口：

```
void CSpeechRecognition::Destroy()
{
    if (m_cpDictationGrammar)
        m_cpDictationGrammar.Release();
    if (m_cpRecoCtxt)
        m_cpRecoCtxt.Release();
    if (m_cpRecoEngine)
        m_cpRecoEngine.Release();
    CoUninitialize();
}
```

函数 Start 和 Stop 用来控制开始和停止接受及识别语音，它们通过调用引擎接口的 SetRecoState 方法来实现：

```
BOOL CSpeechRecognition::Start()
{
    if (m_bOnDictation)
        return TRUE;

        HRESULT hr = m_cpRecoEngine->SetRecoState( SPRST_ACTIVE );
    if (FAILED(hr))
          return FALSE;

    m_bOnDictation = TRUE;
    return TRUE;
}

BOOL CSpeechRecognition::Stop()
{
    if (! m_bOnDictation)
        return TRUE;

      HRESULT hr = m_cpRecoEngine->SetRecoState( SPRST_INACTIVE );
```

```
    if (FAILED(hr))
            return FALSE;

    m_bOnDictation = FALSE;
    return TRUE;
}
```

函数 GetText 是获取从语音中已识别出的文字的关键，应该在响应识别事件/消息的响应函数中调用，其代码如下所示。

```
void CSpeechRecognition::GetText(WCHAR **ppszCoMemText, ULONG ulStart, ULONG nlCount)
{
    USES_CONVERSION;
    CSpEvent event;

    // Process all of the recognition events
    while (event.GetFrom(m_cpRecoCtxt) == S_OK)
    {
        switch (event.eEventId)
        {
            case SPEI_RECOGNITION:
                // There may be multiple recognition results, so get all of them
                {
                    HRESULT hr = S_OK;
                    if (nlCount == -1)
                        event.RecoResult()->GetText(SP_GETWHOLEPHRASE,
                          SP_GETWHOLEPHRASE, TRUE, ppszCoMemText, NULL);
                    else
                    {
                        ASSERT(nlCount > 0);
                        event.RecoResult()->GetText(ulStart, nlCount, FALSE,
                            ppszCoMemText, NULL);
                    }
                }
                break;
        }
    }
}
```

函数 InitTokenList 调用 SpInitTokenComboBox 和 SpInitTokenListBox 函数来实现语音语言在列表或组合列表中的列表显示和选择：

```
HRESULT CSpeechRecognition::InitTokenList(HWND hWnd, BOOL bIsComboBox)
{
    if (bIsComboBox)
        return SpInitTokenComboBox(hWnd, SPCAT_RECOGNIZERS);
    else
        return SpInitTokenListBox(hWnd, SPCAT_RECOGNIZERS);
}
```

语音识别涉及语音的输入，通常用话筒来输入语音。进行语音识别前，需要判断话筒的位置和设置是否合理，以保证语音识别引擎能获得有效的语音输入。函数 MicrophoneSetup 调用语音识别引擎接口的 DisplayUI 方法来显示一个设置话筒的向导，如图 4-1 所示。示例代码如下所示：

```
HRESULT CSpeechRecognition::MicrophoneSetup(HWND hWndParent)
{
    return m_cpRecoEngine->DisplayUI(hWndParent, NULL, SPDUI_MicTraining,
                                     NULL, 0);
}
```

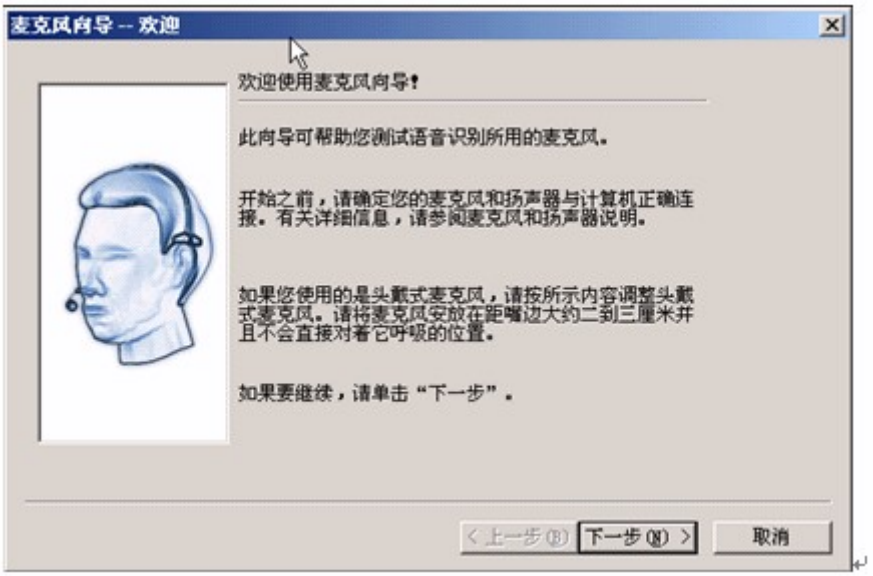



图 4-1　话筒设置向导

语音训练是语音识别的重要基础，为了获得期望的识别效果，必须进行语音训练，以让语音识别引擎熟悉说话者的口音。函数 VoiceTraining 调用语音识别引擎接口的 DisplayUI 方法来显示一个语音训练向导，如图 4-2 所示。示例代码如下所示：

```
HRESULT CSpeechRecognition::VoiceTraining(HWND hWndParent)
{
    return m_cpRecoEngine->DisplayUI(hWndParent, NULL, SPDUI_UserTraining,
                                     NULL, 0);
}
```

图 4-2 语音训练向导

与 CText2Speech 类似，CSpeechRecognition 类也提供错误处理机制，由 GetErrorString 函数可以获得错误信息。

## （二） 示例：用 CSpeechRecognition 类编制听写程序

使用 CSpeechRecognition 类来编写语音识别程序很简单，下面让我们实现一个听写程序 Stenotypist，其界面如图 4-3 所示。


图 4-3 听写者程序界面

用 VisualC++编制 Stenotypist 的步骤和要点如下：

1）使用 AppWizard 生成一个基于对话框的项目 Stenotypist；

2）将 SpeechRecognition.H，SpeechRecognition.CPP 增加到 Stenotypist 项目中；

3）在资源编辑器中编辑好响应的控件；

4）用 ClassWizard 为控件在 CStenotypistDlg 类中生成相应的成员；

5）修改 StenotypistDlg.h 文件，为类 CStenotypistDlg 增加相应的变量和函数；

6）用 ClassWizard 为 CStenotypistDlg 类添加对控件和消息的响应函数。StenotypistDlg.h 的代码如下。

```
#include "SpeechRecognition.h"

/////////////////////////////////////////////////////////////////////////
// CStenotypistDlg dialog

class CStenotypistDlg : public CDialog
{
// Construction
public:
    CStenotypistDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CStenotypistDlg)
    enum { IDD = IDD_STENOTYPIST_DIALOG };
    CButton    m_btDictation;
    CString    m_strText;
```

```
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CStenotypistDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

    CSpeechRecognition   m_SpeechRecognition;

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CStenotypistDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnButtonVt();
    afx_msg void OnButtonMs();
    afx_msg void OnButtonDictate();
    //}}AFX_MSG
    afx_msg LRESULT OnSREvent(WPARAM, LPARAM);
    DECLARE_MESSAGE_MAP()
};
```

注意，在 CStenotypistDlg 类中定义了一个 CSpeechRecognition 类的对象。

在 OnInitDialog 函数中调用 CSpeechRecognition 的初始化函数和设置语音语言列表：

```
BOOL CStenotypistDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
```

```
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);         // Set big icon
    SetIcon(m_hIcon, FALSE);    // Set small icon

    // TODO: Add extra initialization here
    if (! m_SpeechRecognition.Initialize(m_hWnd))
            AfxMessageBox(m_SpeechRecognition.GetErrorString());

    m_SpeechRecognition.InitTokenList(GetDlgItem(IDC_LIST1)->m_hWnd);

    m_SpeechRecognition.Stop();

    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

开始听写和停止听写的实现较简单，只需调用 CSpeechRecognition 类的响应函数就能实现，其代码如下所示。注意，停止和开始是互相切换的。

```
void CStenotypistDlg::OnButtonDictate()
{
    if (m_SpeechRecognition.IsDictationOn())
    {
        m_SpeechRecognition.Stop();
        m_btDictation.SetWindowText("听写(&D)");

        SetWindowText("听写者 - 请按<听写>按钮开始听写!");
    }
    else
    {
        m_SpeechRecognition.Start();
        m_btDictation.SetWindowText("停止(&S)");

        SetWindowText("听写者 - 正在记录，请口述...");
    }
}
```

设置话筒和语音训练也通过直接调用 CSpeechRecognition 类的成员函数来实现：

```
void CStenotypistDlg::OnButtonVt()
{
    m_SpeechRecognition.VoiceTraining(m_hWnd);
}

void CStenotypistDlg::OnButtonMs()
{
```

```
    m_SpeechRecognition.MicrophoneSetup(m_hWnd);
}
```

为了响应消息 WM_SREVENT，需要添加相应的消息响应函数：

```
BEGIN_MESSAGE_MAP(CStenotypistDlg, CDialog)
    //{{AFX_MSG_MAP(CStenotypistDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_BUTTON_VT, OnButtonVt)
    ON_BN_CLICKED(IDC_BUTTON_MS, OnButtonMs)
    ON_BN_CLICKED(IDC_BUTTON_DICTATE, OnButtonDictate)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_SREVENT, OnSREvent)
END_MESSAGE_MAP()

LRESULT CStenotypistDlg::OnSREvent(WPARAM, LPARAM)
{
    WCHAR *pwzText;
    m_SpeechRecognition.GetText(&pwzText);

    m_strText += CString(pwzText);
    UpdateData(FALSE);

    return 0L;
}
```

7）为了调用 Speech 引擎，应该在 Microsoft Visual C++编程环境中设置好相应的 include 和 lib 设置：

① 设置 include 路径

- 通过 Project→Settings 菜单项打开 Project Settings 对话框；
- 点击 C/C++项；
- 在 Category 下拉列表中选取 Preprocessor；
- 在“Additional include directories”编辑框中输入安装 Speech SDK 的 include 的路径，默认的路径是 C:\Program Files\Microsoft Speech SDK 5.1\Include。

② 设置 lib 信息

- 通过 Project→Settings 菜单项打开 Project Settings 对话框；
- 选择 Link 项；
- 在 Category 下拉列表中选取 Input 项；
- 在“Additional library path”编辑框中输入安装 Speech SDK 的 lib 的路径，默认的路径是 C:\Program Files\Microsoft Speech SDK 5.1\ Lib\i386；
- 将“sapi.lib”输入“Object/library modules”所标识的编辑框中。

8）编译连接该项目，就可让听写者开始听写了。