关于COM, COM+, and DCOM的简单描述

A Simple Description of COM, COM+ and DCOM

William Xu
7/10/2022

摘要 ：
COM (Component Object Model) 是Microsoft 90年代开发的通讯工具. 后期又增加了一些功能, 称之为 COM+. 之后又出现了DCOM. DCOM (Distributed Component Object Model), 也就是分布式 COM, 它允许计算机通过网络在另一台计算机上运行程序。DCOM 最早出现于 Microsoft 1993年发行的 Windows NT 系统. 该系统支持多台PC联网.　本文描述COM最简单的原理和使用方法.最后给出一个实例 (COM Server + COM Client)以供读者参考.

关于COM, COM+, and DCOM的简单描述

William Xu July 4, 2022

目 录

I. 什么是COM, COM+, and DCOM

1.1 COM

COM（Component Object Model）是微软主要为 Windows 开发的组件软件。 它是 OLE 和 ActiveX 的基础，并提供了一种无需重新编译即可重用代码的方法。 在 COM 中，组件是兼容应用程序和其他组件可以使用的特定于平台的二进制文件。 包含组件服务的程序永远无法访问其内部数据结构，而是包含指向其标准化接口的指针。 因此，无论组件如何工作或使用何种语言编写，组件都可以相互交互。

1.2 COM+

COM+ 是 COM 的增强版本，可提供更好的安全性和更多的性能。它是 Microsoft COM 和 MTS (Microsoft Transaction Server)的演变。 COM+ 扩展了 COM 技术编写程序的能力，它可以处理许多以前必须程序员自己编写的资源管理任务,例如 thread 分配和安全。COM+ 还通过提供 thread pooling, object pooling, and just-in-time object activation, 等功能使程序更具可扩展性。COM+ 还通过提供 transaction support 来帮助保护数据的完整性，可以使数据跨越网络上的多个数据库。

1.3 DCOM

DCOM 实际上有80%就是COM. 它增加的20%仅仅是网络安全部分. DCOM（分布式组件对象模型）是 COM 的扩展，它允许应用程序和组件通过网络相互通信

作为 COM 的扩展，DCOM 解决了 COM 模型的一些固有问题，以便更好地在网络上使用. DCOM 主要有三项改动,分别叙述如下.

1.3.1 Marshalling (编组)

编组解决了将数据从一个 COM 对象实例传递到另一台计算机上的另一个的需要——在编程术语中，这称为“传递参数”。例如，如果我想要 Zaphod 的姓氏，我会调用带有 Zaphod 参数的 COM 对象 LastName。 LastName 函数将使用远程过程调用 (RPC) 向目标服务器上的其他 COM 对象询问 LastName(Zaphod) 的返回值，然后它将答案 - Beeblebrox - 发送回第一个 COM 对象。

1.3.2 Distributed Garbage Collection (分布式垃圾收集)

Distributed Garbage Collection 旨在扩展 DCOM 以支持大容量互联网流量，分布式垃圾收集还解决了一种销毁和回收已完成或放弃的 DCOM 对象的方法，以避免炸毁网络服务器上的内存。反过来，它与交易链中的其他服务器通信，让他们知道他们可以摆脱与交易相关的对象。

1.3.3 Using DCE/RPC as the underlying RPC mechanism

使用 DCE/RPC 作为底层 RPC 机制. 为了实现前面的项目并尝试扩展以支持大容量 Web 流量，Microsoft 将 DCE/RPC 实现为 DCOM 的底层技术——这就是 DCOM 中的 D 的来源。

II.COM 的存在形式以及操作方法

2.1 COM 以 Client/Server 这形式存在于电脑中.

COM 以 Client / Server 对这形式存在于电脑中.COM 的关键是 Clint 和 Server 如何交互。基本过程是:

A) COM Server 为客户端提供需要的服务.

B) COM Client 获得指向 Server 的指针并通过调用其 Interface 的方法来取得 Server 的服务。

2.2 COM Server 可以有两种, DLL 和 EXE

2.2.1 两种 COM Server, In-process DLL 和 Out-of-process EXE

主要有两种类型的服务器，In-process 和 Out-of-process。In-process 在 DLL(Dynamic Link Library) 中实现; Out-of-process 服务器在 EXE 中实现。注意 Out-of-process 服务器可以驻留在本地计算机或远程计算机上, 这样就可以运行跨机执行程序. 此外 COM 提供一种机制允许 DLL 在代理 EXE 进程中运行，以获得能够在远程计算机上运行该进程的优势。

如果是在 Windows 系统下面, DLL 还可能以两种形式存在: 普通的 DLL 形式和 Windows Service 形式. Windows Service 形式在编程的时候要用特别的格式.另外它的 de-bugger也比较困难.

2.2.2 如何注册 COM Server

对于 EXE Server:

ServerName.exe /regserver <CR>　　(注册)

ServerName.exe /unregserver <CR>　　(退出注册)

对于 DLL Server:

Regsvr32 servername.dll<CR>　　(注册)

Regsvr32 /u servername.dll <CR>　　(退出注册)

对于 DLL Windows COM Service:

可以用 Component Services 程序来启动和停止. 怎样启动?

Control Panel -> Administrative Tools -> Component Services

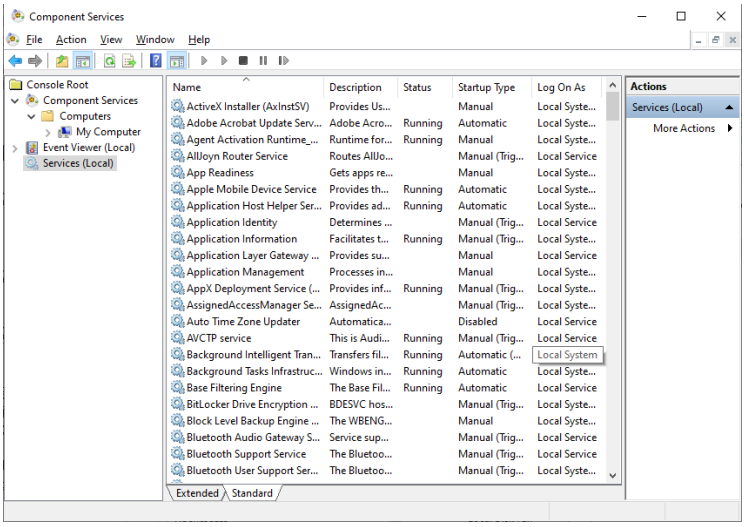


图1. Windows 的 Component Services 控制程序

2.3 建造 COM 的几种方式

2.3.1 C++ 方式

用 VC++ 可以直接手工编制 COM 程序. 对初学者来说这并不容易, 这里有一些基本的规则要学会.

建立COM 接口需要建立一抽象接口, Interface Class, IUnknown. 这是所有 COM 程序必须有的基本 Interface. 该 class 不仅实现 IUnknown 接口, 还为该接口提供各种 Method。COM Client 靠着调用不同的 Method 来实现它的功能. 例如在下面的实例中有一个 method 叫 SpellingChecker(),它用来检查拼法.起初这 IUnknown 可能看起来很复杂, 但它是 COM 技术的基础。

```
class IUnknown
{
public:
   virtual HRESULT QueryInterface(REFID riid, void** ppv)=0;
   virtual ULONG AddRef () = 0;
   virtual ULONG Release() = 0;
};
```

2.3.2 ATL COM方式

ATL (Active Template Library) 是 Microsoft 专为 COM 设计的. 它是一组基于C++的模板，可快速创建小型 COM。 它支持关键 COM 功能，包括库存实现、双接口、标准 COM 枚举器接口、连接点、分离接口和 ActiveX 控件。很多作者都认为用 ATL 它来建立 COM 是最简单和快捷的方式,

本文后面介绍的 COM Client / Server 例子就是 ATL 建 COM 的实例. 下面可以看到因为 ATL COM Wizard 可以帮你自动生成大部分的 code, 所以编程就简单得多.

III. 用 ATL COM 方式建立 Client / Server

3.1 用一个实例来说明如何用 ATL方式建立 COM Client / Server.

3.1.1 这个实例具有以下功能

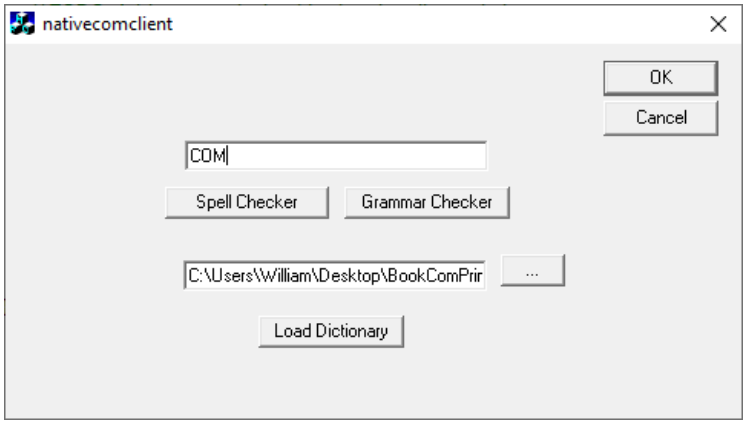


图2. COM Client 程序用于检查拼法和语法

1) 一个简单的 Spelling Checker + 一个简单的 Grammar Checker.
2) 为 Spelling Checker 建立一个五个字的内部字典.(COM, COM+, ATL, ActiveX,OLT等). 如果输入的字不在这五个字内,那就算拼错.
3) Grammar Checker 就更简单. 如果输入的字串没有逗号”,”和句号”.”就算出错.
4) 除了内部的五字字典, 该程序还可以装载用户自己定义的字典. 这样这个程序就具有可扩充性. 自定义的字典是一个 *.TXT 文件, 每一行写一个字就可以了. 比如下面的 test.txt 文件就增加了三个词汇: BOOK ,STREET, 和 FAMILY.

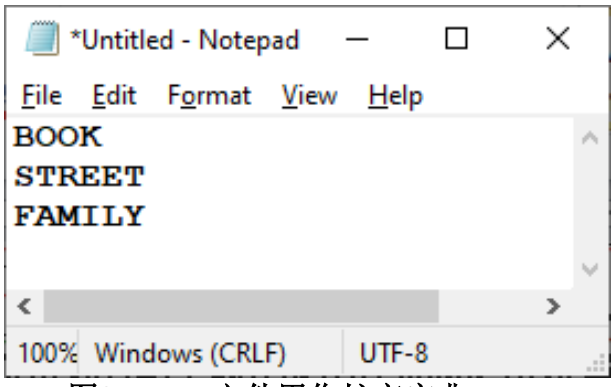


图3. TXT 文件用作扩充字典

3.1.2 这个实例的软件结构

实例的软件结构可见图10. 以下是文字描述:

1) Project 名: WritersComponent
2) COM Class 名: CDocumentChecker
3) COM Interface 1 : ISpellChecker (ATL自动产生)
   它带有2个 Method: CheckSpelling(),
   UseCustomDictionary().
4) COM Interface 2 : IGrammarChecker (程序员手工编写)
   它带有1个 Method: CheckGrammar()

按照本书的描述, 用 ATL 的方式产生 COM 时只能自动产生一个 COM Interface. 如果你要产生第2个 COM Interface, 必须手工编写. 所以在这个例子中作者也教你如何手工编写第2个 COM Interface (IGrammarChecker). 但是Interface所带的 Method 可自动产生不受限制, 即是说ATL 可以帮你产生多个 Method.

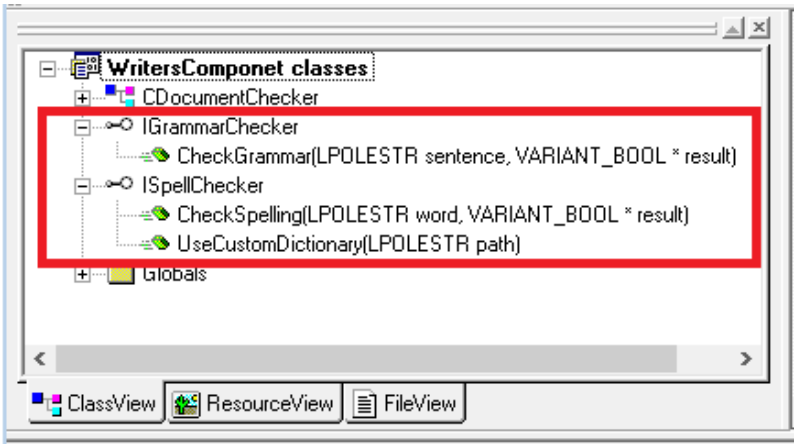


图4. 实例的软件结构

3.2 用 ATL方式建立 COM Server

3.2.1 建立ATL Server 概述.

建立ATL Server 概念上有以下几步:

1) 用ATL/COM AppWizard 建立ATL Project
2) 用ATL object Wizard 增加一个名为 DocumentChecker 的 object.
3) 建立 interface “ISPellChecker”.
4) 建立 interface “IGarmmarChecker”.

3.2.2 用ATL/COM AppWizard 建立ATL Project

3.2.2.1 启动 VC++ 6.0 建立ATL Project

参见图4.

. 启动 VC++ 6.0,
. New -> Project.
. 用ATL/COM AppWizard 建立ATL Project, 名为“WritersComponent”.
. 然后按 OK.

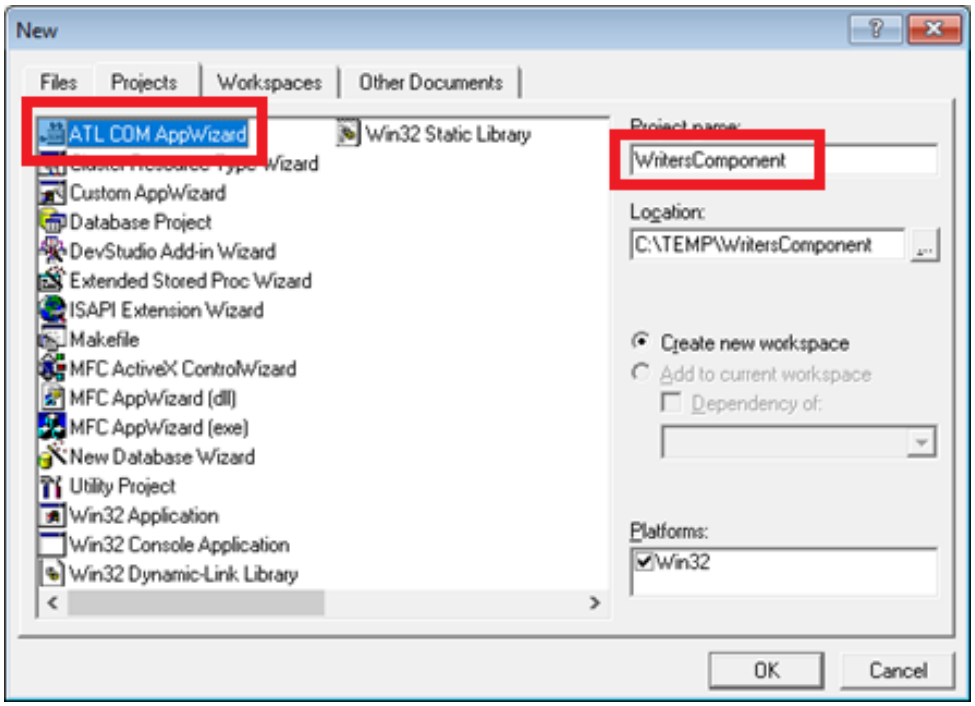


图5. 选择 ATL COM AppWizard 并且命名为 “WritersComponent”

参见图5.

. 选择 “Dynamic Link Library (DLL),

. 然后按 Finish.

3.2.2.2 建立 New ATL Object

参见图6.

可以看到这时候已经自动产生好多文件. 选择Inset->New ATL Object

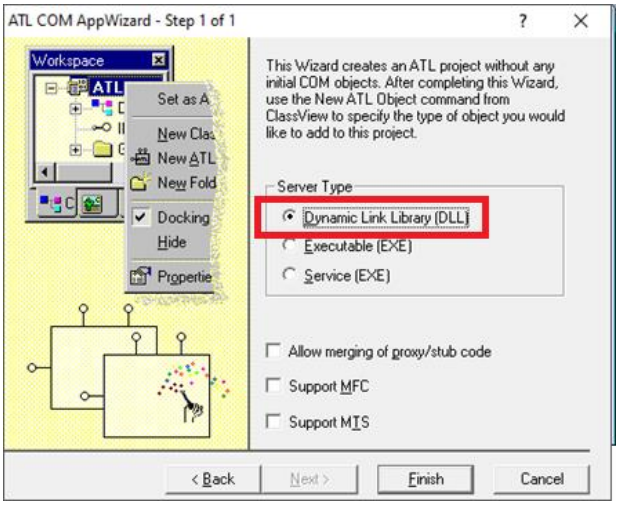


图6. ATL COM AppWizard

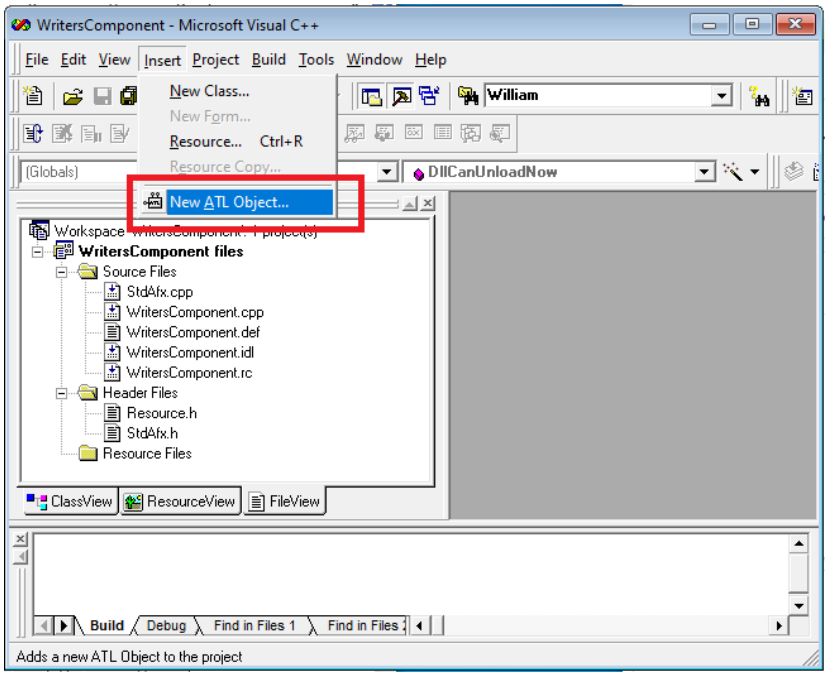


图7. Inset->New ATL Object

参见图8.
选择 Simple Object -> Next. 这时看到图9.

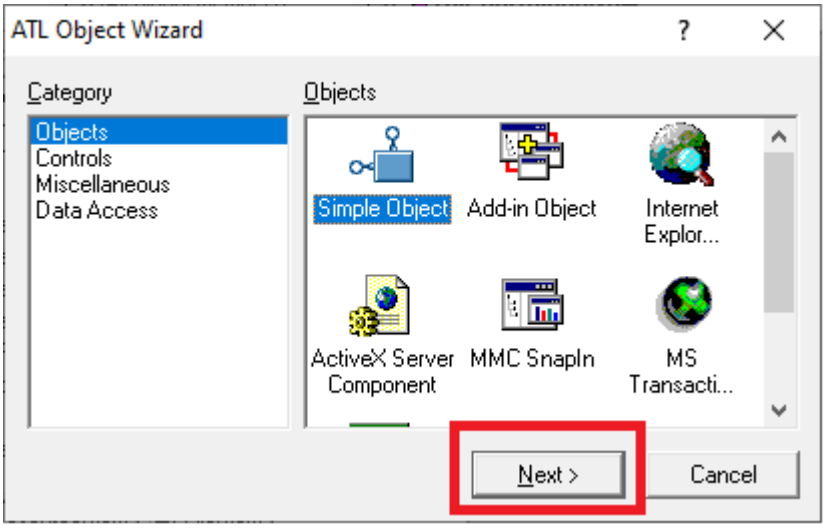


图8.ATL Object Wizard

3.2.2.3 **建立** New Interface

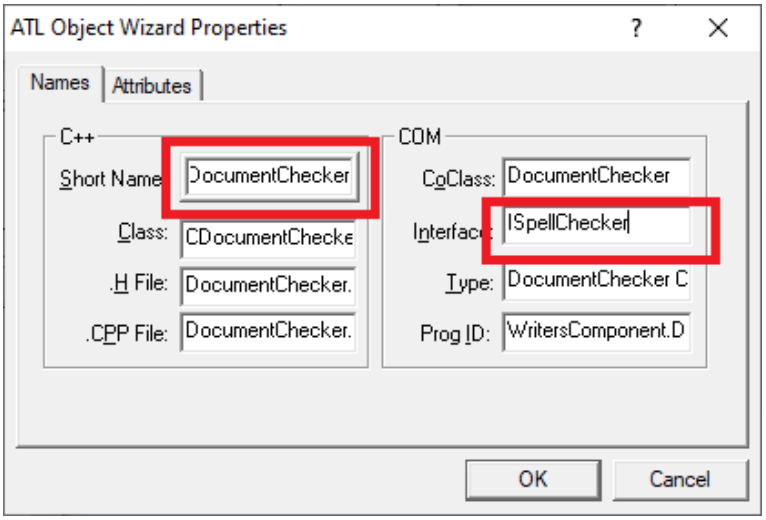


图9. Names Page 的选择

在 Names Page, 必须有如图8的两个输入,其他的都会自动产生.

在 Attributes Page, 必须有如图9的选择.

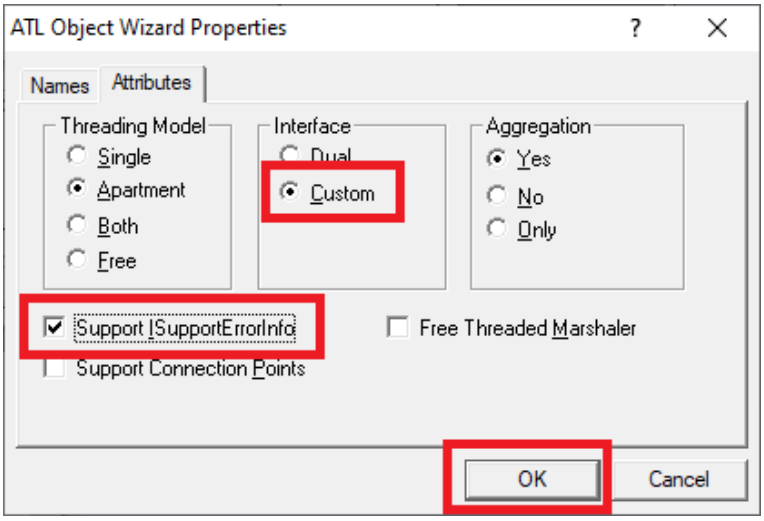


图10. Attributes Page 的选择

这个时候就看到一个叫做 ISpellChecker 的 Interface (源于IUnknown)(见图11)已经产生.

3.2.2.4 在Interface 下增加 Method

以下,(参见图12) 我们要在 Interface “ISpellChecker” 下面加两个 Method, 即 CheckSpelling 和 CheckGrammar.

Right click on “ISpellChecker” -> “Add Method”

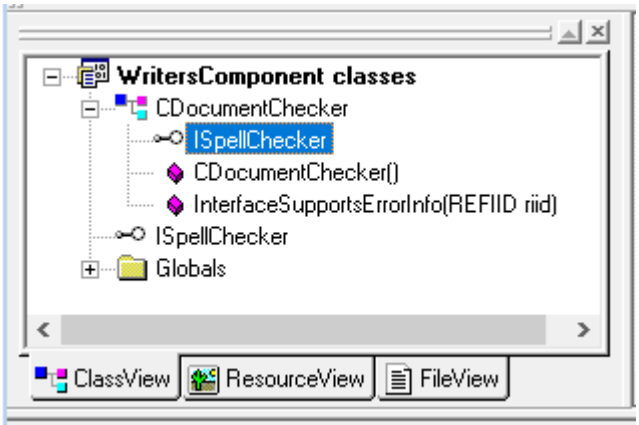


图11. Interface “ISpellChecker” 已产生

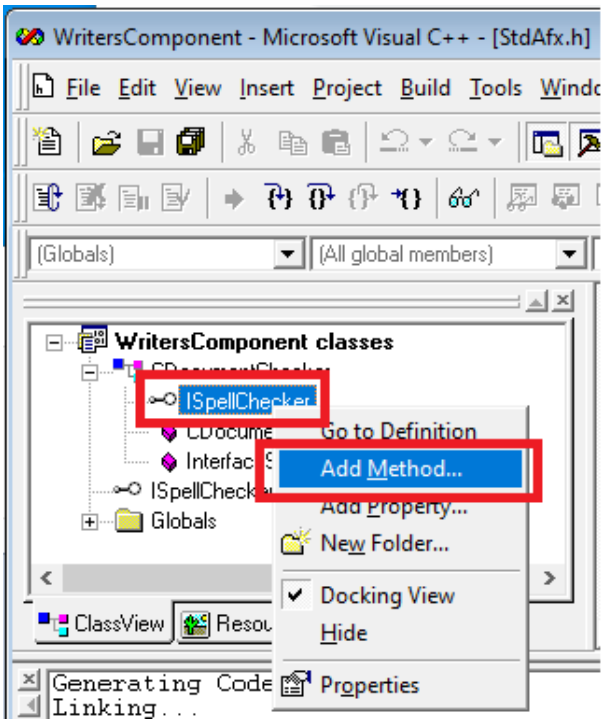


图12. ISpellChecker -> Add Method

这时你看到图13. 在图13中必须输入以下各项, 然后按 OK.

Method Name: CheckSpelling

Parameters: [in] LPOLESTR word, [out, retval] VARIANT_BOOL *result

图12上的 Parametrs 表明这个Method

输入是: LPOLESTR word

输出是: retval, 是一个 VARIANT_BOOL *result

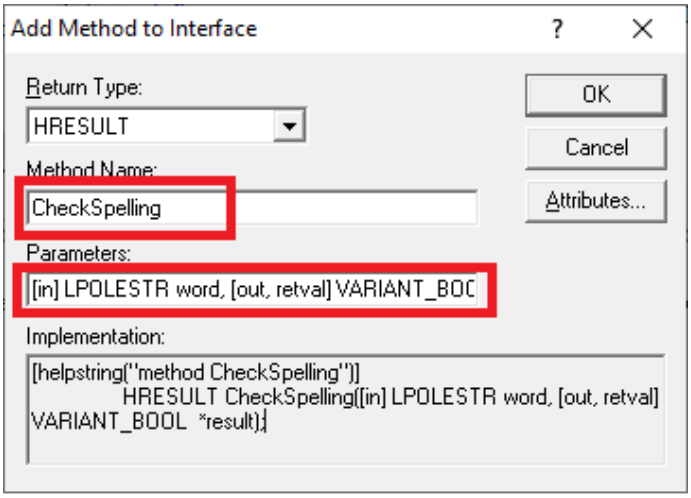


图13. Add Method to Interface

按图13 “OK”,就看到图14. 这时 Method “CheckSpelling” 已经产生. 在 DocumentChecker.cpp 这个文件中已自动加入了一个空的 Method “CheckSpelling”.

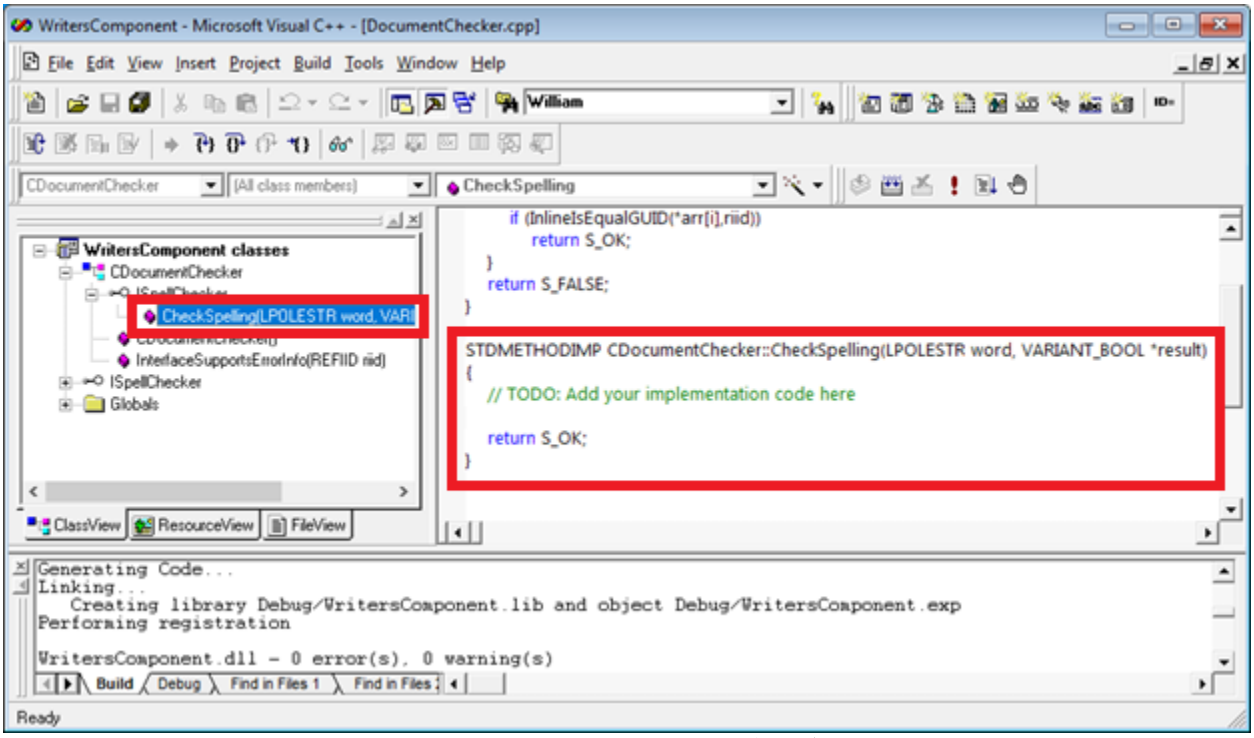


图14. Method “CheckSpelling” 已自动加入

3.2.2.5 在 Method 中加入执行 Code

下面是我们自行加入的 CheckSpelling()执行code. 把这些 code 填到上述空的 Method CheckSpelling 即可.

在文件 DocumentChecker.cpp 加入以下各项:

```
STDMETHODIMP CDocumentChecker::CheckSpelling(LPOLESTR word, VARIANT_BOOL *result)
{
	// TODO: Add your implementation code here
	wstring str;
	vector<wstring >::iterator iter;
	*result = VARIANT_FALSE;
	for ( iter = m_dictionary.begin(); iter != m_dictionary.end(); iter++) {
		str = *iter;
		if (_wcsicmp(str.c_str(), word) == 0) {
			*result = VARIANT_TRUE;
			break;
		}
	}
	return S_OK;
}
```

在文件 DocumentChecker.h 加入以下各项:

```
////////////////////////////////////////////////////////////////////////////
// CDocumentChecker

// William
#include <vector>
#include <string>
using namespace std;

class ATL_NO_VTABLE CDocumentChecker :
	public CComObjectRootEx<CComSingleThreadModel>,
	public CComCoClass<CDocumentChecker, &CLSID_DocumentChecker>,
	public ISupportErrorInfo,
	public ISpellChecker,
	public IGrammarChecker
{
public:
	CDocumentChecker()
	{
		// William
		m_dictionary.push_back(OLESTR("Alan"));
		m_dictionary.push_back(OLESTR("Gordon"));
		m_dictionary.push_back(OLESTR("COM"));
		m_dictionary.push_back(OLESTR("ActiveX"));
		m_dictionary.push_back(OLESTR("OLE"));
		m_dictionary.push_back(OLESTR("COM+"));
	}

... ...
// William
private:
	vector <wstring> m_dictionary;
... ...
// IGrammarChecker
};
```

以下我们要建立第2个 Method, 称为 UseCostomDictionary. 按照图11的方式让系统自行产生这个空的 Method. 它的输入参数只有一个, 就是 LPOLESTR path. 参见图15.

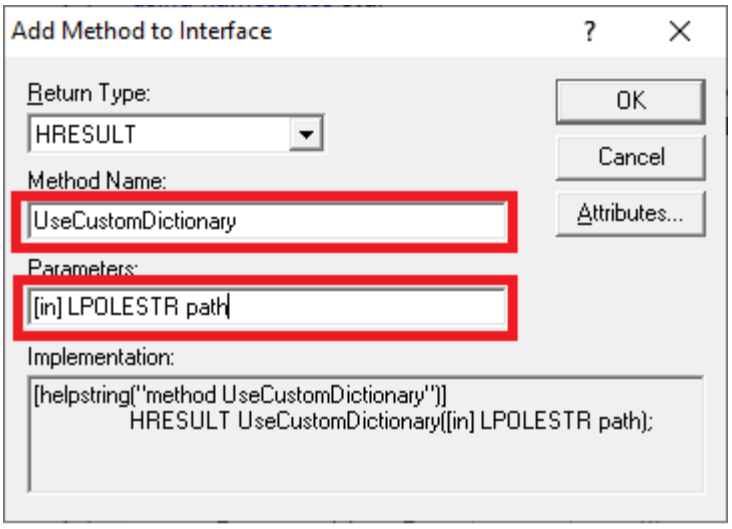


图15. 建立下一个Method “UseCostomDictionary”

空的 Method “UseCostomDictionary” 自动建成. 把下列 code 加到 DocumentChecker.cpp. 请注意在开始的几行程序中定义了一个出错信息, DICTIONARYFILENOTEFOUND. 这个只是在调用 UseCustomDictionary() 出错时使用.

下列 code 加到 DocumentChecker.cpp.

```
////////////////////////////////////////////////////////////////////////////
// CDocumentChecker

// William
// Define an Error message DICTIONARYFILENOTEFOUND for method UseCustomDictionary
#include <fstream>
const E_DICTIONARYFILENOTEFOUND = MAKE_HRESULT (SEVERITY_ERROR, FACILITY_ITF, 0x200+99);

… …
STDMETHODIMP CDocumentChecker::UseCustomDictionary(LPOLESTR path)
{
	// TODO: Add your implementation code here

	// William
	wstring str;
	char sz_path[255];
	wifstream dictionaryFile;
	size_t bufsize;
	bufsize = wcstombs (sz_path, path , 255);
	if ( bufsize==sizeof(sz_path) || bufsize == -1) {
	    return Error(_T("Invalid or excessively long pah specifined"),
	                 IID_ISpellChecker, E_INVALIDARG);
	}

	dictionaryFile.open(sz_path);
```

```
	if (!dictionaryFile.is_open()) {
//		return Error(_T("Dictionary File not found"),  IID_ISpellChecker,
E_DICTIONARYFILENOTFOUND);
		return Error(_T("Dictionary File not found"),  IID_ISpellChecker,
E_INVALIDARG);
	}

	while ( !dictionaryFile.eof() ){
		dictionaryFile >> str;
		m_dictionary.push_back(str);
	}

	return S_OK;
}
```

3.2.3 手工编写第2个 COM Interface

用 ATL 的方式产生 COM 时只能自动产生一个 COM Interface. 如果你要产生第2个 COM Interface, 必须手工编写. **以下描述**如何手工编写第2个 COM Interface IGrammarChecker.

3.2.3.1 修改 COM DLL Source 文件 IDL

下列蓝色 code 加到 WritersComponet.idl. 请注意这个 uuid 是我们自创的:
uuid(20FD9098-A292-423E-B37C-343BF3115BCA)
只要取与上一个Intereface “CheckSpelling”不同的 uuid 值就可以.

```
// WritersComponet.idl : IDL source for WritersComponet.dll
//

// This file will be processed by the MIDL tool to
// produce the type library (WritersComponet.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
	[
		object,
		uuid(19FD9098-A292-423E-B37C-343BF3115BCA),

		helpstring("ISpellChecker Interface"),
		pointer_default(unique)
	]
	interface ISpellChecker : IUnknown
	{
```

```
		[helpstring("method CheckSpelling")] HRESULT CheckSpelling([in] LPOLEST word,
[out, retval] VARIANT_BOOL  *result);
		[helpstring("method UseCustomDictionary")] HRESULT UseCustomDictionary([in]
LPOLESTR path);
	};

	// William
	[
		object,
		uuid(20FD9098-A292-423E-B37C-343BF3115BCA),

		helpstring("IGrammaChecker Interface"),
		pointer_default(unique)
	]
	interface IGrammarChecker : IUnknown
	{
		[helpstring("method CheckGrammar")] HRESULT CheckGrammar([in] LPOLESTR
sentence, [out, retval] VARIANT_BOOL  *result);
	};

[
	uuid(40D9BB73-57A3-4573-956F-2C93AD7CABA1),
	version(1.0),
	helpstring("WritersComponet 1.0 Type Library")
]
library WRITERSCOMPONETLib
{
	importlib("stdole32.tlb");
	importlib("stdole2.tlb");

	[
		uuid(F42090D2-9C68-4E00-AC09-2FF5E57FB4AE),
		helpstring("DocumentChecker Class")
	]
	coclass DocumentChecker
	{
		[default] interface ISpellChecker;
		// William
		interface IGrammarChecker;
	};
};
```

3.2.3.2 编译这个新的 Source 文件 IDL

修改以后的 IDL 文件须重新进行编译. 打开WritersComponet.idl, 按 F7即可. 见图16.

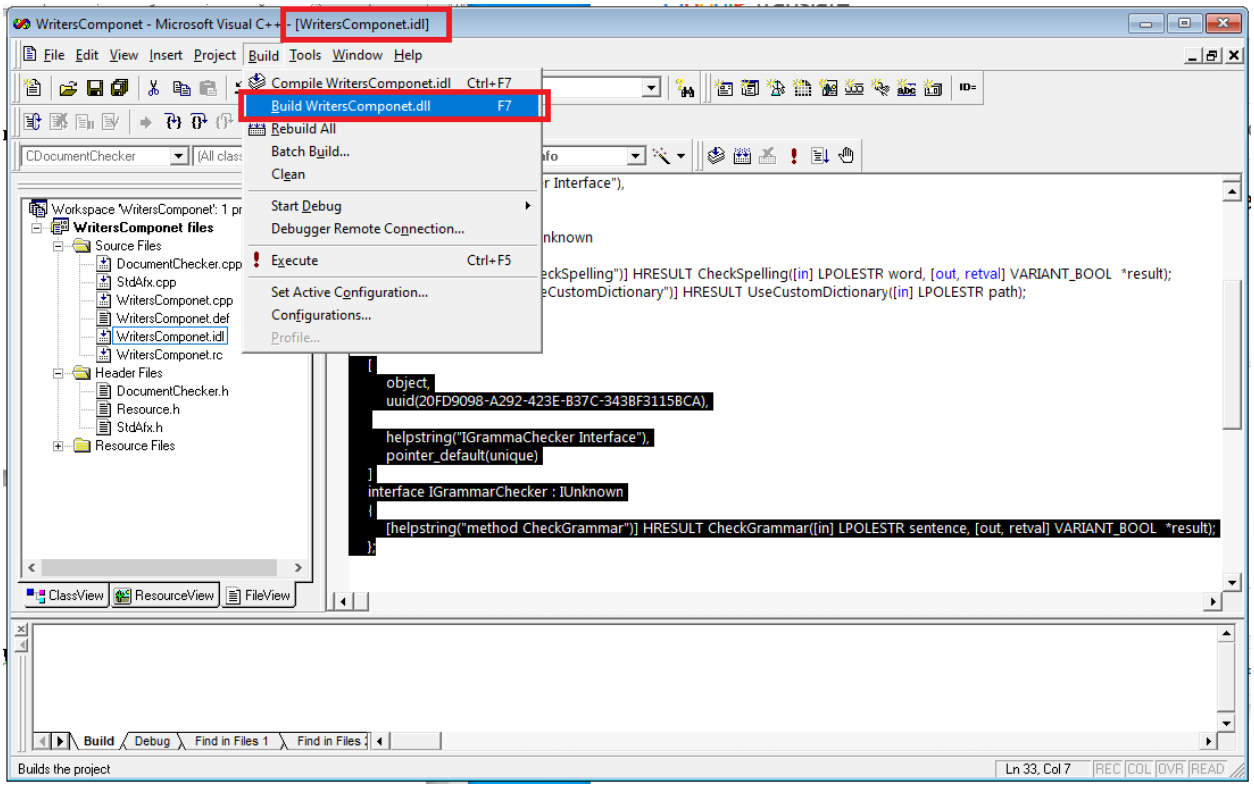


图16. 重新编译 WritersComponet.idl

3.2.3.3 Run Implement Interface 把新增的 IGrammarChecker 加到 COM Server

做以下几步:

A) 参考图16 , Right Click on CDocumentChecker -> Implement Interface.
这时就看到图17, Implement Interface Window.

B) 选择 IGrammarChecker, OK.
这时就看到第2个interface IGrammarChecker 已经加上. 图18.

图17. Implement Interface 把新增的 IGrammarChecker 加到 COM Server

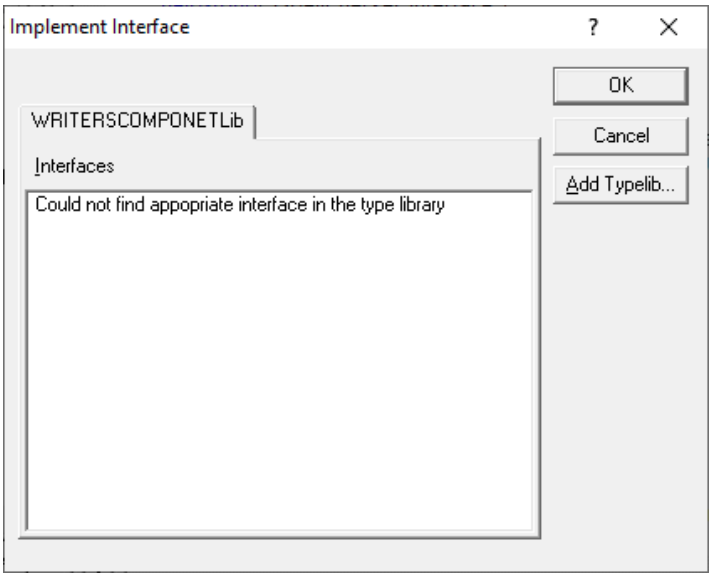


图17. Implement Interface

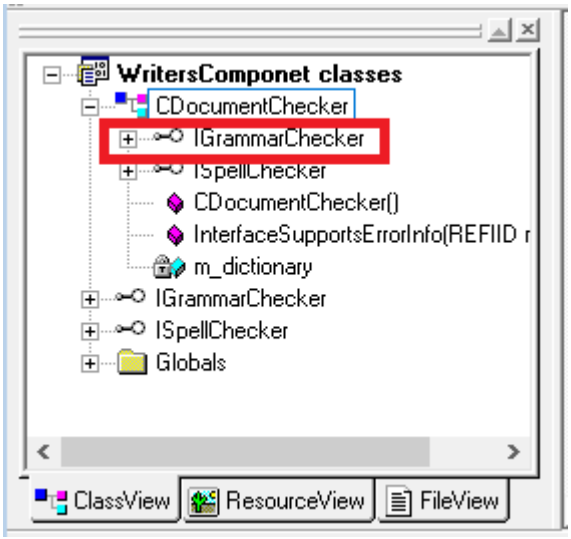


图18. 第2个interface IGrammarChecker 已加上

3.2.3.4 修改 DocumentChecker.cpp 增加出错处理信息

在文件 DocumentChecker.cpp 中加上一下code

```
STDMETHODIMP CDocumentChecker::InterfaceSupportsErrorInfo(REFIID riid)
{
	static const IID* arr[] =
	{
		&IID_ISpellChecker,
		// William
```

```
            &IID_IGrammarChecker
        };
        for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
        {
            if (InlineIsEqualGUID(*arr[i],riid))
                return S_OK;
        }
        return S_FALSE;
}
```

3.2.3.5 在新增的 Interface IGrammarChecker下加 Method CheckGrammar()

按照图11的办法在新增的 IGrammarChecker下加 Method CheckGrammar().它的输入输出参数是:

输入是: LPOLESTR sentence

输出是: retval, 是一个 VARIANT_BOOL *result

3.2.3.6 在新增的 Method CheckGrammar()加执行Code

把下列 Code 增加到 DocumentChecker.cpp 文件中

```
STDMETHODIMP CDocumentChecker::CheckGrammar(LPOLESTR sentence, VARIANT_BOOL *result)
{
        // TODO: Add your implementation code here

        // William
        *result = VARIANT_FALSE;
        if (wcschr(sentence,',')  && wcschr(sentence,'.'))
            *result = VARIANT_TRUE;

        return S_OK;
}
```

3.2.4 编译和注册 COM Server DLL

到此为止 COM Server 编程已经完工. 下面是最后一步编译和注册. 共有三步:

A) Set Active Project Configuration -> Win32 Debug

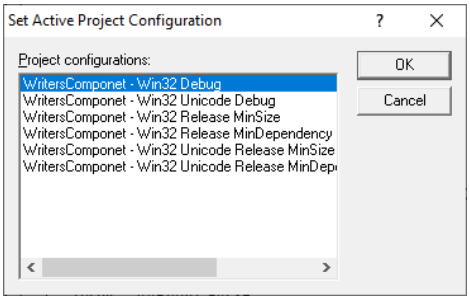


图19. Set Active Project Configuration

B) Run F7 编译 COM Server DLL

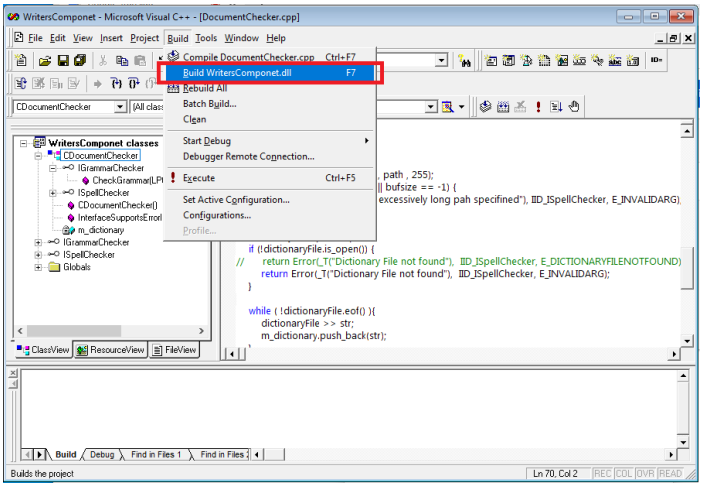


图20. F7 编译 COM Server DLL

C) 注册 COM Server DLL

Tools -> Registration Control

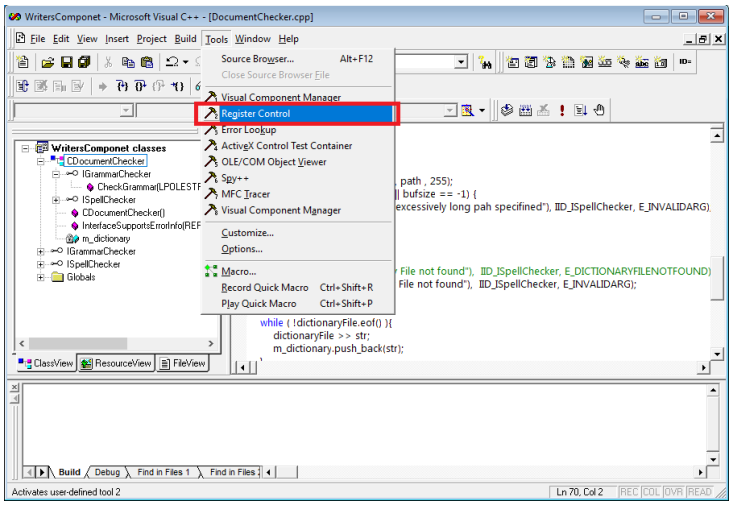


图21. Registration Control

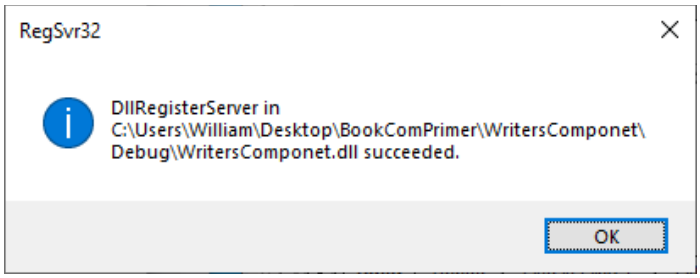


图22. 注册 COM Server DLL完成

3.3 用 ATL方式建立 COM Client

3.3.1 建立ATL Client 概述

概念上有以下几步:

1) 在 COM Server 里定义的好多内容(如 UUID等) 都在一个文件 Server名.tlb 中表述. 因此在建立 Client 的时候必须把这个 Server 文件输入进来.

```
#import "WritersComponet.tlb"
```

2) 按照 ATL 的规定, 如要调用 ATL COM, 必须 #include 以下文件

```
#include <AfxOle.h>
AfxOleInit();
```

3) 程序开始必须建立 COM 的 Instance

```
HRESULT hr = mSpellChecker.CreateInstance \
            ("WritersComponet.DocumentChecker");
```

4) 建立一个 COM 的 Smart Pointer, 以后COM各项功能的调用都通过这个指针来实现.

```
private:
      ISpellCheckerPtr mSpellChecker;
```

5) 到此,所有准备工作都已经完毕. 以下就可以随意建立用户的 Console App or MFC App Client 程序.

3.3.2 建立一个 Dialog Based V++ 6.0 应用程序, Nativecomclient

3.2.2.1 定义应用程序 Nativecomclient

见图22, 建立的应用程序有两个 Edit Field, 分别被定义为:
(见 nativecomclientDlg.h)

```
CString     m_word;
CString     m_dictionaryFile;
```

该程序又有4个 Click Buttons, “Spell Checker”, “GammarChecker”, “…” 和 “Load Dictionary”. 他们分别有下列四个callback function:
(见 nativecomclientDlg.cpp)

```
OnSpellcheckButton();
OnGrammercheckerButton();
OnBrowseButton();
OnLoadDictionary()
```

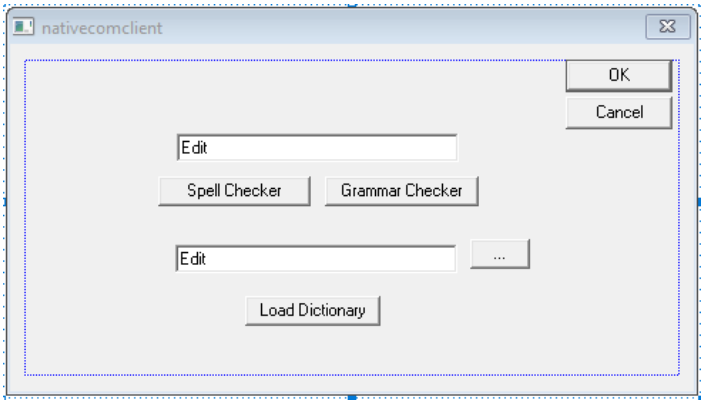


图22. V++ 6.0 应用程序, Nativecomclient

3.2.2.2 输入 COM Server Class **原始数据文件** TLB

COM Server Class 原始数据都保存在一个 TLB 文件中. 它是在编译 COM Debug version 时自动产生的. 可以在 ../../Debug 这个目录下找到. 在我们的例子下,这个文件就是 “WritersComponet.tlb”.

把这个文件 WritersComponet.tlb 拷贝到 Project 找得到的地方, 然后在 StdAfx.h文件中加入一行字:

```
// StdAfx.h
… …
// William
#import "WritersComponet.tlb"

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif //     !defined(AFX_STDAFX_H__BAEDD675_E47C_44A9_B337_EC4C4B289D78__INCLUDED_)
```

3.2.2.3 修改 Nativecomclient 各个文件

3.2.2.3.1 修改 NativecomclientApp.cpp, Enable Support COM

```
BOOL CNativecomclientApp::InitInstance()
{
	// William
	if (!AfxOleInit())
		AfxMessageBox("Could not initalize COM");

	AfxEnableControlContainer();
	… …
}
```

3.2.2.3.2 修改 NativecomclientDlg.h, Adding a Smart Pointer

```
/////////////////////////////////////////////////////////////////////////////
// CNativecomclientDlg dialog

// William
using namespace WRITERSCOMPONETLib;
class CNativecomclientDlg : public CDialog
{
// Construction
// William
private:
	ISpellCheckerPtr mSpellChecker;
… …
}
```

3.2.2.3.3 修改 NativecomclientDlg.cpp, OnInitDaialog(),建立 Smart Pointer Instance

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

	… …

	// 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

	// William
	HRESULT hr = mSpellChecker.CreateInstance ("WritersComponet.DocumentChecker");
	if (FAILED(hr)) {
		_com_error err (hr);
		AfxMessageBox("Error: mSpellChecker.CreateInstance");
			AfxMessageBox(err.ErrorMessage());
	}
	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

这里是建立一个 COM Smart Pointer 的 Instance. 在这里, ("WritersComponet.DocumentChecker") 这个字符串很长,容易写错. 如果写错你就会看到一个出错信号:

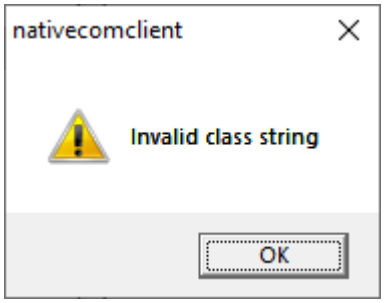


图23. 出错信号

参考 “Appendix A 查验正确的 COM 字符串” 可明白写错在哪里.

3.2.2.3.4 修改 NativecomclientDlg.cpp, 加入4四个按钮的功能 Code

```
void CNativecomclientDlg::OnSpellcheckButton()
{
	// TODO: Add your control notification handler code here

	// William
	UpdateData(TRUE);
	bstr_t word = m_word;
	VARIANT_BOOL isCorrect = mSpellChecker->CheckSpelling(word);
	if (isCorrect == VARIANT_TRUE) {
		AfxMessageBox("Pass: Spell check complete");
	}
	else {
		AfxMessageBox("Error: Word misspelled");
	}
}
```

```
void CNativecomclientDlg::OnGrammercheckerButton()
{
	// TODO: Add your control notification handler code here

	// William
	VARIANT_BOOL isCorrect;
	UpdateData(TRUE);
	bstr_t word = m_word;


	IGrammarCheckerPtr    grammarChecker;
	try {
		grammarChecker = mSpellChecker;
		isCorrect = grammarChecker->CheckGrammar(word);
		if ( isCorrect)
			AfxMessageBox("Pass: Grammer check complete");
		else
			AfxMessageBox("Error: Incorrect Grammar");
	}

	catch (_com_error err) {
		AfxMessageBox(err.Description());
	}
}

void CNativecomclientDlg::OnBrowseButton()
{
	// TODO: Add your control notification handler code here

	// William
	CFileDialog dlg(TRUE);
	if (dlg.DoModal() == IDOK) {
		m_dictionaryFile = dlg.GetPathName();
		UpdateData(FALSE);
	}
}

void CNativecomclientDlg::OnLoadDictionary()
{
	// TODO: Add your control notification handler code here

	// William
	UpdateData(TRUE);
	bstr_t dictionaryFile = m_dictionaryFile;
	try {
		mSpellChecker->UseCustomDictionary(dictionaryFile);
	}
	catch (_com_error err) {
```

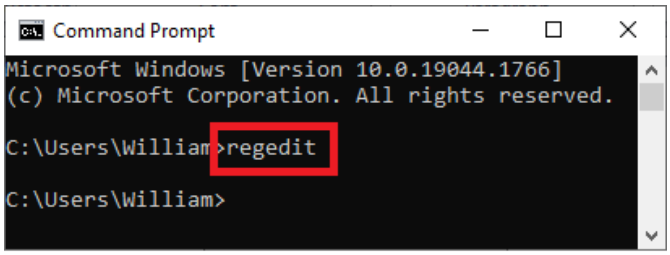
```
            AfxMessageBox(err.ErrorMessage());
        }
}
```

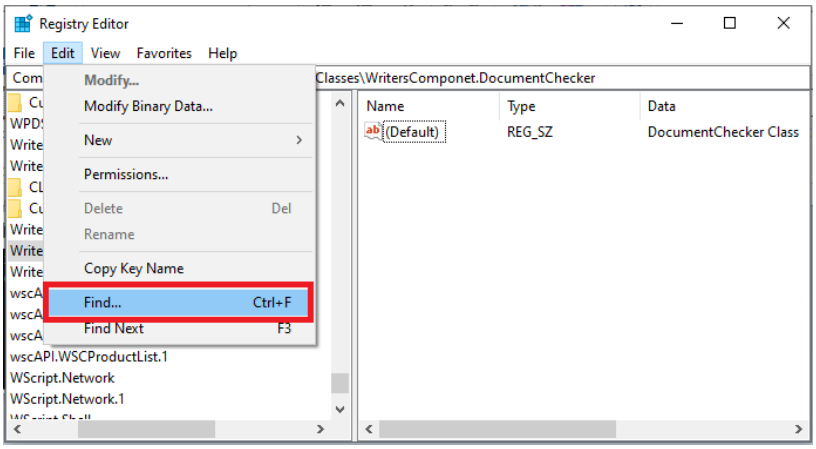
3.3.3 编译和运行 COM Client 程序 Nativecomclient

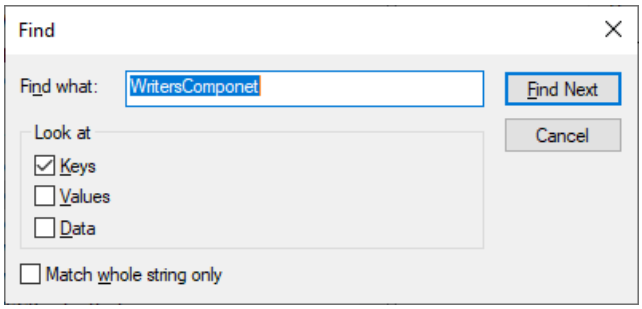
到此，COM Client 编程已经完成. 用F7建立程序，用 Ctl/F5 运行程序.

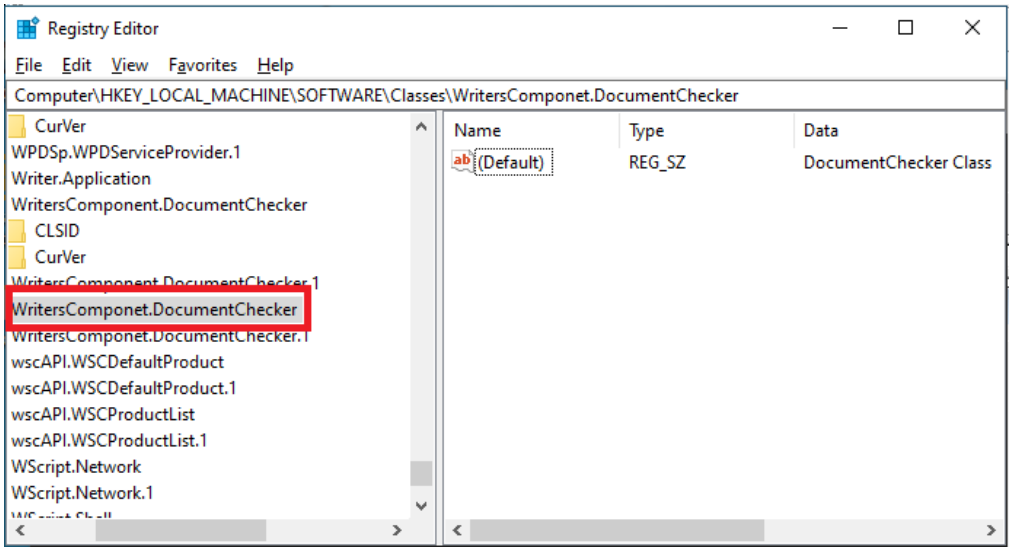
IV. Appendix A 查验正确的 COM 字符串

如想查验正确的字符串可用 Windows 10 utility “RegEdit”法显示.








Registry Editor
File Edit View Favorites Help
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\WritersComponet.DocumentChecker
CurVer
WPDSp.WPDServiceProvider.1
Writer.Application
WritersComponent.DocumentChecker
CLSID
CurVer
WritersComponet.DocumentChecker
wscAPI.WSCDefaultProduct
wscAPI.WSCDefaultProduct.1
wscAPI.WSCProductList
wscAPI.WSCProductList.1
WScript.Network
WScript.Network.1
Name
Type
Data
(Default)
REG_SZ
DocumentChecker Class