COM开发

潘爱民

http://www.icst.pku.edu.cn/compcourse

内容

- Win32 SDK和MFC介绍
- MFC对COM的支持
- 用MFC开发COM组件
- ATL对COM的支持
- 用ATL开发COM组件
- 布置作业

Win32 SDK： Windows程序结构

- 入口函数WinMain
- 应用初始化
- 主窗口的创建及显示
- 消息分发循环
- 程序结束处理

Win32 SDK对COM的支持

- Win32 SDK包括COM库函数的支持

Win32 SDK 提供的一些头文件的说明

头文件	说明
Unknwn.h	标准接口 IUnknown 和 IClassFactory 的 IID 及接口成员函数的定义。
Wtypes.h	包含了 COM 使用的数据结构的说明。
Objidl.h	所有标准接口的定义，既可用于 C 语言风格的定义，也可用于 C++语言风格的定义。
Comdef.h	所有标准接口以及 COM 和 OLE 内部对象的 CLSID。
ObjBase.h	所有的 COM API 函数的说明。
Ole2.h	所有经过封装的 OLE 辅助函数。

利用宏描述接口

```
DECLARE_INTERFACE_(IClassFactory, IUnknown)
    {
       STDMETHOD(QueryInterface) (THIS_
                        REFIID riid,
                        LPVOID FAR* ppvObj) PURE;
       STDMETHOD_(ULONG,AddRef) (THIS) PURE;
       STDMETHOD_(ULONG,Release) (THIS) PURE;

       STDMETHOD(CreateInstance) (THIS_
                        LPUNKNOWN pUnkOuter,
                        REFIID riid,
                        LPVOID FAR* ppvObject) PURE;
       STDMETHOD(LockServer)(THIS_
                        BOOL fLock) PURE;
    };
```

VC提供的用于描述接口的宏

与 COM 接口有关的一些宏的说明

宏	说明
DECLARE_INTERFACE(iface)	声明接口 iface，它不从其它的接口派生。
DECLARE_INTERFACE_(iface, baseiface)	声明接口 iface，它从接口 baseiface 派生。
STDMETHOD(method)	声明接口成员函数 method，函数返回类型为 HRESULT。
STDMETHOD_(type,method)	声明接口成员函数 method，函数返回类型为 type。

MFC基础

- 应用类
  - AfxGetApp
  - CWinApp::InitInstance
  - CWinApp::ExitInstance
  - CWinApp::OnIdle
  - CWinApp::Run
  - CWnd *m_pMainWnd
- 窗口类
  - AfxGetMainWnd

MFC的消息处理机制
——消息映射表

- 在CWnd派生类定义中加入声明：

  DECLARE_MESSAGE_MAP()

- 在类的实现文件中加入表和表项的定义：

  BEGIN_MESSAGE_MAP(theClass, baseClass)

  .....

  END_MESSAGE_MAP

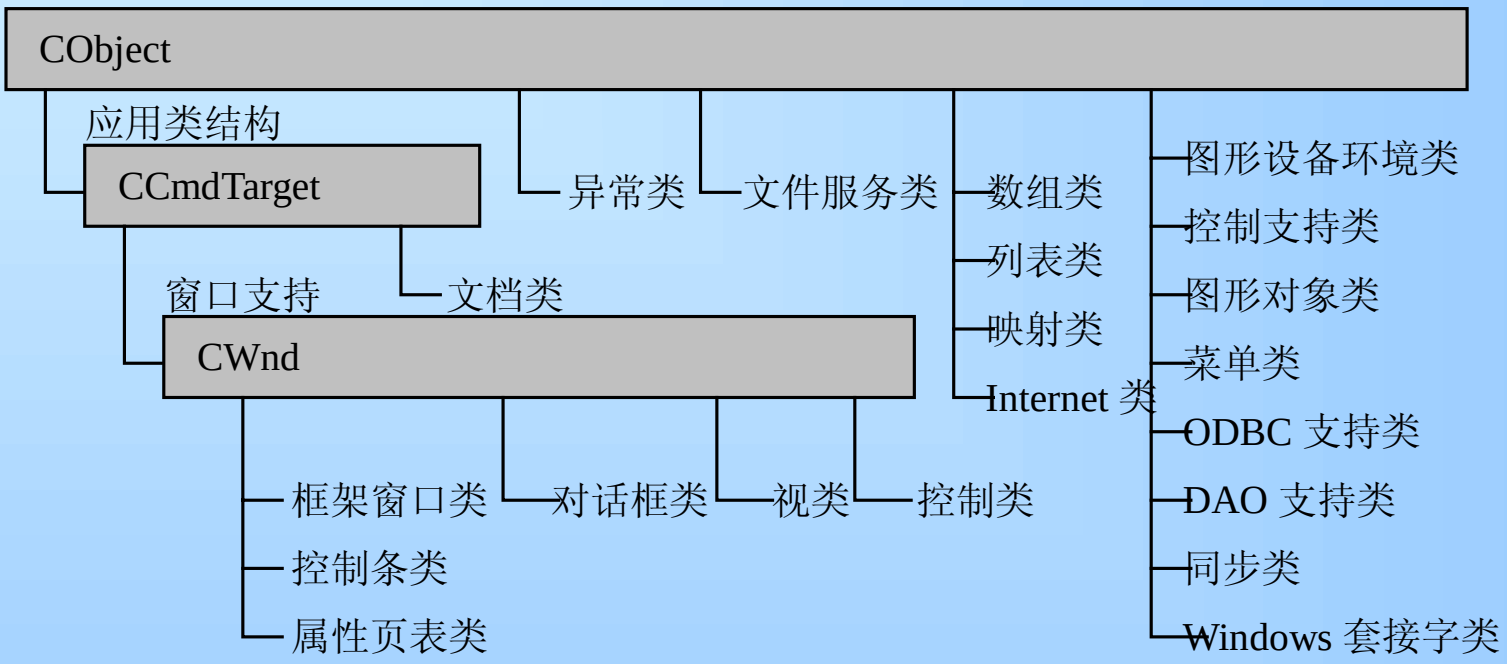
消息映射表示例

```
BEGIN_MESSAGE_MAP(theClass, baseClass)
    //{{AFX_MSG_MAP(theClass)
    ON_WM_SETFOCUS()
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_CLOSE()
    ON_WM_SIZE()
    ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
    ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
    ON_UPDATE_COMMAND_UI(ID_VIEW_STATUS_BAR,
    OnUpdateControlBarMenu)
    ON_NOTIFY_EX_RANGE(TTN_NEEDTEXTW, 0, 0xFFFF,
    OnToolTipText)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP
```

MFC应用类型

- 常规应用：MDI应用、SDI应用、基于对话框程序
- DLL应用：静态连接MFC库的正规DLL、动态连接MFC库的正规DLL、MFC扩展DLL
- 其他应用：
  - 支持OLE服务或者包容器的SDI应用
  - 支持OLE服务或者包容器的MDI应用
  - 支持自动化(Automation)服务的SDI或者MDI程序
  - ActiveX控制应用(OCX应用)

MFC库结构



其它类：

Internet 支持类

运行时刻对象支持

简单值类型

结构

其它支持类

集合模板类

自动化类型

用于同步的类

MFC对COM应用的支持

用嵌套类实现COM接口

```
class CDictionary
{
    …… //构造函数和析构函数
        HRESULT __stdcall  QueryInterface(REFIID iid, void** ppvObj);
        ULONG __stdcall AddRef();
        ULONG __stdcall Release();

      class XDictionaryObj : public IDictionary {
      public:
        CDictionary * m_pParent;
        virtual HRESULT __stdcall  QueryInterface(REFIID iid, void** ppvObj);
        virtual ULONG __stdcall AddRef();
        virtual ULONG __stdcall Release();
        virtual BOOL __stdcall Initialize();
        …...
        virtual void __stdcall FreeLibrary();
      } m_dictionaryObj;
```

未完

用嵌套类实现COM接口(续一)

续

```
class XSpellCheckObj : public ISpellCheck {
    public:
        CDictionary * m_pParent;
        virtual HRESULT __stdcall  QueryInterface(REFIID iid, void** ppvObj);
        virtual ULONG __stdcall AddRef();
        virtual ULONG __stdcall Release();
        virtual BOOL __stdcall CheckWord (String word, String *);
  } m_spellCheckObj;

    private :
        struct      DictWord *m_pData;
        char        *m_DictFilename[128];
        int         m_Ref ;
        int         m_nWordNumber, m_nStructNumber;
};
```

用嵌套类实现COM接口(续二)

```
CDictionary::CDictionary()
{
        .......      // Initializtion

        m_dictionaryObj. m_pParent = this;
        m_spellCheckObj. m_pParent = this;
}
```

用嵌套类实现COM接口(续三)

```
HRESULT  CDictionary::QueryInterface(const IID& iid, void **ppvObj)
{
     if (iid == IID_IUnknown || iid == IID_Dictionary)    {
          *ppvObj = &m_dictionaryObj;
          AddRef();
          return S_OK;
     }  else if (iid == IID_SpellCheck) {
          *ppvObj = &m_spellCheckObj;
          AddRef();
          return S_OK;
     }
     *ppv = NULL;
     return E_NOINTERFACE ;
}
```

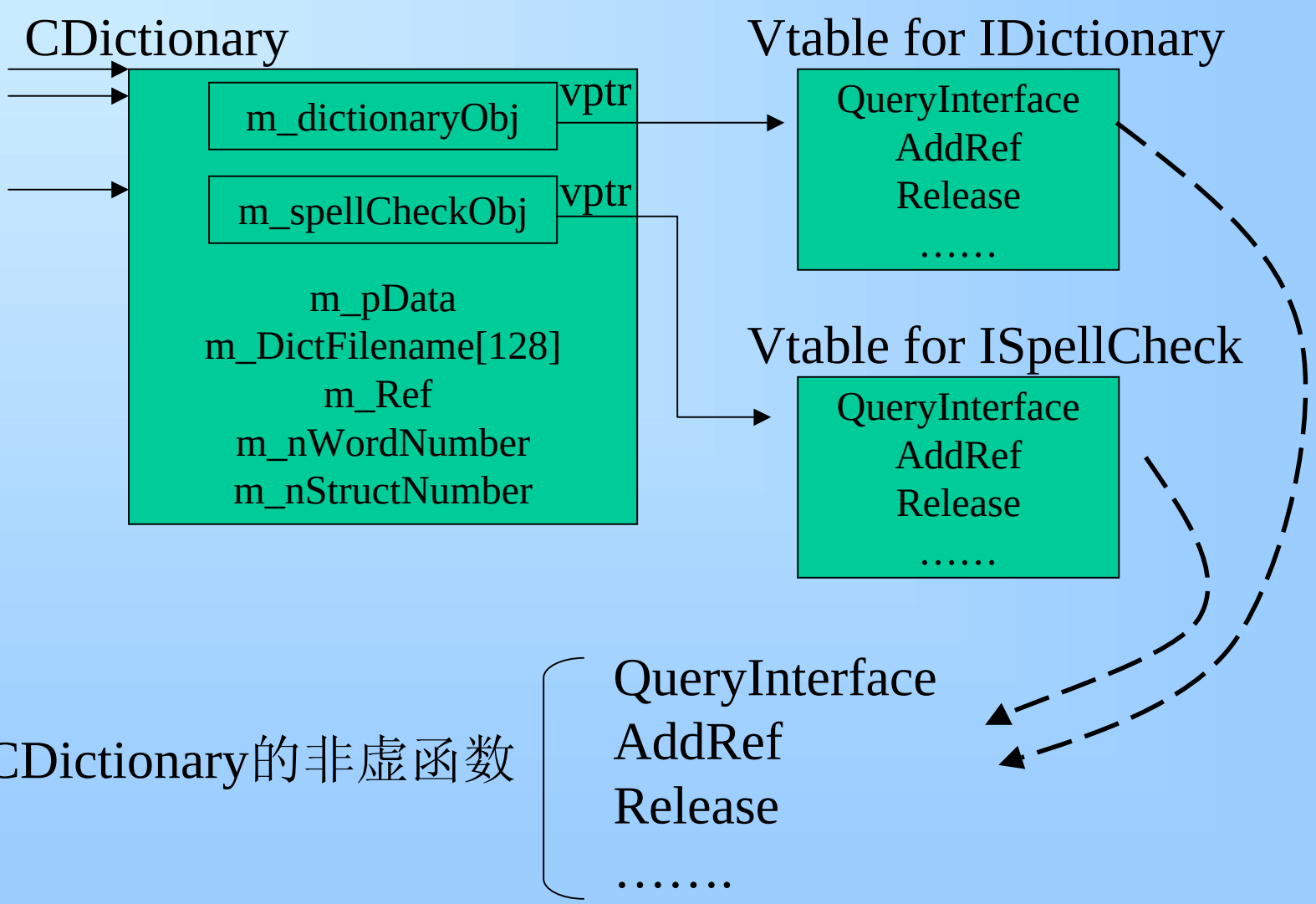
用嵌套类实现COM接口(续四)

```
ULONG CDictionary::XDictionaryObj::QueryInterface(const IID& iid,
                                                  void **ppvObj)
{
    return m_pParent->QueryInterface(iid, ppvObj);
}

ULONG CDictionary::XDictionaryObj::AddRef()
{
    return m_pParent->AddRef();
}

ULONG CDictionary::XDictionaryObj::Release ()
{
    return m_pParent->Release ();
}
```

“用嵌套类实现COM接口”原理



MFC：接口映射表

- CCmdTarget类
- CCmdTarget::m_dwRef为引用计数
- 接口映射表与消息映射表非常类似
- 接口映射表：记录了CCmdTarget类中每一个嵌套类的接口ID以及接口vtable与父类this指针之间的偏移量
- offsetof宏：成员类与父类之间的偏移值

DECLARE_INTERFACE_MAP

```
#define DECLARE_INTERFACE_MAP() \
private: \
    static const AFX_INTERFACEMAP_ENTRY _interfaceEntries[]; \
protected: \
    static AFX_DATA const AFX_INTERFACEMAP interfaceMap; \
    static const AFX_INTERFACEMAP* PASCAL _GetBaseInterfaceMap(); \
        virtual const AFX_INTERFACEMAP* GetInterfaceMap() const; \
```

```
struct AFX_INTERFACEMAP_ENTRY
{
    const void* piid;
    size_t nOffset;
};
```

```
struct AFX_INTERFACEMAP
{
#ifdef _AFXDLL
    const AFX_INTERFACEMAP* (PASCAL
    pfnGetBaseMap)();
#else
    const AFX_INTERFACEMAP* pBaseMap;
#endif
    const AFX_INTERFACEMAP_ENTRY* pEntry;
};
```

接口映射表定义

```
BEGIN_INTERFACE_MAP(CDictionary, CCmdTarget)
    INTERFACE_PART(CDictionary, IID_IDictionary, Dictionary)
    INTERFACE_PART(CDictionary, IID_ISpellCheck, SpellCheck)
END_INTERFACE_MAP()
```

接口映射表的宏定义

```
#define BEGIN_INTERFACE_MAP(theClass, theBase) \
    const AFX_INTERFACEMAP* PASCAL theClass::_GetBaseInterfaceMap() \
            { return &theBase::interfaceMap; } \
    const AFX_INTERFACEMAP* theClass::GetInterfaceMap() const \
            { return &theClass::interfaceMap; } \
    AFX_COMDAT const AFX_DATADEF \
                    AFX_INTERFACEMAP theClass::interfaceMap = \
            { &theClass::_GetBaseInterfaceMap, &theClass::_interfaceEntries[0], }; \
    AFX_COMDAT const AFX_DATADEF
                    AFX_INTERFACEMAP_ENTRY theClass::_interfaceEntries[] = \
    { \

#define INTERFACE_PART(theClass, iid, localClass) \
            { &iid, offsetof(theClass, m_x##localClass) }, \

#define END_INTERFACE_MAP() \
            { NULL, (size_t)-1 } \
    }; \
```

MFC版本的字典对象类定义

```
class CDictionary : public CCmdTarget
{
    DECLARE_DYNCREATE(CDictionary)

    CDictionary();          // protected constructor used by dynamic creation
        DECLARE_INTERFACE_MAP()
    ......
        // IDictionary
    BEGIN_INTERFACE_PART(Dictionary, IDictionary)
        INIT_INTERFACE_PART(CDictionary, Dictionary)
        STDMETHOD_(BOOL, Initialize)();
        ……
        STDMETHOD_(void, FreeLibrary)();
    END_INTERFACE_PART_STATIC(Dictionary)

        // ISpellCheck
    BEGIN_INTERFACE_PART(SpellCheck, ISpellCheck)
        INIT_INTERFACE_PART(CDictionary, SpellCheck)
        STDMETHOD_(BOOL, CheckWord)(LPOLESTR, LPOLESTR *);
    END_INTERFACE_PART_STATIC(SpellCheck)
};
```

MFC版本的字典对象类实现

```
STDMETHODIMP_(ULONG) CDictionary::XDictionary::AddRef()
{
    METHOD_PROLOGUE_EX_(CDictionary, Dictionary)
    return pThis->ExternalAddRef();
}
```

- **METHOD_PROLOGUE_EX_宏定义：**

```
#define METHOD_PROLOGUE_EX(theClass, localClass) \
    theClass* pThis = ((theClass*)((BYTE*)this - m_nOffset)); \
            AFX_MANAGE_STATE(pThis->m_pModuleState) \
            pThis; // avoid warning from compiler \
```

CCmdTarget类实现IUnknown

```
public:
        // data used when CCmdTarget is made OLE aware
    long m_dwRef;
    LPUNKNOWN m_pOuterUnknown;  // external controlling unknown if != NULL
    DWORD m_xInnerUnknown;  // place-holder for inner controlling unknown

public:
    // advanced operations
    void EnableAggregation();       // call to enable aggregation
    void ExternalDisconnect();      // forcibly disconnect
    LPUNKNOWN GetControllingUnknown();
                    // get controlling IUnknown for aggregate creation
```

CCmdTarget类实现IUnknown(续)

```
public:
    // these versions do not delegate to m_pOuterUnknown
    DWORD InternalQueryInterface(const void*, LPVOID* ppvObj);
    DWORD InternalAddRef();
    DWORD InternalRelease();

    // these versions delegate to m_pOuterUnknown
    DWORD ExternalQueryInterface(const void*, LPVOID* ppvObj);
    DWORD ExternalAddRef();
    DWORD ExternalRelease();
```

CCmdTarget中QueryInterface实现

```
DWORD CCmdTarget::InternalQueryInterface(const void* iid, LPVOID* ppvObj)
{
        // check local interfaces
    if ((*ppvObj = GetInterface(iid)) != NULL)    {
        // interface was found -- add a reference
        ExternalAddRef();
        return S_OK;
    }
        // check aggregates
    if ((*ppvObj = QueryAggregates(iid)) != NULL)
        return S_OK;

    // interface ID not found, fail the call
    return (DWORD)E_NOINTERFACE;
}
```

CCmdTarget中External*XXX*成员实现

```
DWORD CCmdTarget::ExternalAddRef()
{
    // delegate to controlling unknown if aggregated
    if (m_pOuterUnknown != NULL)
        return m_pOuterUnknown->AddRef();
    return InternalAddRef();
}

DWORD CCmdTarget::ExternalRelease() // …...

// QueryInterface that is exported to normal clients
DWORD CCmdTarget::ExternalQueryInterface(const void* iid, LPVOID* ppvObj)
{
    // delegate to controlling unknown if aggregated
    if (m_pOuterUnknown != NULL)
        return m_pOuterUnknown->QueryInterface(*(IID*)iid, ppvObj);
    return InternalQueryInterface(iid, ppvObj);
}
```

嵌套类内部实现IUnknown的成员函数

```
STDMETHODIMP_(ULONG) CDictionary::XDictionary::QueryInterface (
                                        const void* iid, LPVOID* ppvObj)
{
    METHOD_PROLOGUE_EX_(CDictionary, Dictionary)
        return pThis->ExternalQueryInterface (iid, ppvObj);
}
```

COM引出函数和类厂实现

- 在AppWizard中选中“Automation”检查框

```
STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return AfxDllGetClassObject(rclsid, riid, ppv);
}

STDAPI DllCanUnloadNow(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return AfxDllCanUnloadNow();
}

// by exporting DllRegisterServer, you can use regsvr.exe
STDAPI DllRegisterServer(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    COleObjectFactory::UpdateRegistryAll();
    return S_OK;
}
```

COleObjectFactory

- 通用的类厂，实现了IClassFactory2接口
- COleObjectFactory的主要信息是对象的CLSID和对象的类型信息。
- 它利用MFC的动态对象创建机制：
  - DECLARE_DYNCREATE
- 对象方面的支持：
  - DECLARE_OLECREATE(...)，定义如下

```
#define DECLARE_OLECREATE(class_name) \
public: \
    static AFX_DATA COleObjectFactory factory; \
    static AFX_DATA const GUID guid; \
```

MFC中组件对象的创建支持

- DECLARE_OLECREATE(...)
- IMPLEMENT_OLECREATE

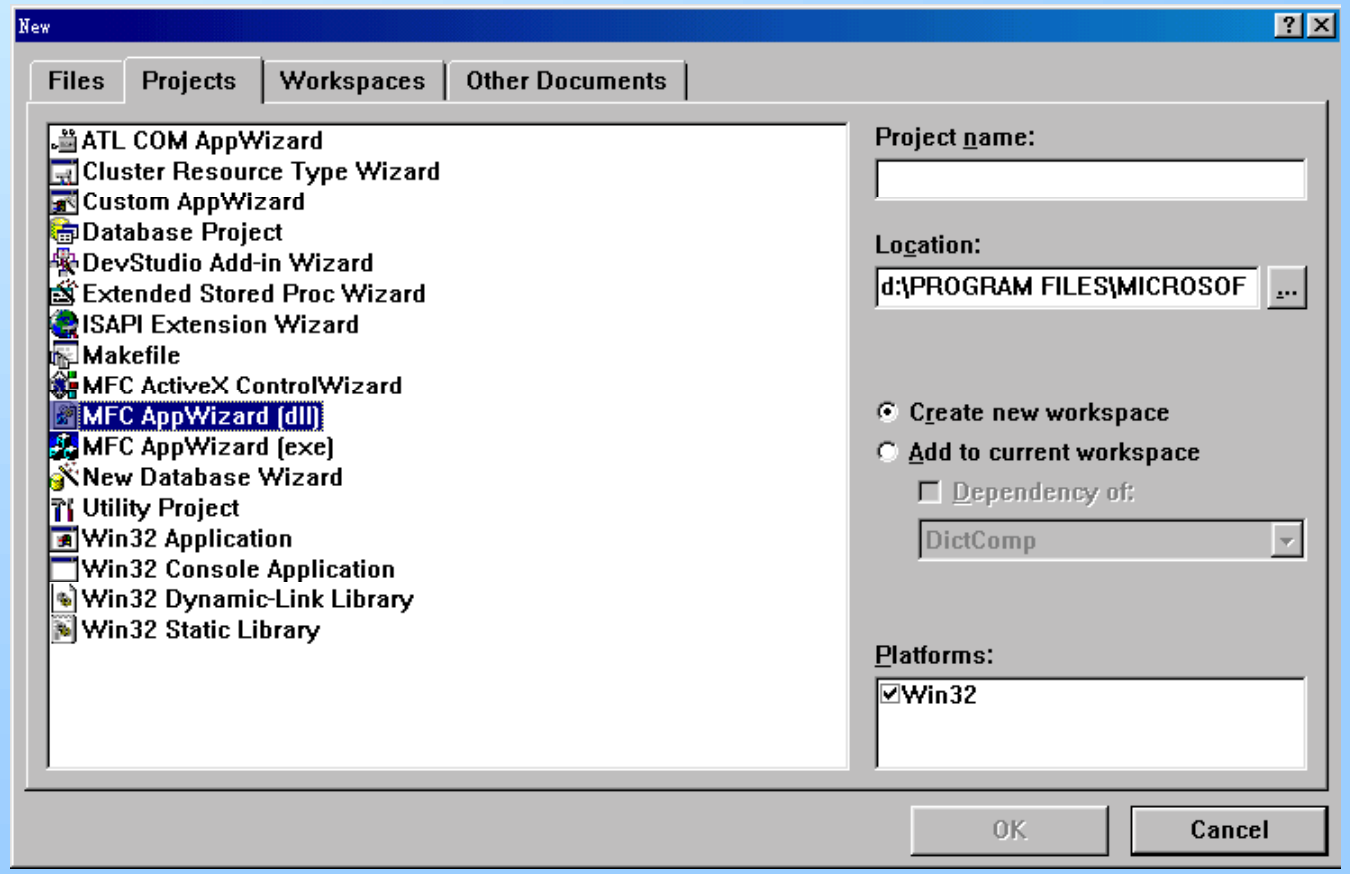
```
#define IMPLEMENT_OLECREATE(class_name, external_name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8) \
    AFX_DATADEF COleObjectFactory class_name::factory(class_name::guid, \
                RUNTIME_CLASS(class_name), FALSE, _T(external_name)); \
    AFX_COMDAT const AFX_DATADEF GUID class_name::guid = \
                { l, w1, w2, { b1, b2, b3, b4, b5, b6, b7, b8 } }; \
```

- 状态结构：AFX_MODULE_STATE，除了一些基本的全局信息，还包括一个类厂表。
- DllGetClassObject-〉AfxDllGetClassObject-〉AfxGetModuleState进一步得到类厂表
- 类厂对象的构造函数和析构函数维护类厂表

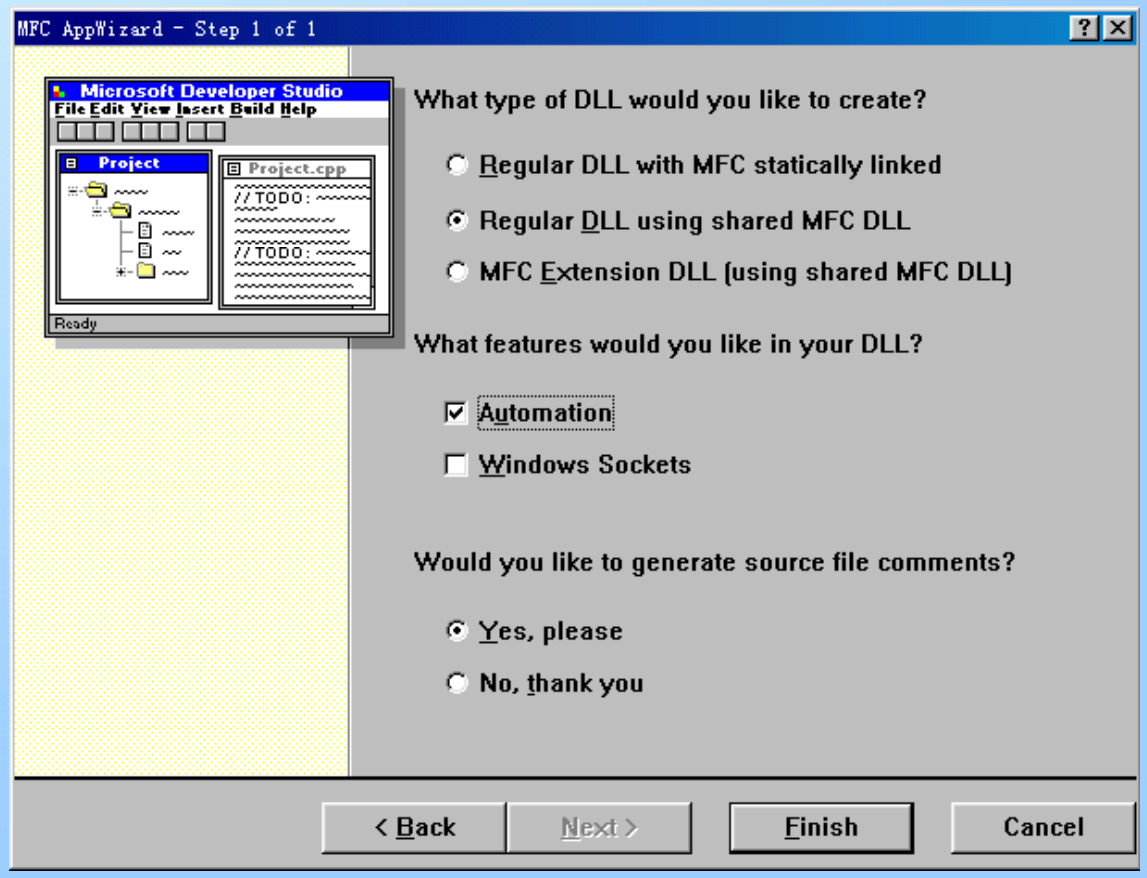
用MFC开发COM应用

- 利用AppWizard创建COM程序工程框架
- 利用ClassWizard添加COM对象类

AppWizard创建COM工程(一)



AppWizard创建COM工程(二)

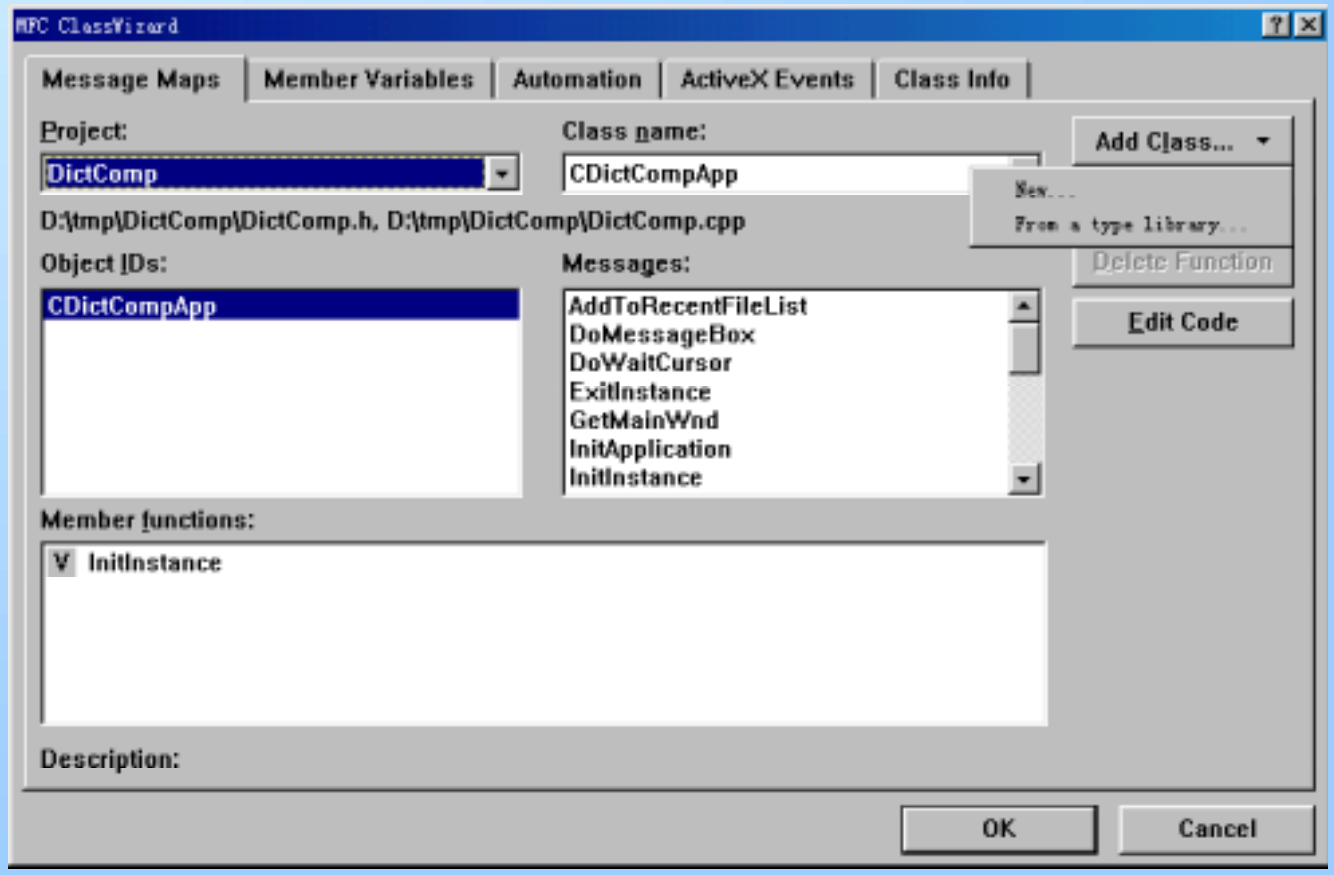


AppWizard创建COM工程(三)

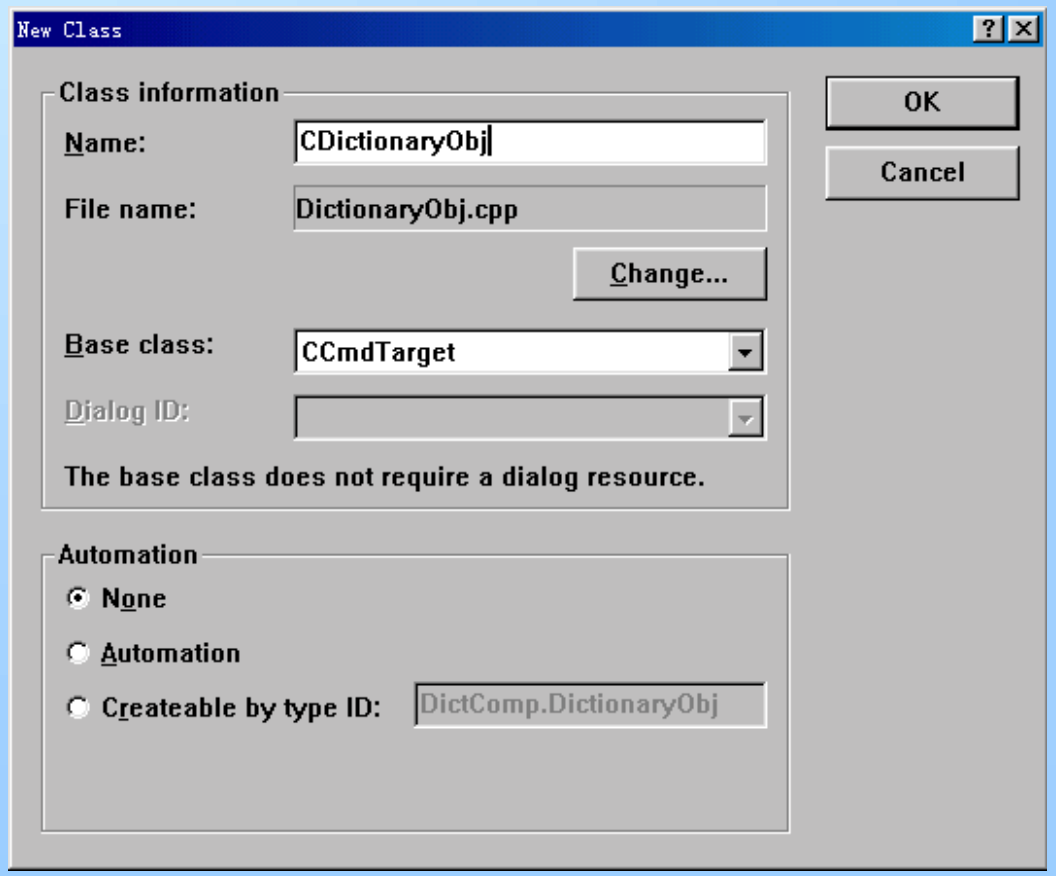
```
BOOL CDictCompApp::InitInstance()
{
        // Register all OLE server (factories) as running.  This enables the
        //  OLE libraries to create objects from other applications.
        COleObjectFactory::RegisterAll();

        return TRUE;
}
```

ClassWizard添加COM对象类(一)



ClassWizard添加COM对象类(二)



CDictionaryObj声明中加入接口定义

```
BEGIN_INTERFACE_PART(Dictionary, IDictionary)
        INIT_INTERFACE_PART(CDictionary, Dictionary)
        STDMETHOD_(BOOL, Initialize)();
        STDMETHOD_(BOOL, LoadLibrary)(LPOLESTR);
        STDMETHOD_(BOOL, InsertWord)(LPOLESTR, LPOLESTR);
        STDMETHOD_(void, DeleteWord)( LPOLESTR);
        STDMETHOD_(BOOL, LookupWord)(LPOLESTR, LPOLESTR *);
        STDMETHOD_(BOOL, RestoreLibrary)(LPOLESTR);
        STDMETHOD_(void, FreeLibrary)();
END_INTERFACE_PART_STATIC(Dictionary)

        // ISpellCheck
BEGIN_INTERFACE_PART(SpellCheck, ISpellCheck)
        INIT_INTERFACE_PART(CDictionary, SpellCheck)
        STDMETHOD_(BOOL, CheckWord)(LPOLESTR, LPOLESTR *);
END_INTERFACE_PART_STATIC(SpellCheck)

DECLARE_INTERFACE_MAP()
```

CDictionaryObj类实现文件中加入相应的定义

```
extern "C" const IID IID_Dictionary =
        { 0x54bf6568, 0x1007, 0x11d1,
        { 0xb0, 0xaa, 0x44, 0x45, 0x53, 0x54, 0x00, 0x00} } ;

extern "C" const IID IID_SpellCheck =
        { 0x54bf6569, 0x1007, 0x11d1,
        { 0xb0, 0xaa, 0x44, 0x45, 0x53, 0x54, 0x00, 0x00} } ;

BEGIN_INTERFACE_MAP(CDictionaryObj, CCmdTarget)
    INTERFACE_PART(CDictionaryObj, IID_IDictionary, Dictionary)
    INTERFACE_PART(CDictionaryObj, IID_ISpellCheck, SpellCheck)
END_INTERFACE_MAP()
```

类厂支持

- 在CDictionaryObj声明中加入：

```
DECLARE_OLECREATE(CDictionaryObj)
```

- 在CDictionaryObj实现文件中加入：

```
// {54BF6567-1007-11D1-B0AA-444553540000}
IMPLEMENT_OLECREATE(CDictionaryObj, "  Dictionary.Object ",
                0x54bf6567, 0x1007, 0x11d1, 0xb0, 0xaa, 0x44,
                0x45, 0x53, 0x54, 0x00, 0x00)
```

ATL介绍

- ATL实现COM的机制完全不同于MFC
- 使用多继承技术实现多个接口
- 支持多线程
- 实现QueryInterface用到了特殊的技术
- 创建对象机制不同于以往的技术
- 优化

ATL概况

- 封装了一些数据类型
  - CComBSTR、CComVariant、CComPtr，等
- 实现COM接口和COM对象
  - 接口映射表、对象映射表，等
- 窗口的支持
  - CWindow、CWindowImpl、CDialogImpl，等
- 其他COM特征的支持
  - 永久性支持
  - 连接点支持
  - 集合对象和枚举器对象
  - ActiveX control and container
  - 等

CComBSTR

- 封装了BSTR类型
- 提供了大量便利的字符串操作
  - 构造函数
  - 各种操作符以及一般的字符串操作
  - 对于流(stream)的支持
- 在需要BSTR的地方，都可以用CComBSTR来代替
- 注意owership

CComVariant

- 封装了VARIANT属性
- 提供了常用的操作
  - 构造函数
  - 各种操作符以及一般的管理操作
  - 对于流(stream)的支持
- 在需要VARIANT的地方，都可以用CComVARIANT来代替

CComPtr、CComQIPtr

- Smart pointer

```
template<class T>
class CComPtr
{
    public:
    T* p;
    …
};
```

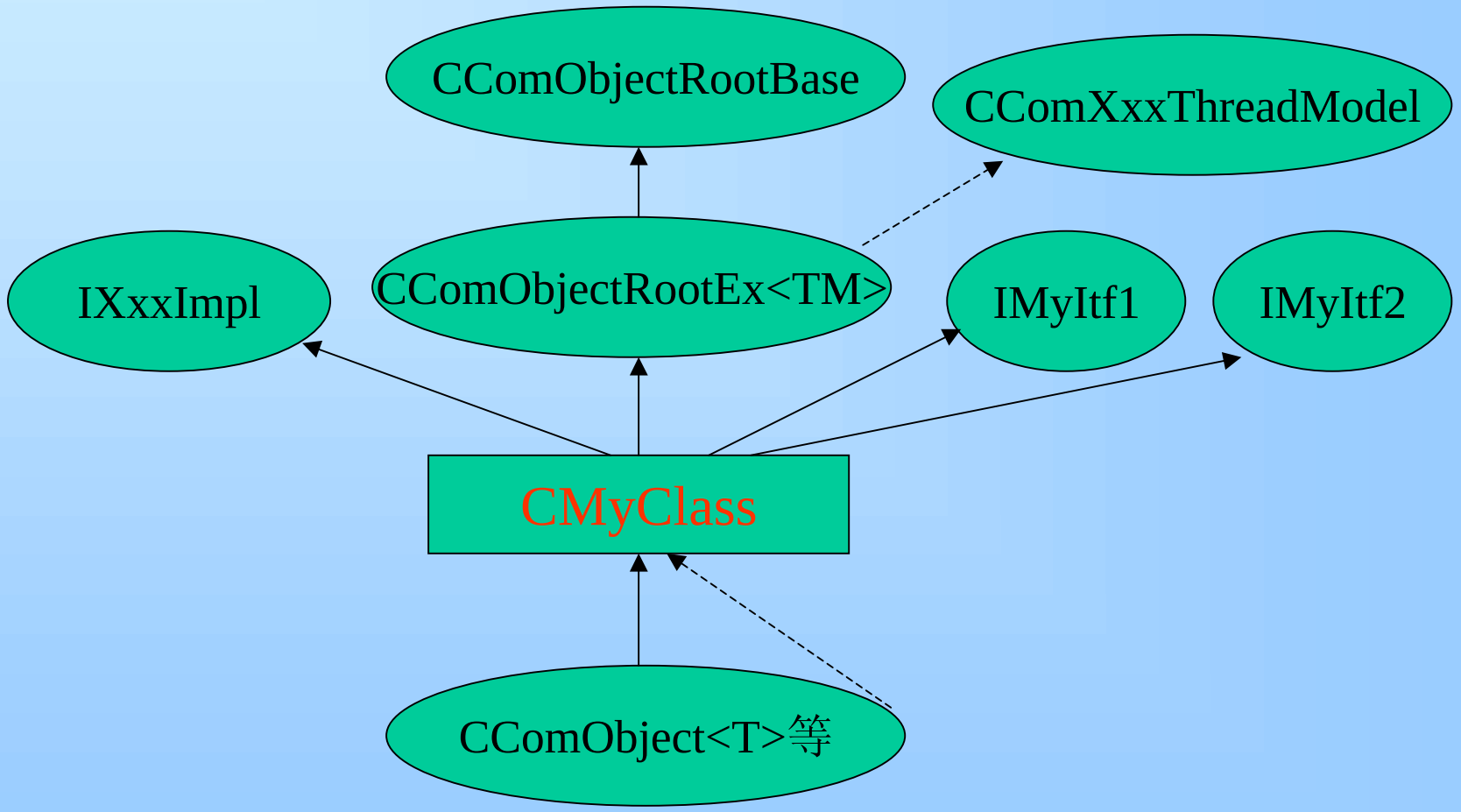
```
template<class T, const IID* piid = &__uuidof(T)>
class CComQIPtr
{
public:
T* p;
...
};
```

- 优点：
  - 自动管理AddRef/Release
  - 在大多数情况下，可以当作接口指针来使用
- 注意：禁止调用“->Release”和“->AddRef”

CComDispatchDriver

- 封装了IDispatch接口
- 除了对接口指针的管理之外，有下面的功能：
  - 属性访问函数：
    - GetIDOfName/ GetProperty/ PutProperty
    - GetPropertyByName/ PutPropertyByName
  - 方法访问函数：
    - by DISPID：Invoke0/Invoke1/Invoke2/InvokeN
    - by Name：Invoke0/Invoke1/Invoke2/InvokeN
  - 两个静态函数：
    - By DISPID：GetProperty/PutProperty

ATL的类层次


CComObjectRootBase
CComXxxThreadModel
IXxxImpl
CComObjectRootEx<TM>
IMyItf1
IMyItf2
CMyClass
CComObject<T>等

CComObjectRootBase

- ObjectMain
- static InternalQueryInterface
- OuterAddRef/OuterRelease/OuterQueryInterface
- InternalFinalConstructAddRef/ InternalFinalConstructRelease
- 其他一些静态函数
- 联合：

```
union
    {
      long m_dwRef;
      IUnknown* m_pOuterUnknown;
    };
```

ATL对象的线程模型

- 用到了trait技术
- 通过编译时刻的类型提供just thread-safe enough
- CComSingleThreadModel
- CComMultiThreadModel
- CComMultiThreadNoCS
- 提供了两个静态成员函数和三个typedef
  - Increment、Decrement
  - AutoCriticalSection、CriticalSection、ThreadModelNoCS

ATL对象实现引用计数

- CComObjectRootEx
  - InternalAddRef
  - InternalRelease
  - 作用在匿名联合的m_dwRef成员上
- CComObjectRootEx定义了一把锁
  - 锁的类型为AutoCriticalSection
  - 对锁的封装ObjectLock，wrapper
- 用于未被聚合的情况下

ATL对象实现QueryInterface

- Table-driven QueryInterface
- Interface Map

  BEGIN_COM_MAP(class)

  COM_INTERFACE_ENTRY(itf)

  END_COM_MAP
- 表中每一项

```
struct _ATL_INTMAP_ENTRY {
        const IID*  piid;
        DWORD    dw;
        _ATL_CREATORARGFUNC *pFunc;
};
```

ATL实现的接口类

- IDispatchImpl
- IPersistStreamInitImpl
- IConnectionPointContainerImpl
- 举例：

```
template<class T, const IID* piid, …> class IDispatchImpl : public T {…};
template <class T> class IConnectionPointContainerImpl : public
    IConnectionPointContainer{...};

class CMyObject :
    public IDispatchImpl<IMyDispInterface, &IID_IMyDispInterface,... >,
    public IConnectionPointContainerImpl< CMyObject >
{…};
```

真正的ATL对象类

- 决定这个COM对象如何被分配，是否被聚合等
- 区别：
  - 线程模型是以每个类为基础的，per-class，可以封装到基类中
  - 对象的生命周期和身份标识是以每个对象实例为基础的，per-object，要延后到最终的派生类最初决定
- CComObject类：
  - template <class Base> CComObject : public Base {…};
  - 支持聚合：class CComAggObject;
  - 支持聚合： class CComPolyObject;
  - template <class Base> CComObjectCached : public Base {…};
  - template <class Base> CComObjectNoLock : public Base {…};
  - template <class Base> CComObjectGlobal : public Base {…};
  - template <class Base> CComObjectStack : public Base {…};

ATL对象的创建

- 两个步骤：
  - 使用CRT的构造器
  - FinalConstruct
- 对应于FinalConstruct有FinalRelease
- 举例：

```
CMyClassFactory::CreateInstance() {
    ……
    CComObject<CMyObject> *pObj = new CComObject<CMyObject>;
    …...
    pObj->InternalFinalConstructAddRef();
    HRESULT hr = FinalConstruct();
    pObj->InternalFinalConstructRelease();
    ……
}
```

ATL Creators

- 每个creator类有一个静态CreateInstance函数：

  HRESULT WINAPI CreateInstance(void* pv, REFIID riid, LPVOID* ppv);

- 举例：

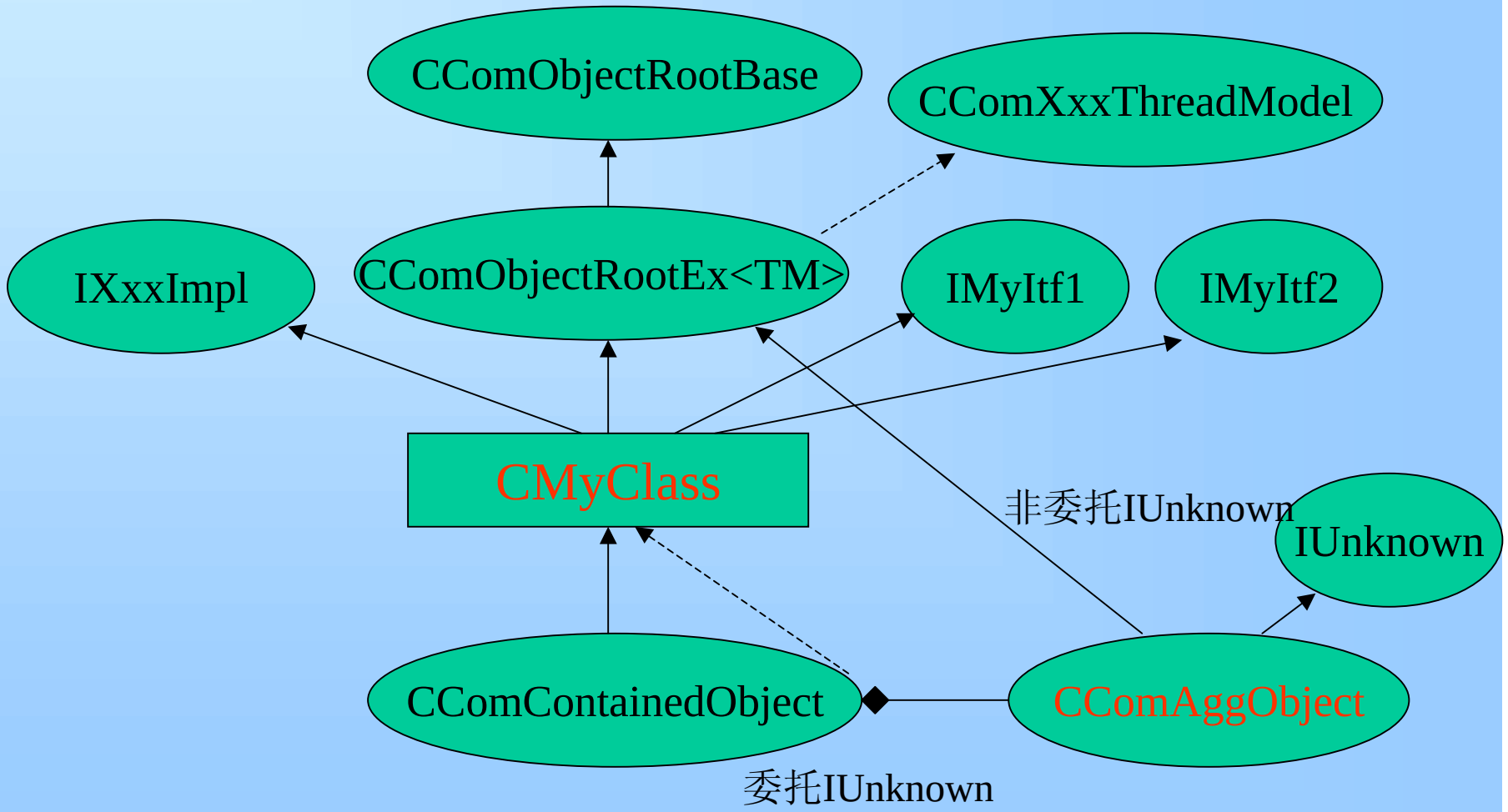
```
template <class T> class CComCreator {
public:
    static HRESULT WINAPI CreateInstance(void* pv, REFIID riid, LPVOID* ppv) {
      ……
      T *pObj = new T(pv);
      …...
      pObj->InternalFinalConstructAddRef();
      HRESULT hr = FinalConstruct();
      pObj->InternalFinalConstructRelease();
      ……
      hr = p->QueryInterface(riid, ppv);
      ……
    }
};
```

ATL Creators(续)

- 其他的creator类有
  - CComCreator2 —— 根据pv参数是否为null从两个对象类中择一
  - CComFailCreator —— 假的创建类
- 在CMyObject类中定义一个类型_CreatorClass，例如
  - typedef CComCreator<CComObject<CMyObject>> _CreatorClass;
- CComCoClass定义：

```
template<class T, const CLSID* pclsid = &CLSID_NULL>
class CComCoClass {
    public:
    ……
    template<class Q> static HRESULT CreateInstance(IUnknown* punkOuter, Q** pp)
    { return T::_CreatorClass::CreateInstance(punkOuter, __uuidof(Q), (void **)pp); }
    ……
};
```

聚合情况下对象结构图


CComObjectRootBase
CComXxxThreadModel
IXxxImpl
CComObjectRootEx<TM>
IMyItf1
IMyItf2
CMyClass
非委托IUnknown
IUnknown
CComContainedObject
CComAggObject
委托IUnknown

ATL中对象聚合的实现

```
template <class contained>
class CComAggObject :
    public IUnknown,
    public CComObjectRootEx< contained::_ThreadModel::ThreadModelNoCS >
{
public :
    STDMETHOD_(ULONG, AddRef)() {…}
    STDMETHOD_(ULONG, Release)() {…}
    STDMETHOD(QueryInterface)(REFIID iid, void ** ppvObject){…}
    ……
    CComContainedObject<contained> m_contained;
};
```

非委托IUnknown

委托IUnknown

ATL中对象聚合的实现(续)

```
template <class Base> //Base must be derived from CComObjectRoot
class CComContainedObject : public Base
{
public:
    typedef Base _BaseClass;
    CComContainedObject(void* pv) {m_pOuterUnknown = (IUnknown*)pv;}
    STDMETHOD_(ULONG, AddRef)() {return OuterAddRef();}
    STDMETHOD_(ULONG, Release)() {return OuterRelease();}
    STDMETHOD(QueryInterface)(REFIID iid, void ** ppvObject)
    { …… // 调用 OuterQueryInterface(iid, ppvObject); }
    ……
    IUnknown* GetControllingUnknown() { … }
};
```

// CComObjectRootBase基类中联合成员m_pOuterUnknown起作用

接口映射表项类型

- COM_INTERFACE_ENTRY
- COM_INTERFACE_ENTRY_IID(iid, x)
- COM_INTERFACE_ENTRY2(x, x2)
- COM_INTERFACE_ENTRY2_IID(iid, x, x2)
- COM_INTERFACE_ENTRY_FUNC(iid, dw, func)
- COM_INTERFACE_ENTRY_FUNC_BLIND(dw, func)
- COM_INTERFACE_ENTRY_TEAR_OFF(iid, x)
- COM_INTERFACE_ENTRY_CACHED_TEAR_OFF(iid, x, punk)
- COM_INTERFACE_ENTRY_AGGREGATE(iid, punk)
- COM_INTERFACE_ENTRY_AGGREGATE_BLIND(punk)
- COM_INTERFACE_ENTRY_CHAIN(classname)

ATL Servers

- 功能
  - Register and Unregister all class
  - exposing class object
  - managing server’s lifetime
- ATL实现结构
  - object map
  - CComModule

Object Map

- 示例

```
BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_DictionaryObj, CDictionaryObj)
OBJECT_ENTRY_NON_CREATEABLE(COtherObj)
END_OBJECT_MAP()
```

- 宏定义：

```
#define BEGIN_OBJECT_MAP(x) static _ATL_OBJMAP_ENTRY x[] = {
#define END_OBJECT_MAP()   {NULL, NULL, NULL, NULL, NULL,
    NULL, NULL, NULL}};
```

_ATL_OBJMAP_ENTRY定义

```
struct _ATL_OBJMAP_ENTRY
{
    const CLSID* pclsid;
    HRESULT (WINAPI *pfnUpdateRegistry)(BOOL bRegister);
    _ATL_CREATORFUNC* pfnGetClassObject;
    _ATL_CREATORFUNC* pfnCreateInstance;
    IUnknown* pCF;
    DWORD dwRegister;
    _ATL_DESCRIPTIONFUNC* pfnGetObjectDescription;
    _ATL_CATMAPFUNC* pfnGetCategoryMap;
    void (WINAPI *pfnObjectMain)(bool bStarting);
};
```

OBJECT_ENTRY定义

```
#define OBJECT_ENTRY(clsid, class) {&clsid,
    class::UpdateRegistry,
    class::_ClassFactoryCreatorClass::CreateInstance,
    class::_CreatorClass::CreateInstance, NULL, 0,
    class::GetObjectDescription, class::GetCategoryMap,
    class::ObjectMain },

#define OBJECT_ENTRY_NON_CREATEABLE(class)
    {&CLSID_NULL, class::UpdateRegistry, NULL, NULL,
    NULL, 0, NULL, class::GetCategoryMap,
    class::ObjectMain },
```

类的注册

- OBJECT_ENTRY中的class::UpdateRegistry项
  - 要求每个类都要提供UpdateRegistry成员
  - 宏：
    - DECLARE_NO_REGISTRY()
    - DECLARE_REGISTRY(class, pid, vpid, nid, flags)
    - DECLARE_REGISTRY_RESOURCE(x)
    - DECLARE_REGISTRY_RESOURCEID(x)
- 缺省生成的工程使用资源进行注册
  - Registry Script File

类 厂

- OBJECT_ENTRY宏包含：

  class::_ClassFactoryCreatorClass::CreateInstance

- DECLARE_CLASSFACTORY_EX宏定义：

```
#define DECLARE_CLASSFACTORY_EX(cf) \
 typedef CComCreator< CComObjectCached< cf > > \
 _ClassFactoryCreatorClass;
#define DECLARE_CLASSFACTORY()  \
DECLARE_CLASSFACTORY_EX(CComClassFactory)
```

- CComCoClass定义中包含

  DECLARE_CLASSFACTORY()

类实例的创建

- OBJECT_ENTRY宏包含：

  class::_CreatorClass::CreateInstance

- DECLARE_AGGREGATABLE(x)宏定义：

```
#define DECLARE_AGGREGATABLE(x) public:\
    typedef CComCreator2< CComCreator< CComObject< x > >,  \
    CComCreator< CComAggObject< x > > > _CreatorClass;
```

- CComCoClass定义中包含

  DECLARE_AGGREGATABLE(T)

类厂与类实例的连接

- 类厂的初始化
  - in-proc server，DllGetClassObject
  - out-of-proc server，RegisterClassObject
- 把实例创建函数传递给类厂
  - CComClassFactory类具有以下成员：

    _ATL_CREATORFUNC* m_pfnCreateInstance;
- 类厂的CreateInstance方法调用m_pfnCreateInstance

CComModule

- 全局变量
  - ATL inproc server：CComModule _Module;
  - ATL local server：CExeModule _Module;
  - service-base server：CServiceModule _Module;
- Init/Term函数
- 注册功能
- 提供全局锁功能

编译优化

- ATL_NO_VTABLE
  - 阻止在构造/析构过程中调整vptr，由于纯虚函数的vtable引用只是被构造/析构函数访问，所以这会导致链接器优化掉纯虚函数的vptr
- _ATL_MIN_CRT
  - 不链接标准C/C++运行库
- _ATL_DLL
  - 动态链接atl.dll
- _ATL_STATIC_REGISTRY
  - 静态链接组件注册功能

字典类的ATL对象

```
class  CDictionary :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CDictionary, &CLSID_Dictionary>,
        public IDictionary,
        public ISpellCheck
{
public:
        CDictionary() { }

        DECLARE_REGISTRY_RESOURCEID(IDR_DICTIONARY)

        DECLARE_PROTECT_FINAL_CONSTRUCT()

        BEGIN_COM_MAP(CDictionary)
                COM_INTERFACE_ENTRY(IDictionary)
                COM_INTERFACE_ENTRY(ISpellCheck)
        END_COM_MAP()
```

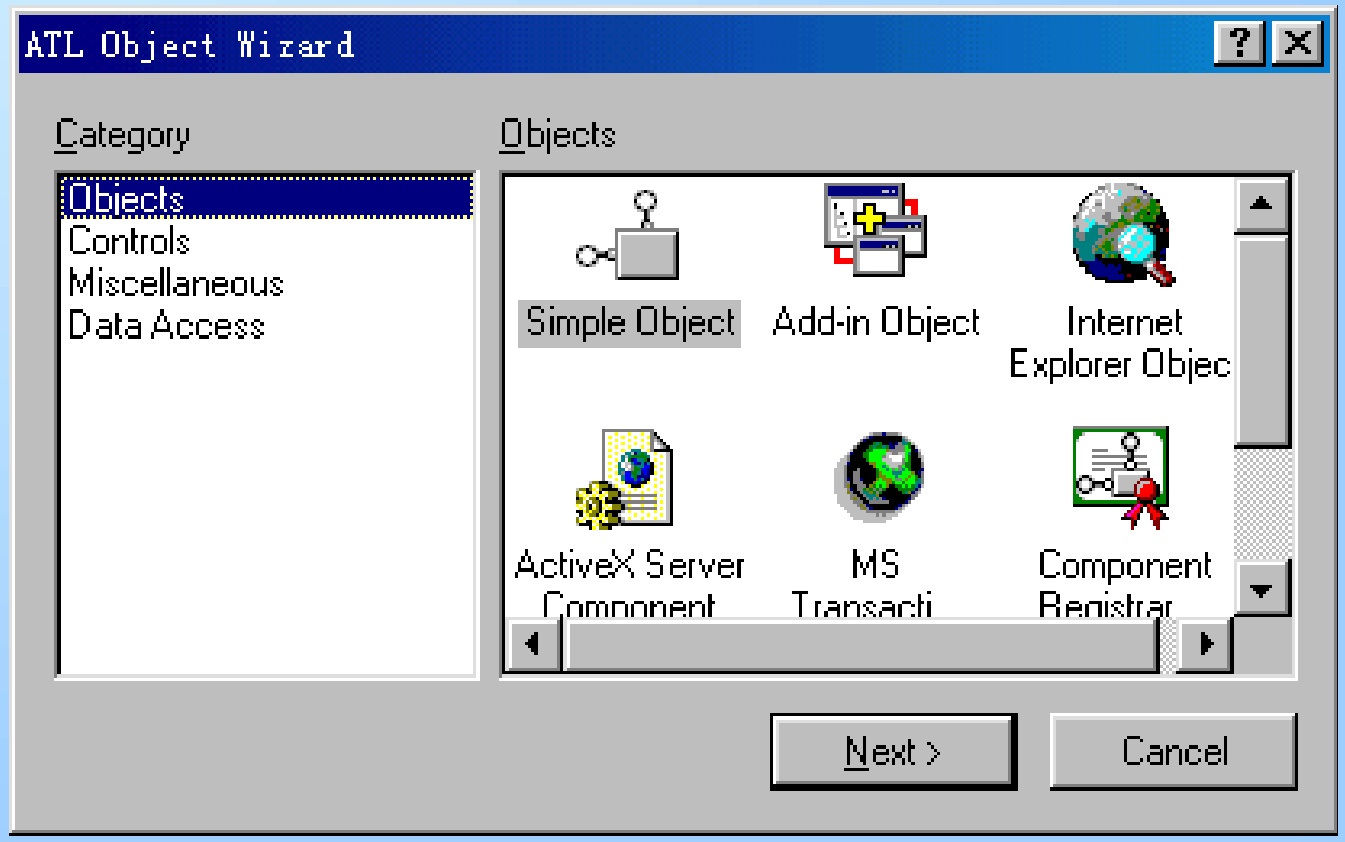
字典类的ATL对象(续)

```
public:
// IDictionary
        STDMETHOD_(BOOL, Initialize)();
        STDMETHOD_(BOOL, LoadLibrary)(LPOLESTR);
        STDMETHOD_(BOOL, InsertWord)(LPOLESTR, LPOLESTR);
        STDMETHOD_(void, DeleteWord)( LPOLESTR);
        STDMETHOD_(BOOL, LookupWord)(LPOLESTR, LPOLESTR *);
        STDMETHOD_(BOOL, RestoreLibrary)(LPOLESTR);
        STDMETHOD_(void, FreeLibrary)();

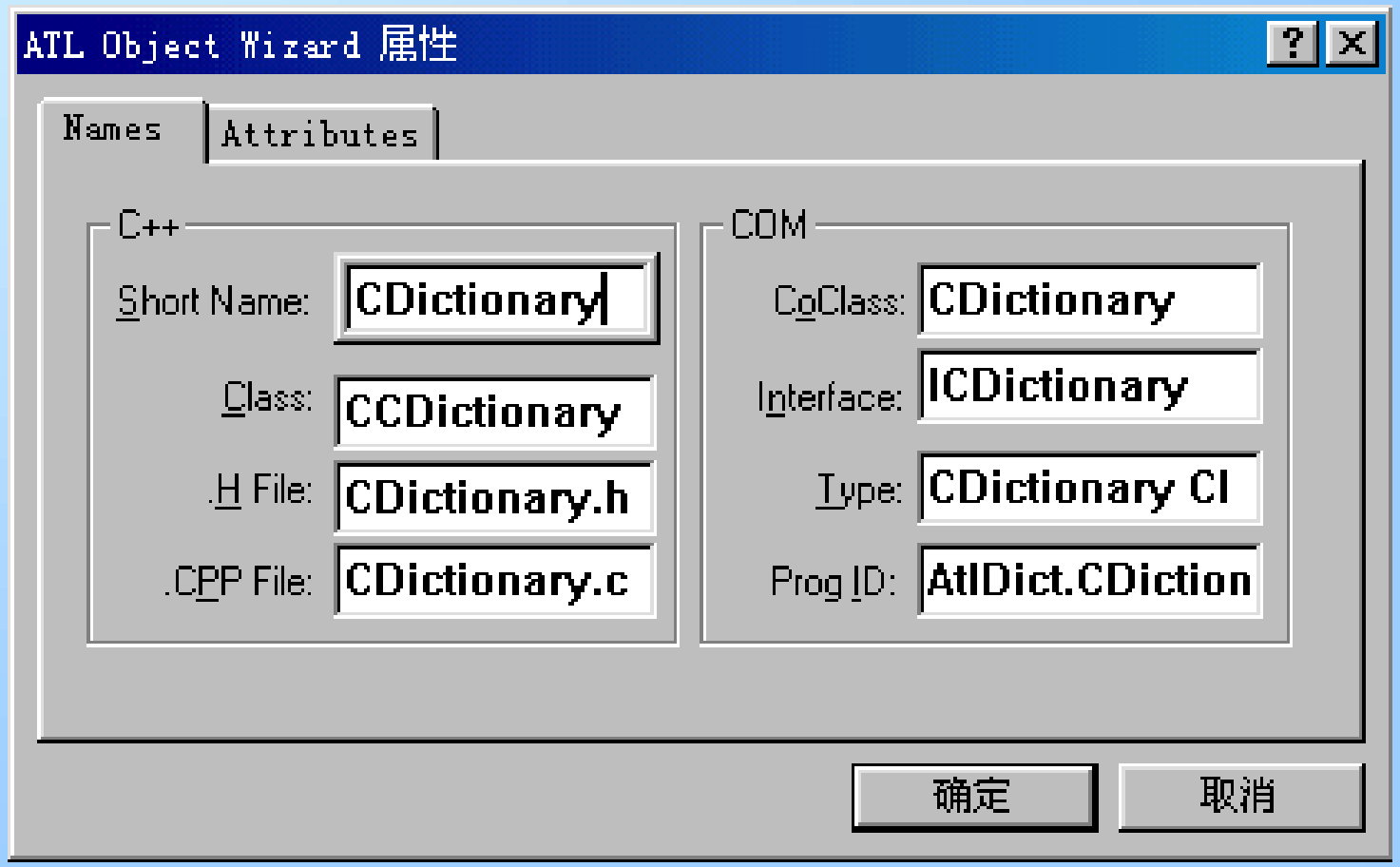
// ISpellCheck
        STDMETHOD_(BOOL, CheckWord)(LPOLESTR, LPOLESTR *);

private:
        ……
};
```

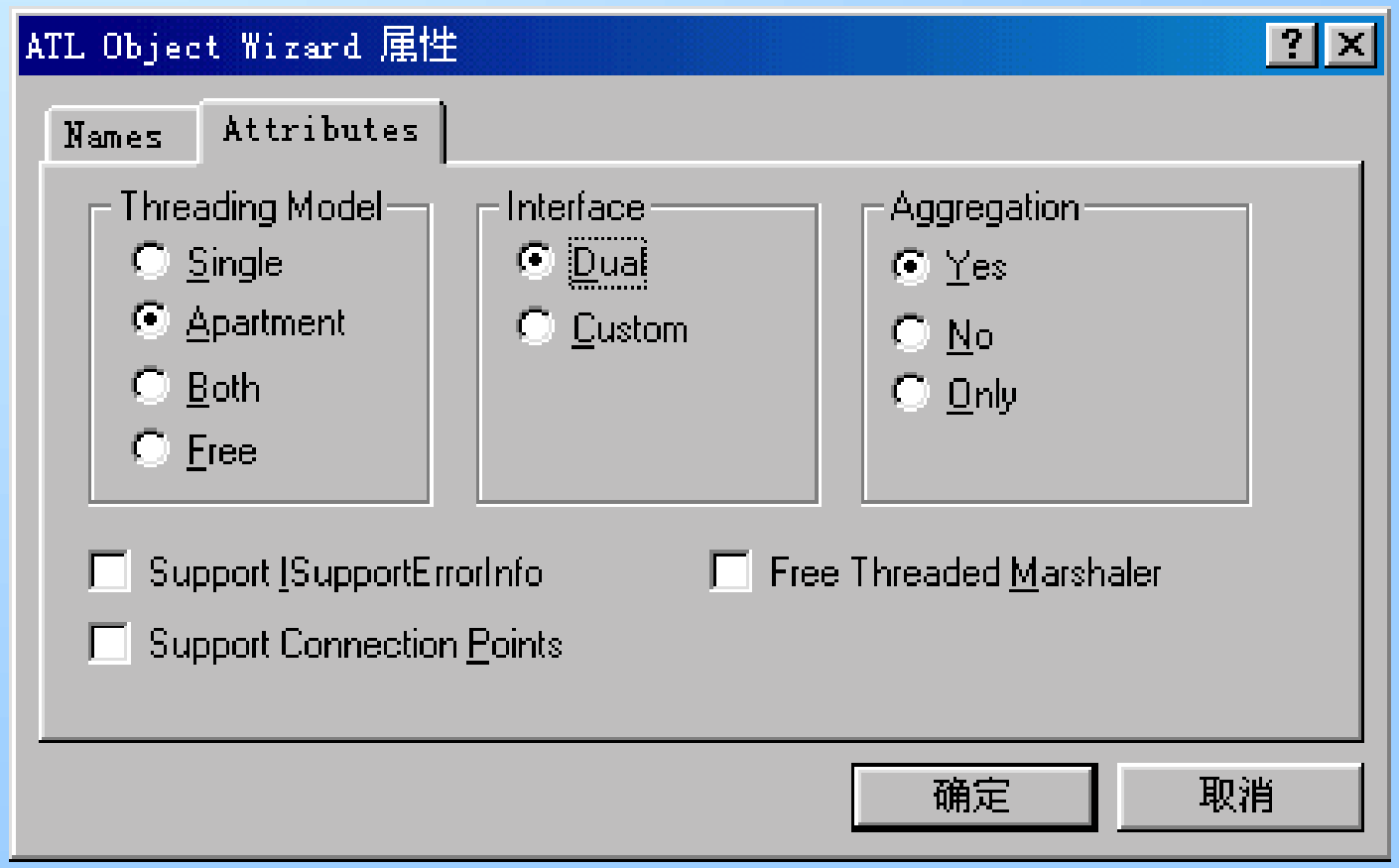
通过 ATL Object Wizard创建对象



通过 ATL Object Wizard设置对象名字



通过 ATL Object Wizard设置对象属性



演示和作业