## [COM](#目录)

### 概述

1991年Microsoft引入OLE对象链接与嵌入技术（容器嵌入组件，容器链接文档），比如excel表格放到word中，OLE1版本推广了文档可以作为其他程序功能组件的容器的思想；

1993年Microsoft推出COM，并用COM重建了OLE，形成OLE2版本，从此OLE技术不再局限于对象链接与嵌入，功能逐渐强大，变成桌面系统上程序通信的一个技术统称了，比如OLE剪贴板、OLE可视化编辑、OLE拖放、OLE自动化、OLE控件；

1995年，Microsoft为Internet推出了ActiveX（把OLE技术作为ActiveX销售，ActiveX加入了对Internet的支持）；

1996年，Microsoft发布DCOM（分布式），为通过网络连接的计算机在二进制级别上提供对象通信服务；

1997年，COM和MTS（微软事务服务器）开发小组合并，发布COM+，增强了COM和DCOM；

**组件的优势**：

1. 二进制可重用，搭积木式开发
2. 分布式
3. 语言无关，用C++开发的COM，很容易在Java等其他语言中使用
4. 进程透明

想想看，现在的软件大多要多个产商互相配合，模块已不再是代码集了，而是自给自足的组件，它们可以在一台机器、局域网、Internet的任何机器上；

**应用：**

1. Windows操作系统已经大量使用，如DirectX软件包，DAO数据访问对象（VS支持OLE DB/ADO开发），网络库。。。

### 组件化的思想

#### 一个实例

假设有个类通过dll导出给客户使用，

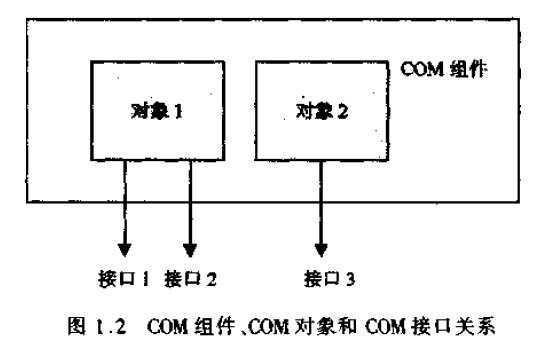
class CDB;

**首先**，它的成员一览无余！

**其次**，类改变了数据成员，或者成员大小，虽然你给了dll拿去替换，但是.h没有进入exe中，会造成崩溃！

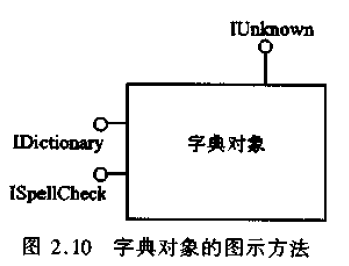
#### 基础概念

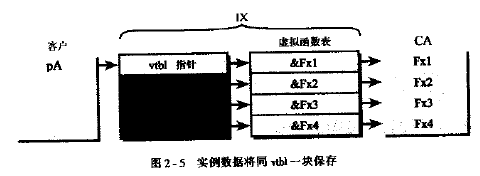
**将复杂的应用程序设计成一些小的、功能单一的组件模块，组件之间可以 跨进程、跨机器、跨语言、甚至跨操作平台 进行通信；**



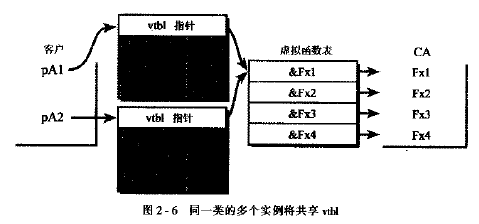
组件由接口构成，接口是一系列函数的集合，客户实际是与COM对象交互，COM对象就是继承接口，有属性接口的类实例，但是客户只知道接口；

CLSID，COM对象的标识；

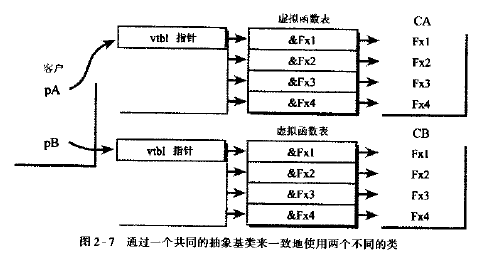




客户创建组件并拿到接口（其实是创建COM对象），实例数据同vtbl指针一块保存，

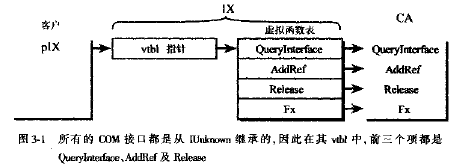


pA1和pA2是同一个类（组件），

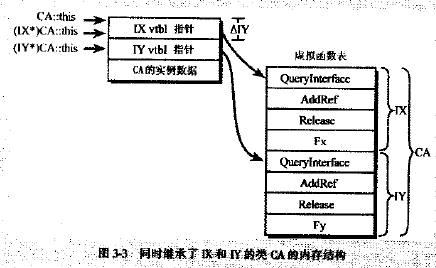


pA和pB不是同一个组件，但是都有同一个接口，

#### IUnknown



所有COM接口都继承自IUnknown，



CA::this和IX\*是同一个，实现IUnknown的三个接口只实现一次，

**最简单的QueryInterface实现**：

HRESULT \_\_stdcall CA::QueryInterface(const IID& iid, void\*\* ppv)

{

if (iid == IID\_IUnknown){

\*ppv = static\_cast<IX\*>(this) ; //组件的IUnknown就是内存结构的首地址

}

else if (iid == IID\_IX){

\*ppv = static\_cast<IX\*>(this) ;

}

else if (iid == IID\_IY){

\*ppv = static\_cast<IY\*>(this) ;

}

else{

\*ppv = NULL ;

return E\_NOINTERFACE ;

}

reinterpret\_cast<IUnknown\*>(\*ppv)->AddRef() ; //引用计数

return S\_OK ;

}

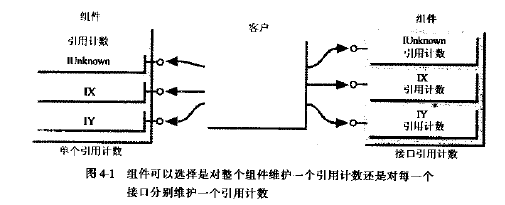
IX\* pIX = NULL ;

hr = pIUnknown->QueryInterface(IID\_IX, (void\*\*)&pIX) ; //引用计数非常合理，确实有人在引用

IY\* pIYfromIX = NULL ;

hr = pIX->QueryInterface(IID\_IY, (void\*\*)&pIYfromIX) ;

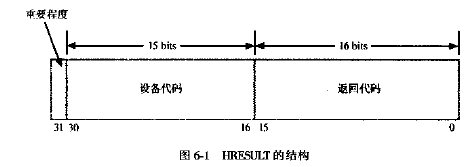
从pIX请求IID\_IY，其实不管哪个指针请求的，都是(IID\_IY\*)(this)，只要pIX是CA\*，



接口的引用计数意味着要调用相应接口指针的AddRef/Release，

好处：1，便于程序调试（找到资源泄漏）；2，资源按需分配释放（不过聚合更优）

#### HRESULT



设备代码标识代表的操作系统部分，保证错误代码不冲突，而自定义代码就不保证了，

winerror.h包含所有错误码（包括COM及OLE等），如果HRESULT返回具有FACILITY\_WIN32设备代码的值，那你可能找不到它的记录，因为它是win32错误代码映射成HRESULT的，如0x80070103，找它的低16位即可，

FormatMessage输出文本描述，

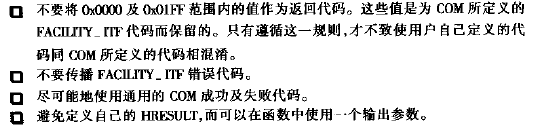
HRESULT注意：

成功和失败返回可能有多个，不要hr==S\_OK，要SECCEEDED(hr)，

失败代码可能会变化，如将进程内组件升级为远程，会有网络错误，

自定义HRESULT

设置FACILITY\_ITF，表示与接口相关的，



MAKE\_HRESULT(SEVERITY\_ERROR, FACILITY\_ITF, 100);

#### 注册表

COM只使用HKEY\_CLASSES\_ROOT，

其下有：

CLSID

InprocServer32

ProgID

(ProgID)SAPI.SpPhraseBuilder.1

ProgID一般格式：<Program>.<Component>.<Version>

与版本无关的ProgID、与CLSID的转换，

**自注册**

dll导出 DllRegisterServer 和 DllUn...，

regsvr32.exe、安装程序等都调用它们，

使用：

对于dll组件，

regsvr32 组件路径 -》调用DllRegisterServer

Regsvr32 /u 组件路径

对于exe组件，程序必须支持/RegServer和/UnregServer命令行参数，大小写无关，’/’和’-‘一个意思，

MessageBox(0, L"hello", 0, MB\_OK); 可以打日志，但是断点进不去；

#### 组件类别

一个接口集合，CATID，方便客户找到所需组件，

实现：windows自带的Component Category Manager，CLSID\_StdComponentCategoriesMgr，

OleView工具，vs2015-》tools-》，第一次运行要以管理员运行，

《Com技术内幕，chap06》，

#### 进程内组件

* 创建组件

CoCreateInstance CLSCTX\_INPROC\_SERVER，创建组件并返回接口；它的**不灵活性**表现在：没有提供一种控制创建过程的方法（工厂，一调用就创建好了），

其封装了CoGetClassObject，

《COM技术内幕-》Chap7》简单的组件注册（不成功进步了断点）和使用CoCreateInstance的实例代码，

为了更灵活，引入类厂

CoGetClassObject(CLSID,IID,PVOID) 和 CoCreateInstance 类似，只是前者会创建组件的工厂和IClassFactory，IClassFactory2（具有安全性）；实际上后者用前者实现，想想就知道了，

灵活性：1）用其他IClassFactory；2）创建多个组件实例，只要创建类厂一次；3）创建的前后提供更多的控制；

**CoGetClassObject伪代码**：

if 进程内组件

调用dll的DllGetClassObject

类厂-》类厂创建组件，

else if 进程外组件

启动组件进程，带上”/Embedding”命令行参数，函数等待，直到组件将类厂注册到COM库，

-》返回类厂代理-》创建组件，

CoCreateInstanceEx，返回多接口，

#### 包容与聚合

##### 包容

再实现一个接口，调用另一个组件的接口，前后还可以加些控制，《技术内幕-》P144，chap8实例代码》，

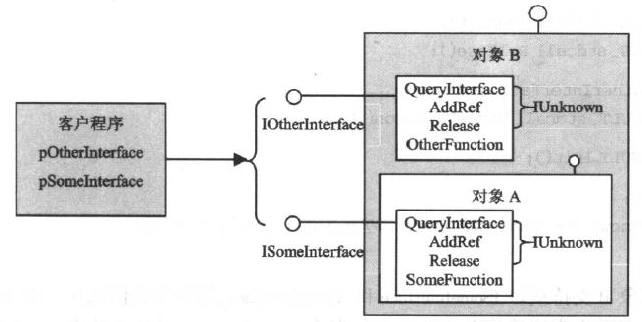
对象的生存期控制：构造时创建包容对象，优化-》需要时创建，

##### 聚合

看一个问题先：外部组件有IX，IY，内部组件有IY，IZ，如果客户查询IY接口时，只是 return m\_pUnknownInner->QueryInterface(iid,ppv);当通过IY查询IX时，就会失败，有一条规则是：QI返回的总是同一个IUnknown\*，不然接口无法互相获取；

**聚合的实现**

组件不用实现内部组件的接口，客户通过外部组件的QI同样可以查询内部组件接口：else if(iid == IID\_SomeInterface){ return m\_pUnknownInner->QueryInterface(iid,ppv); }



客户请求得到接口指针，但再QI->IUnknown确实不一样的，并且从ISomeInterface请求不到IOtherInterface，

CoCreateInstance等API有一个IUnknown\* 指向外部组件，非空即表示组件被聚合，

对于支持聚合的组件，要实现两个IUnknown，代理和非代理（委托和非委托），直接使用的IUnknown为非委托，委托在不同情况下有不同表现：1）正常使用，把调用传给非委托IUnknown；2）聚合使用，把调用传给外部IUnknown；

代码：

1. NondelegatingQueryInterface(...) 请求IID\_IUnknown时，转成INondelegatingUnknown\*，因为IUnknown只能有一个，且是外部的，
2. 委托IUnknown：

if m\_pUnknownOuter != 0

return m\_pUnknownOuter->QI / AddRef / Release。。。

else return NondelegatingQI / AddRef /Release。。。

1. 类厂创建外部组件时，Init初始化创建内部组件（优化：需要时创建）并将外部组件作为参数传入，并请求内部组件的IUnknown接口，
2. 聚合组件的构造，
3. 引用计数

《COM原理与应用》P108，chap4，

P139

#### 进程外组件

* 进程间通信方法：动态数据交换（DDE）、命名管道、共享内存等，COM使用本地过程调用（LPC），它基于RPC的一种“同一机器上进程间通信”的方法，RPC被DCOM用于跨网络进程通信，
* LPC由系统实现，因为系统知道进程的逻辑地址及对应的物理地址，
* 大部分win32 API调用都用到LPC：调API-》客户与一个模仿组件的dll（代理）通信，调dll函数-》dll函数通过LPC调实际代码，因为跨进程，故系统代码非常安全，
* 组件1 、 代理dll（同组件2行为相同） –》 代理dll能访问组件1进程空间，

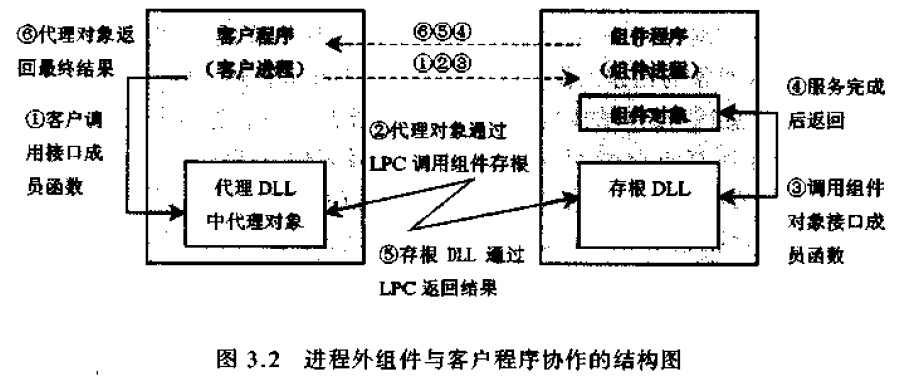
组件2 、 残根dll -》 残根dll能访问组件2进程空间，

所以除了CoGetClassObject需要额外参数外，客户调用进程外组件和进程内组件没什么区别；

* 对跨越进程边界的接口，我们需要考虑的东西有：

1. 一个进程能调用另一个进程的函数；2，传递数据；3，客户用相同方法调用；

将函数参数传到另一个进程的地址空间中的方法叫做“**调整**”，IMarahal，即列集和散集；



##### IDL/MIDL

**功能1**：接口描述语言提供了一种不依赖任何语言的接口描述方法，因此可成为组件和客户之间的共同语言；

**功能2**：IDL对接口和组件描述之后，用MIDL编译即可得到相应的代理和残根dll（主要目的），所以很多地方都和“函数参数调整”有关，

语法

[

object, //COM接口

uuid(32bb..-b41b-...),

helpstring(“IX Interface”),

pointer\_default(unique)

]

interface IX : IUnknown

{

HRESULT Func([in, string] wchar\_t\* szIn);

//...

};

**学习标准COM接口的IDL定义是学习IDL编写的极好方法**

* pointer\_default

告诉编译器，在没有为指针指定其他属性时如何处理此指针，

ref 指针当成引用，总是指向合法地址，函数内部不能指定别名，

unique 指针可为空，不能别名，

ptr 指针就是一个c指针，

* in, out, [in,out] 输入输出参数
* string 字符串，MIDL查找末尾\0决定长度
* #import 同include，但不用担心重复包含问题，
* size\_is数组元素个数

HRESULT Fun([in] long sizeIn, [in, size\_is(sizeIn)] long arr[]);

sizeIn必须是[in]或者[in,out]，尽量避免使用[in,out]参数，

* 结构

可定义c++结构作参数，如 struct Point3d { double x,y,z; };

[out] Point3d\* pt

MIDL需要精确知道指针所指的内容，所以不要使用void\*作参数，

* iid\_is 告诉MIDL相应接口的IID为何

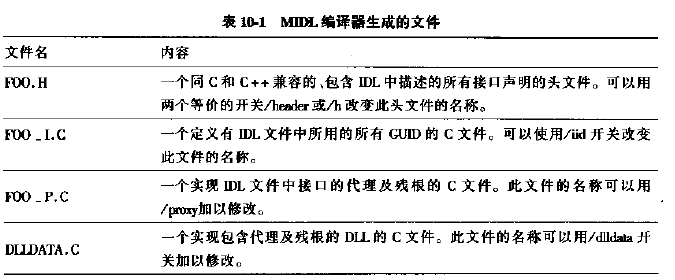
若要传递一个一般性的接口指针，可以使用IUnkown\* + IID，如：

HRESULT GetIface( [in] const IID& iid, [out, iid\_is(iid)]IUnknown\*\* ppI);

**编译IDL**

midl foo.idl

如果idl中有library语句时，MIDL将生成类型库，



P225，makefile使用上面文件生成dll和exe组件的过程，

P226，生成并登记代理残根dll，

**库定义**

库定义可以把一组数据类型（interface，coclass，typedef等）组织到一个空间中，它们都将被符号化，放在类型库中；

一个IDL文件最多一个library语句;

import “xx.idl”; 导入其它声明

[

uuid(...) 库GUID， version, lcid地域, helpstring库名。。

]

library XX

{

importlib(“xx.tlb”); 导入类型库

[[uuid(...)]

coclass XX{ ... }

//其它

}

##### 列集

##### 安全性机制

windows平台上，COM使用windows NT的鉴定服务机制，NT系统上所有的资源都是受保护的，如文件、外设、进程、同步对象、注册表。。。受保护指的是它们与特定的访问权限关联，访问它们时，系统要验证权限，以便允许或禁止；

《VC++COM》，P108；

#### 多线程

#### 异常

COM提供api，SetErrorInfo 抛出， GetErrorInfo 捕获；

IErrorInfo， CreateErrorInfo， ICreateErrorInfo，MSDN；

抛出的异常，直到被捕获之前，COM库都会保存它的一份引用；

代码示例： P74；

把COM异常映射到c++异常，P77；

#### 枚举器

class IEnum<T> : public IUnknown{

Next(...);

Skip(...);

Reset();

Clone(...);

};

是一个COM类，有具体的实现，如IEnumString，IEnumUnknown。。。

使用：

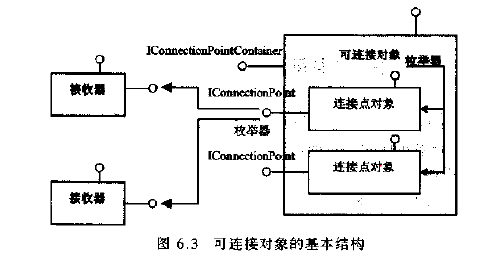
1. 获取枚举器；2）while遍历；3）释放；

#### 可连接对象（源对象）

##### 概念

客户请求的接口，这是**入接口**，这种单向通信功能单薄；**组件主动与客户通信，这是出接口；**

出接口由客户实现，实现的对象叫接收器**sink**；



源对象实现IConnectionPointContainer{ 枚举/获取连接点对象 }，获取不支持的连接点返回CONNECT\_E\_NOCONNECTION，

连接点枚举器：IEnumConnectionPoints{ Next, Skip, Reset, Clone }

连接点：IConnectionPoint { 获取IID，获取容器，建立/取消连接，枚举连接 }

连接枚举器：IEnumConnections{ CONNECTDATA{ IUnknown\*, DWORD}，4常客 }

**建立连接的过程**

1. 客户创建对象，QI(IID\_IConnectionPointContainer...)，询问是否支持出接口
2. pConnectionPointContainer->查找(IID\_出接口,...)是否有出接口，pConnectionPointContainer->Release
3. pConnectPoint->Advise(接收器sink, &dwCookie)建立sink与源对象的连接
4. 取消连接，释放对象。。。

COM已经提供了IDispatch的**代理和存根**对象，所以它可以直接用于进程外源对象的出接口；

##### 实现

**用IDispatch作为出接口**，为什么？

1. 通常组件和客户是独立开发的，客户编译的时候并不知道对象支持哪些接口
2. IDispatch可以用名字访问属性和方法，支持迟邦定（不用编译期）

Invoke(

\_In\_ DISPID dispIdMember,分发ID

\_In\_ REFIID riid,

\_In\_ LCID lcid,

\_In\_ WORD wFlags,属性还是方法

\_In\_ DISPPARAMS \*pDispParams,参数信息

\_Out\_opt\_ VARIANT \*pVarResult,返回信息

\_Out\_opt\_ EXCEPINFO \*pExcepInfo,

\_Out\_opt\_ UINT \*puArgErr) = 0;

**实现出接口**

从IDispatch派生，把属性方法加入，并赋予分发ID；

源对象暴露出接口的类型信息，如类型库.tlb，或IProvideClassInfo接口；

源对象激发事件或请求，传入正确的入参和出参，调用Invoke即可；

**事件映射表**

如果编译时就知道客户应该响应哪些事件或请求，则可以建立一张表，把DISPID和接口方法放入表中，即“事件映射表”；

脚本语言的引擎用动态绑定的；

#### 自动化

##### 概念

###### 基础

实现IDispatch就是自动化对象；

COM库及COM API使用的一些数据类型在其他语言中很难表达，如结构体指针函数指针等，所以要想在其他语言比如VB中编写或使用COM对象常常非常困难，其次VBA（VB for Application）已经成为Microsoft程序如office的扩展，而VBA通过自动化访问程序的数据，所以程序要实现IDispatch，方便暴露内部数据和功能供客户访问；

###### 类型库 & ODL

组件对象的类型信息有：CLSID、接口IID、接口成员函数、函数参数、返回值类型。。。，所有与外界交互的必要信息，客户知道了这些才能使用组件；

ODL是IDL的扩充，用来描述类型信息的；

接口描述同IDL，自定义数据类型可以用C，一个ODL可描述多个组件对象（通常放在一个ODL）；

经过MIDL或MkTypLib工具可编译得类型库；

《COM原理与应用》，P336，Point组件的ODL描述

类型库干嘛用的？

1. VBA使用它浏览组件对象的属性和方法
2. VS类向导读入可生成C++代码供客户使用

##### IDispatch

###### Invoke

牺牲速度与编译时类型检查，带来灵活性，对于进程内组件，vtbl接口比调度接口快100倍；C++开发用vtbl好，IDispatch只是更通用，VB和Java等都能用；

Invoke(

DISPID dispIdMember,分发ID，-7-0有特殊意义，

REFIID riid, 必须IID\_NULL

LCID lcid,本地化标识

WORD wFlags,属性还是方法 DISPATCH\_METHOD

DISPPARAMS \*pDispParams,参数信息

VARIANT \*pVarResult,返回信息

EXCEPINFO \*pExcepInfo,异常信息

UINT \*puArgErr) = 0; 产生错误的参数索引

由于自动化对象提供DISPID机制，所以**动态绑定属性和方法 并进行参数类型检查**成为可能；

###### VARIANT

自动化的方法和属性必须使用VARIANT类型（VARIANTARG等价），虽然没c++好用，但它考虑了其他弱语言；

CY 货币类型、DATE日期（double整数1899/12/30至今的天数，小数为时间值）

BSTR 4B（字符长度）LPTSTR，SysAllocString系列API专门处理BSTR

SAFEARRAY 各种数组，SafeArrayAllocData系列20左右API

{

USHORT cDims;维度

USHORT fFeatures;标识

ULONG cbElements;每个元素大小

ULONG cLocks;锁个数

PVOID pvData;数据

SAFEARRAYBOUND rgsabound[ 1 ];每一维度的边界

} SAFEARRAY;

###### 参数类型转换

VariantChangeType(Ex)、Var<type>From<type>、DispGetParam

###### DISPPARAMS

{

VARIANTARG \*rgvarg;参数数组类型

DISPID \*rgdispidNamedArgs;命名参数的ID

UINT cArgs;参数个数

UINT cNamedArgs;命名参数个数

} DISPPARAMS;

对于属性，一般rgvarg就一个，方法0-n个；如果vt没有指定VT\_BYREF，那就是只读的入参，否则是出参；

**rgvarg参数顺序**与客户调用相反：

Object.Method(arg1,arg2,arg3) –》rgvarg[0] = arg3...

**可选参数**

ODL用(optional)标记可选参数，它在DISPPARAMS中也会出现，Invoke检查vt是否为VT\_ERROR ， scode 是否为DISP\_E\_PARAMNOTFOUND判断可选参数；

**命名参数**

不受次序约束，和定位参数不能交叉，对于客户调用，必须先指定定位参数了；

假设：BSTR CompareWord（int no，BSTR Word1，BSTR Word2，BSTR Word3）;

Obj. CompareWord（3，Word1=”dog”，Word2=”pig”，Word3=”elephant”）;

**rgdispidNamedArgs**

Invoke如何决定命名参数的参数顺序呢？

一旦函数声明了，类型库中每个方法的参数都有参数id，从0开始，如CompareWord，Word1参数id=0...，然后rgdispidNamedArgs = 0,1,2；

**特例**

Invoke执行属性设置时，DISPPARAM = { cArgs = 1,cNamedArgs = 1, rgdispidNamedArgs[0] = DISPID\_PROPERTYPUT, rgvarg[0] = 属性值 }

###### IDispatchEx

IDispatch描述了静态的类型信息，对于一些脚本语言，我们需要动态提供类型信息，如VBScript、JavaScript；

##### 实现自动化对象

客户是方便了，但实现者有一些工作要做；

###### 提供类型库支持

编写ODL，《COM原理与应用》P346，有一个；

编译生成.h和.tlb，如：MIDL /h <header>.h /tlb <library>.tlb <file>.odl –》VS会生成在“中间目录”里；

实现IDispatch的GetTypeInfoCount和GetTypeInfo，《COM原理与应用》P347，大致就是：加载.tlb到内存-》通过GUID获取ITypeLib、ITypeInfo；

###### 实现Invoke

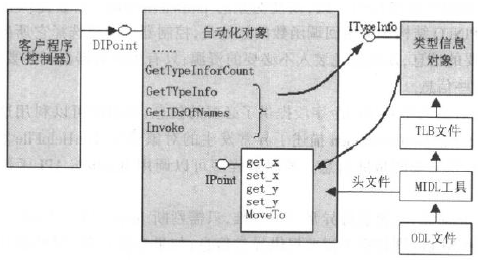
实现GetIDsOfNames

if/switch或者表映射，或者用ITypeInfo::GetIDsOfNames，P350；

实现Invoke

P346把接口分为IPoint和DIPoit是为了分离“业务逻辑”和“IDispatch实现”，P352的伪代码有助于理解；

客户使用自动化对象的流程：



**用CreateStdDispatch实现自动化对象**

上面的方法是基于“tlb”类型信息的，而OLE是提供了一个标准IDispatch实现的，用CreateStdDispatch获取；

P356，大致流程：1）为我们的自动化对象添加Init，方便类厂调用，Init使用CreateStdDispatch获取IDispatch接口；2）为对象的QI添加IID\_IDispatch；

###### 异常

Invoke 出参，如果Invoke异常，会填充它；

对象要么填充该结构，要么交给客户提供的回调，回调的好处就是可以“只关心感兴趣的”，避免不必要的资源；

typedef struct tagEXCEPINFO {

WORD wCode; 错误码=(wCode==0?scode:wCode)

WORD wReserved;

BSTR bstrSource;错误源ProgID

BSTR bstrDescription;错误描述

BSTR bstrHelpFile;帮助文件路径 **WinHelp API**

DWORD dwHelpContext;帮助ContextID

PVOID pvReserved;

HRESULT (\_\_stdcall \*pfnDeferredFillIn)(struct tagEXCEPINFO \*);

SCODE scode;

} EXCEPINFO

Invoke通过类型库实现的话，P354，1）中间变量；2）OLE接口ICreateErrorInfo和IErrorInfo；

###### 本地化

LCID，影响：1）字符串资源的表达、比较等；2）日期时间表达；3）货币格式；

每种语言的LCID与它的语言ID、子语言ID、排序ID及版本号有关，winnt.h列出了语言ID及LCID位结构说明；

##### 自动化对象应用

###### 双接口

客户通过IDispatch::Invoke能访问对象的所有方法和属性，但是Invoke内部有层层处理，如参数检查、类型信息处理等，效率损失很大，对于VB，Java等语言没办法，而对于C++的客户，他们是希望直接通过vtable调用对象成员的，因此有了“双接口”；

P358有一份参考；

###### 自动化集合对象

一组同类对象（或数值）的容器对象，P360；

###### 以IDispatch作出接口

ODL在coclass给出接口加上source属性；

大致过程：1）自动化对象调用IDispatch::Invoke激发事件；2）sink实现IDispatch::Invoke处理各种事件；

###### C++包装类

一个或多个COM接口的封装，简化COM接口的调用，类似MFC包装Win32 API，实际就是组件包容，

VS提供了可以为ActiveX及其他COM组件自动生成包装类的工具，

在用向导“添加类-》from tlb”之后，生成的头文件就是“包装类”的声明，它们都有一句：#import "C:\\Program Files (x86)\\Microsoft Office\\Office14\\EXCEL.EXE" no\_namespace

它们会 vc++编译器解析生成.tlh，.tli也就是c++标准类型，（因为odl和tlb并不是c++标准），对于c++程序，我们可以把它注释掉，因为c++能理解；

##### 自动化编程

###### 实例：PointComp.sln；

新建MFC DLL工程-》类向导，添加MFC Class

如果我们的自动化对象要暴露到外面并能被类厂创建的话-》“Create by type ID即ProgID”；

CCmdTarget实现的IDispatch没有用到类型库，而是“分发映射表”，因此不支持多语种；

###### 实例：MFCSampleExe.sln;

向导-》基于对话框-》除了Automation，其他都不勾-》删除主dlg和DlgProxy文件、IDD\_资源、代码dlg相关的，如#include<dlg.h>、idl文件的IExe声明；

Per-User Rediection Link->Gernel，再编译一次就没了；

###### 添加属性和方法

打开Class View，右键类对象，不行，VC6.0有Automation选项，后续版本不知道在哪里；

手动添加方法

1，.h声明；2，.cpp实现；3，DISP\_FUNCTION；4，idl添加方法及dispid；

手动添加属性

1，.h，Get/Set声明；2，.cpp实现；3，DISP\_PROPERTY\_EX或DISP\_PROPERTY\_PARAM；idl添加属性及dispid；

###### 实例：MFCCALC.sln

《COM原理与应用》，ch11

自动化组件 及其客户，整个逻辑非常清晰；

##### COleDispatchDriver

避免复杂的Invoke调用，方便客户使用；

两种用法：

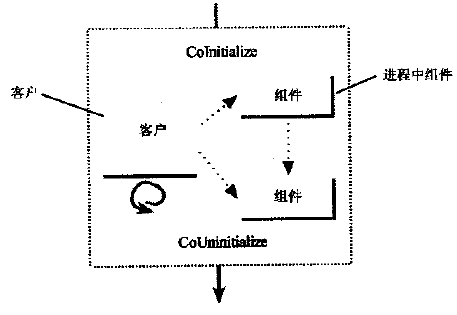
1. 向导-》Add Class-》From a type library，选择.tlb文件，选择接口-》自动生成继承自 COleDispatchDriver的自动化接口；
2. 如果我们已经获取到一个IDispatch接口，那用AttachDispatch与COleDispatchDriver联系，调用InvokeHelper、Set/GetProperty；

#### 多线程

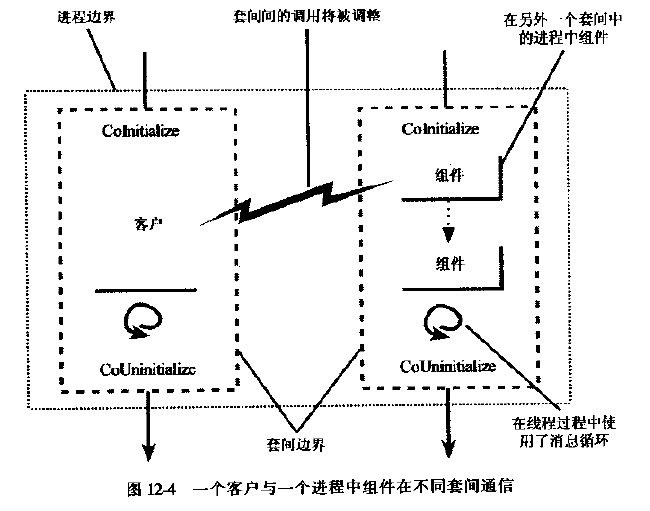
窗口被创建线程所有，所以窗口过程按消息顺序执行，不用同步；

COM使用了win32线程模型：用户界面线程和工作线程，对应“套间线程”和“自由线程”，一个进程的用户界面线程可以有多个，一个用户界面线程关联着n个窗口，COM类似，区别在于：参数调整与线程同步；

一个套间是一个由用户界面风格的线程（套间线程）和一个消息循环构成的，它像“装饰模式”一样套在外面，圈定范围；



对于客户在进程内使用的组件，套间线程即被圈定在进程边界，使用COM库之前和之后调用Co(Un)Initialize即可，



一个套间只有一个套间线程，像用户界面拥有创建窗口一样，套间拥有组件；像消息一样，一个线程调用组件方法时，COM把调用请求放到组件的套间线程队列中执行，所以不用考虑同步问题，

自由线程创建的组件可以被任意线程访问，同步要组件自己控制，

P274

### 对编程有实际指导作用

#### 概念

#### 创建组件工程

##### 简易数据库

接口文件一般提取到一个目录下，因为客户和组件都要包含；

因为不能用\_declspec(dllexport)修饰DllGetClassObject等——在Objbase.h中已经使用了其他修饰，所以.def 要在工程属性-》链接中添加，否则没有.lib生成；

《visual c++》，P67，chap2,，DB组件，DBComp.sln，

新建dll工程，把文件拖进去稍加修改，组件类，组件工厂类，接口.h，

新建客户工程，main函数里一套流程：创建组件，使用，释放。。。

注意GUIDS.CPP的定义，INITGUID，DEFINE\_GUID，公共文件提取出来，

注：如果创建win32 DLL工程，遇到编译时dllmain重定义，可以把\_USRDLL预定义宏去掉；

extern “C” 没有写；

##### 字典组件

《com原理与应用》，P87，chap3，流程介绍，

1）将接口声明提取到公共部分，如interface\...

2）GUID声明使用DEFINE\_GUID

GUID一般放在组件主.cpp 和 客户要使用的地方定义，如：

// {54BF6567-1007-11D1-B0AA-444553540000}

extern "C" const GUID CLSID\_Dictionary = //客户那要定义，用到就要给定义，声明

{ 0x54bf6567, 0x1007, 0x11d1,

{ 0xb0, 0xaa, 0x44, 0x45, 0x53, 0x54, 0x00, 0x00} } ;

在接口.h中声明，

// {54BF6569-1007-11D1-B0AA-444553540000}

extern "C" const GUID IID\_SpellCheck;

在组件.sln中添加GUID.CPP，#define INITGUID，#include “接口.h”...

这样，只要在接口.h中使用一次DEFINE\_GUID宏，在组件这里就是定义接口，而在客户那里，就是声明；

字典组件DictComp.sln

1. 接口.h，类厂.h，注册表.h，COM组件对象.h，其它GUIDS.CPP，.DEF，。。。
2. 调试组件注册过程，组件工程属性，debugging，command + Arguments（-u $(SolutionDir)$(Configuration)\$(TargetFileName)），regedit程序不能开着，否则DllRegister失败！ 但是用powershell命令行调用失败（其实是DictComp.dll，但我写了DBComp.dll，这个dll还没写好呢！）； 一定要用管理员权限注册，否则注册表操作函数调用失败；

调试组件

在debugging里输入客户程序完整路径名；

客户程序

1），接口.h，GUIDS.CPP，。。。

2），使用组件：CoGetClassObject、CoCreateInstance的伪代码，

进程内：COM库优化结构-》CoLoadLibrary-》DllGetClassObject-》。。。

进程外：。。。

##### CoMath组件

MathComp.sln，详见笔记《使用向导》；

#### 在MFC中使用COM

##### 概述

* 为什么MFC使用嵌套类？

1. MFC尽量避免多继承
2. 遇到一些问题：1）接口函数重名，很难实现它们；2）继承一个接口实现有时不是想要的
3. 效率问题：1）vtable、pVtl必须开销；2）编译器为多继承生成隐藏代码；

##### 头文件

<unknwn.h> IUnknown、IClassFactory...

<Wtypes.h> COM数据类型...

<objidl.h> 标准接口idl...

<comdef.h> 标准接口及COM和OLE内部对象的CLSID...

<objbase.h> COM API...

<ole2.h> ole辅助API...

##### 宏

DECLARE\_INTERFACE(iface) 声明接口，它不从其他接口继承

DECLARE\_INTERFACE\_(iface,baseiface) 从baseiface继承，

STDMETHOD(method)声明接口方法method，返回值HRESULT，

STDMETHOD(type, method)返回值type，

##### 接口映射表

MFC使用嵌套类实现组件，在DictComp.sln中，我们可以通过宏一步步简化，这都是CCmdTarget类提供，《COM原理与应用》，P153；

1）**接口映射表定义**

DECLARE\_INTERFACE\_MAP() 类似消息映射表

定义“接口列表”, “接口表（基类+本类）”

“接口列表”元素 = { IID, 与this的偏移 }

BEGIN\_INTERFACE\_MAP(CDictionary组件对象类,基类CCmdTarget)

INTERFACE\_PART(CDictionary组件对象类,IID\_IDictionary,嵌套类名Dictionary)

嵌套类：

class XDictionary : public IDictionary{

//方法

}n)m\_dictionaryObj;

END\_INTERFACE\_MAP()

2）**嵌套类定义**

BEGIN\_INTERFACE\_PART(嵌套类名Dictionary,接口名IDictionary)

INIT\_INTERFACE\_PART(CDictionary组件对象类, 嵌套类名Dictionary) //QI...

STDMETHOD\_...

END\_INTEFFACE\_PART\_STATIC(嵌套类名Dictionary)

##### 聚合

CCmdTarget实现IUnknown，包括内部和外部IUnknown，<afxwin.h>，

使用COM聚合：

1. 声明聚合变量IUnknown\*
2. 在接口映射表中加入INTERFACE\_AGGREGATE宏
3. 在CCmdTarget::OnCreateAggregates()初始化成员变量

《MFC internal》P368，聚合实现，

##### COM引出函数与类厂

**引出函数**

如果在AppWizard中勾选了“Automation”，则自动生成DllGetClassObject，DllCanUnloadNow，DllRegisterServer，

DllUnRegisterServer要自己实现，

**类厂**

**COleObjectFactory** ： public CCmdTarget，MFC的通用类厂，实现了IClassFactory2；

AfxOleRegisterServerClass注册单个COM对象；COleObjectFactory::Register注册进程外组件类厂到全局对象表；

《MFC internal》，P381;

**小技巧**：用AfxOleRegisterHelper更新注册表；

3）为组件添加内嵌类厂

DECLARE\_OLECREATE()

IMPLEMENT\_OLECREATE()

DllGetClassObject如何找到类厂的？

在IMPLEMENT\_OLECREATE中，COleObjectFactory的构造函数将类厂加入到“全局状态结构”的类厂表中（AFX\_MODULE\_STATE），然后根据GUID找到类厂；

oledll.cpp 有AfxDllGetClassObject实现；

##### 使用向导

###### 创建工程

MFC dll-》Automation；

如果要创建进程外COM，MFC exe-》基于对话框，把对话框部分删掉即可作为基础框架；

###### 创建对象类

project-》向导，添加MFC Class，继承CCmdTarget，Automation支持自动化功能，

这样就创建了一个COM组件下对象类了，如CoMath；

class CoMath : public CCmdTarget

{

DECLARE\_DYNAMIC(CoMath)

public:

CoMath();

virtual ~CoMath();

protected:

DECLARE\_MESSAGE\_MAP()

//定义嵌套类

BEGIN\_INTERFACE\_PART(MathObj, IMath)

virtual HRESULT Add(int, int, long&);

virtual HRESULT Subtract(int, int, int&);

END\_INTERFACE\_PART(MathObj)

//声明接口映射表

/\*接口映射表如何工作？

CCmdTarget::InternalQueryInterface()调用GetInterface，《MFC internal》P377

\*/

DECLARE\_INTERFACE\_MAP()

//声明通用类厂，CoMath是组件对象

DECLARE\_OLECREATE(CoMath)

};

###### 完成服务器

1. 实现接口；2）分配GUID；3）提供类厂；
2. METHOD\_PROLOGUE() 将嵌套类的QI，Add，Release委托给外部类

STDMETHODIMP\_(ULONG)

CoMath::XMathObj::AddRef() {

METHOD\_PROLOGUE(CoMath, MathObj);

return pThis->ExternalAddRef();

}

MFC还没提供这部分的向导，可能还要麻烦一点地写它们；

###### 问题

Pre-Use Redirction， 工程配置-》link-》Pre-Use。。。改为Yes；

#### 智能接口指针

ATL的CComPtr和CComQIPtr，MFC的CIP类，

#### COM库的内存管理

《COM原理与应用》P74，

#### 组件的卸载

《COM原理与应用》P78，

进程内

卸载条件：1）组件中对象数为0；2）类厂锁计数为0；

则DllCanUnloadNow返回true，

客户在空闲时调用，CoFreeUnusedLibraries，检查，

进程外

P78

#### 调试COM

##### 调试ActiveX

html博客；

### COM核心编程

#### 激活

##### 简要

**客户找到类对象（类厂）的机制；**

COM有三种激活模型：1，绑定到给定类的类对象，2，根据CLSID创建一个新实例，3，从文件激活

##### SCM

COM服务控制管理器，每台机器都有，用于激活对象和绑定接口指针；

1）COM服务可以在你自己的进程内激活，这样就是映射dll代码了；

2）第三方不给你提供dll，但有exe，那就是进程外COM激活了；

3）别人只提供服务，不给你组件，那就是远程COM激活了；

SCM ：RPCSS.EXE + ORPC 本地/远程过程调用通信协议；

ROT（running object table）运行对象表，SCM的一个设施；

##### 如何激活对象

**请求SCM把一个指针绑定到客户请求的类对象上：**

**CoGetClassObject**(

REFCLSID, //哪个类的类对象

DWORD dwClsCtx,//在哪里激活

COSERVERINFO\*//主机/安全信息

REFIID,

void\*\*);

**dwClsCtx**是一个位掩码，

enum CLSCTX { CLSCTX\_INPROC\_SERVER进程内，LOCAL进程外，REMOTE远程}

多个标志组合，COM会选择最有效率的类型，SDK提供了组合宏：

CLSCTX\_INPROC, CLSCTX\_SERVER, CLSCTX\_ALL

**主机/安全信息**

客户显示指定在哪台机器上激活对象，以及配置请求的安全属性；

struct COSERVERINFO{

DWORD = 0,

LPWSTR 主机名，

COAUTHINFO\* 安全设定，

DWORD = 0

};

**查找接口指针的过程**

注册表在HKCU\Software\Classes\CLSID和HKCR\CLSID下将CLSID映射到实现文件名或远程主机名，找到-》加载dll，没找到-》把请求重定向到远程，然后下载dll到本地自注册；

**自注册**

导出 DllRegisterServer 和 DllUnRegisterServer，注册表下载远程服务，安装程序等会调用；

实现， P92；

加载dll之后，dl要导出一个函数，获取COM的接口指针：

DllGetClassObject(REFCLSID, REFIID, void\*\*);//从dll获取类对象

**CoGetClassObject**的实现伪码：P95；

大致流程：

if(进程内) –》LookupInClassTable进程内COM服务表 –》GetFileFromClassStoreOrRegistry在COM库或注册表中找dll名-》LoadLibrary-》找到DllGetClassObject函数执行，获取接口指针；

一旦获取类对象完成，COM库和SCM不再介入通信过程，如果不是进程内激活，那COM将返回一个代理指针，客户通过它通信；

##### COM库的“对象发现和创建”标准接口

**对象发现**

IOleItemContainer，配合单项名字对象使用；

**对象创建**

IClassFactory –》CreateInstance，LockServer

类对象都要继承的，LockServer进程外激活，CreateInstance创建类的新实例；

class ChimpClass : public IClassFactory{//..};

class Chimp : public Ape{//...};

HRESULT CreateChimp(IApe\* &rpApe)

{

extern const CLSID CLSID\_Chimp;

rpApe = 0;

IClassFactory\* pcf = 0;

HRESULT hr = CoGetClassObject(CLSID\_Chimp, CLSCTX\_All, 0,

IID\_IClassFactory, (void\*\*)&pcf);

if (SUCCEEDED(hr)) {

hr = pcf->CreateInstance(0, IID\_IApe, (void\*\*)&rpApe);

pcf->Release();

}

return hr;

}

上面的创建过程 对于进程外或远程激活，CoGetClassObject，CreateInstance，Release这些操作都要一次来回数据传输，于是api：

**CoCreateInstanceEx**( P99

REFCLSID ,实例化的对象

IUnknown\* 聚合（父对象是谁）

DWORD 哪里激活,

COSERVERINFO\* 主机/安全信息,

ULONG cmqi 接口数目 一个对象 可以有N多接口

MULTI\_QI\* 接口存放位置);

S\_OK, CO\_S\_NOTALLINTERFACES，MULTI\_QI的HRESULT，SERVERITY\_ERROR基础的HRESULT（无法创建对象或接口都不提供）

如果只需要一个接口，api：**CoCreateInstance**更方便，P103；

COM库会缓存类对象，不用SCM，这是最快的；

##### 名字对象

绑定到期望对象的过程， 上面的太麻烦了， COM提供了一种标准机制让**激活对象**更方便灵活，它就是名字对象， 一个支持IMoniker接口的COM对象；

api

运行你的查找或创建对象 的 算法， 返回接口指针；

IMoniker : **BindToObject**(

IBindCtx\*,

IMoniker\*, //复合

REFIID,

void\*\*

)

那么现在的问题就是**如何获取名字对象**了，我们只需要一个字符串，它在注册表中配置，称为显示名，也就是说每一个名字对象都有一个显示名；

**那如何把显示名对应到名字对象上呢**？api：

**MkParseDisplayName**(

IBindCtx\*, //绑定信息

const OLECHAR\* //字符串

ULONG\* //错误过程

IMoniker\*\* //名字对象

)

方法使用解析器检查字符串前缀，匹配注册表信息找到对应的名字对象类型，创建名字对象，有了名字对象，就能绑定到对象了；

api：

**CreateClassMoniker**

创建类名字对象（创建类对象的）；

或者将类名字对象的显示名传给MkParseDisplayName，clsid:571F16... 这个clsid就是名字对象的ProgID；

绑定过程， P107，先定义一个显示名-》MkParseDisplayName找到名字对象-》BindToObject绑定到对象-》调用接口；

api：

**CoGetObject**，简化绑定过程，伪码，P108，

创建绑定环境-》将显示名转换为名字对象-》绑定对象；

**如何绑定到接口？**

上面只是绑定到类名字对象上， 可以创建对象了，绑定到哪个接口呢？

单项名字对象

clsid：571F168-...:!Ursus

类名字对象 与 单项名 “Ursus”，MkParseDisplayName会使用IParseDisplayName: ParseDisplayName 解析剩余的字符串，P110；

**文件名字对象**

api：

**CoGetInstanceFromFile**，P113；

GetClassFile；

CreateFileMoniker；

IPersistFile :: Load；

IRunningObjectTable\* p = GetRunningObjectTable；

##### 服务器生命周期

空闲的时候调用CoFreeUnusedLibraries，它会遍历每个dll，调用导出的DllCanUnloadNow，它的实现规则应该是这样：只要有接口在使用，或者 IClassFactory：：LockServer调用，就不能卸载dll，这个可以实现在AddRef 和 Release里， 每当创建或使用接口，计数加1；

##### 类模仿

指定“老的CLSID已经被新的CLSID替换了”，接口没有变化，只是功能扩展了；

CoTreatAsClass， P122；

##### 组件类别

有时候我们只是需要“某种功能”的类，COM把实现同一组接口的类称为一个类别（category），CATID类别id，每个类的属性，存在注册表中， 一个是HKCR\CLSID\...\Implemented Categories，一个是“Required Categories”，第二个类别是 “站点接口”，将客户提供的服务归类（只有客户提供了服务才能激活对象），对应HKCR\Component Categories；P124；

我们可以使用注册表函数来注册类别，也可以使用COM 组件类别管理器暴露的ICatRegister，ICatInformation接口，类别注册和查询；

书签： P129；

#### 二进制复合

4.8节，多个接口方法冲突的解决办法，复合技术；

4.9节，短暂分配复合技术tearoff，动态按需分配vptr空间；

上面的复合技术是源码级的；

**COM聚合**

直接通过QueryInterfaceI 把一个子组件暴露给客户的技术；

class CarBoat : public IBoat {

IUnknown\* m\_pUnkCar;

//...

};

我们很少直接把另一个对象的接口当作自身的一部分暴露给客户，因为QueryInterface表达了“是一个is-a”的关系，自行车有时候不是一个轮子；

**COM包容**

把一个不可被聚合的类暴露成一个对象实体的一部分，外部对象提供内部对象已经实现的接口；

#### 套间

##### 概念

为了使用对象而不必关心并发，套间定义了一组对象的逻辑组合，它们共享同一组并发性；

线程创建时，没有关联的套间，线程调用：

**CoInitialize**(Ex)(0, DWORD flag COINIT\_MULTITHREADED/ APARTMENTTHREADED)-》线程进入套间（要使用COM必须先进入套间，COM会把套间信息保存在TLS中），CoUnInitialize离开套间；

每个COM对象都属于某个套间，只有运行在对象套间中的线程才能访问该对象；

有2种套间，多线程MTA和单线程STA，进程有一个或多个套间，MTA最多一个，STA一个线程可以执行，意味着对象只有一个线程可以访问它们，不过将来有RTA租用模型；

线程模型

每一个CLSID都有自己独立的线程模型，注册表HKCR\CLSID\..\InprocServer32\ThreadingModel，Both，Free(MTA)，Apartment，无ThreadingModel（在进程的第一个初始化的STA中运行）；

如果客户的套间（代码所在线程) 与CLSID套间（注册表）不兼容，那么对CLSID进程内激活请求将使COM在另一个套间中构造对象，然后给客户返回一个代理；

##### 跨套间访问

###### 简要

为了允许对象驻留在不同套间中；

**引出接口**：从一个套间引出接口，并引入另一个套间

**引入接口**：使套间内部也能看到外部的接口，引入的是代理，这个代理保证把控制传回给对象的套间，这个控制传递过程被称为**方法远程传递，**是COM中跨线程，进程，跨主机通信的基础，基于RPC/ORPC通信协议；

###### 列集

COM使用**列集**（marshal）技术，允许接口指针被跨越套间传递，即把接口指针列集成字节流传输（唯一标识对象和它的套间），然后散集成客户套间的一个代理；

显示列集API：

**CoMarshalInterface** P174，列集到一块内存中，它可以被传输到网络上任意一个套间中，MSHLFLAGS{ 常规列集，表格列集 ...}；

为了使接口指针可被列集到各种介质中，CoMarshalInterface 使用IStream接口作参数，P175例子，列集之后其它套间就能通过CoUnMarshalInterface访问这个接口；

对于在MTA或RTA中执行的线程，传递到另一个套间中去，并不需要散集，P181；

###### 被列集对象的格式;

P178，OXID/OID/IPID组：唯一标识一个接口指针，OXID 对象引出ID，当代理第一次连接到对象上时，OXID可用来找到网络/IPC地址信息，OID 对象ID，唯一标识COM实体，IPID，接口指针ID；

OR，RPCSS，套间。。。

###### 列集辅助函数

P179，简化列集/散集过程；

CoMarshalInterThreadInterfaceInStream

CoGetInterfaceAndReleaseStream

###### 全局接口表（GIT）

常规列集（MSHLFLAGS\_NORMAL）只能散集一次接口指针，很多情况下，多个辅助线程需要散集这个接口，如果辅助线程在MTA中，那没问题，但如果在任意套间中，就用表格列集（可以散集多次），不幸的是，如果接口是代理的话，不支持表格列集，因此出现了GIT；

允许接口指针被进程中所有的套间访问，COM内部为每个进程实现一个GIT，它包含所有列集的接口，可被散集多次；

接口：**IGlobalInterfaceTable** P182；

CLSID\_StdGlobalInterfaceTable

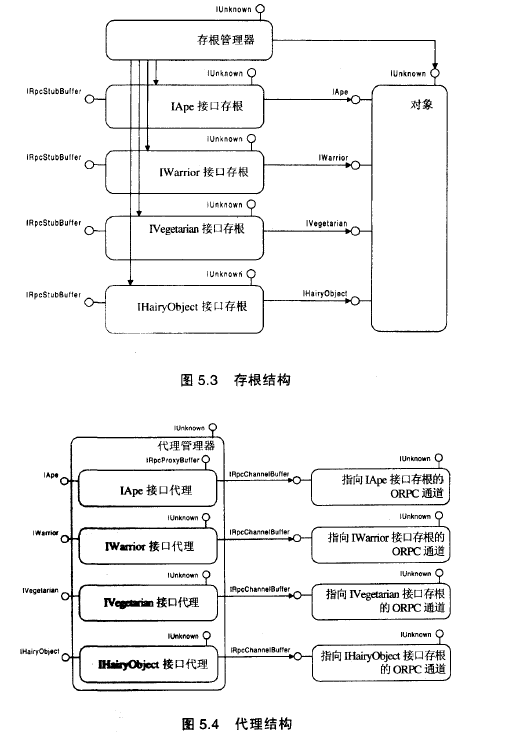
###### 标准列集结构

对象第一次调用CoMarshalInterface时，方法会询问是否使用自己的跨套间通信机制（就是QI(IMarsha)），大多对象没有实现IMarshal，所以使用默认的COM-ORPC；

当对象希望使用标准列集方式时，CoMarshalInterface创建一个特殊的COM对象：**存根管理器**（stub manager），它能唯一标识对象，而且它们是一一对应的，它跟踪记录未完结的外部引用，只要还有引用，资源就存在内存中，类似shared\_ptr；

存根管理器了解“对象身份”，而并不了解通信，它的一个辅助对象：**接口存根**，它知道接口方法的所有细节,P184；

当CoUnMarshalInterface散集一个对象引用时，它返回一个**代理管理器**指针， 也是识别“对象身份”的，它的辅助对象：**接口代理**，处理通信的；



代理与存根之间 是 通过**通道**（channel）的COM对象通信的，通道封装了RPC运行时层，暴露IRpcChannelBuffer，接口存根暴露IRpcStubBuffer，接口代理暴露IRpcProxyBuffer，P187；

接口代理和存根是动态绑定到管理器上的，它们共享CLSID，这个包含两个分支实现的实体称为**接口列集器**（创建接口代理和接口存根的），它的类对象暴露**IPSFactoryBuffer**；

接口与CLSID映射关系：HKCR\Interface 和 HKCU\Software\Classes\Interface，P188；

###### 标准列集、线程和协议

###### 列集与生命周期

###### 自定义列集

有些对象，ORPC提供的缺省行为不是很效率，需要自定义列集方式；

###### 自由线程列集器

#### 应用

##### 进程内激活的缺陷

###### 错误隔离性

COM对象访问违规，或其他运行时错误，客户进程将与对象一起被终止；同样，进程发生错误，或被关闭时，对象也都被销毁（毫无警告）；

###### 安全环境问题

进程内激活的COM对象执行的是 客户的安全令牌，这意味着 特权用户可以做“危险”的事，低权限的用户无法访问“重要的操作”，我们没有办法让进程内对象拥有自己的安全环境；

###### 分布式计算

在进程的地址空间中，就会共享客户的CPU和本地资源，无法提供分布式计算功能，而且也使“多个进程共享同一对象”非常困难；

##### 进程外服务器

HKCR\CLSID\..\LocalServer32，类的代码被包装成exe（进程外服务器），这在安装程序的时候写到注册表中；

不导出DllRegisterServer，而是检查命令行是否有/RegServer 和 /UnregServer；

一个exe只要显示地把自己注册到SCM中，它就变成某个类的服务器进程了，以后对于这个类的进程外激活都将被分发到该exe，api：**CoRegisterClassObject** 注册服务器进程到SCM，P220，COM库会在内部维护表中关联类对象指针（第二个参数）与它的CLSID，局部特征与激活标志参数，P220；

##### 进程外服务器的生命周期

##### 应用id

GUID，被称为AppID，代表一个服务器进程，HKCR\CLSID\..\AppID 和 HKCR\AppID，大多数AppID可以使用DCOMCNFG.EXE设置远程和安全性，OLEVIEW.EXE还提供一个以COM为中心的注册表视图；

AppID的一项设置是 RemoteServerName，如果你没有指定COSERVERINFO的话，P230；

既没有指定主机名，也没有指定标志，SCM处理，P231；

COM支持三种进程模型：NT服务，普通进程，代理进程surrogate process，P232；

##### COM和安全性

###### 简要

认证authentication：确保消息可信；

访问控制access control：允许谁访问服务器对象；

令牌管理token management：可执行哪些操作；

会有默认设置，大部分都可以在注册表中配置，一般系统管理员使用DCOMCNFG.EXE，显示使用api会灵活些；

P234，RPC的可装载模块：传输协议、安全包；

**CoInitializeSecurity**，P236；

###### 自定义安全性

为小部分对象设定，**IClientSecurity**，P240；

###### 访问控制

###### 令牌管理

### VC++实践与提高-COM和COM+篇

#### 命名与绑定

**概念**

另一种创建对象的方法，名字对象（moniker），全面封装了组件对象，客户只需要创建相应的名字对象，它可以代表一个文件，一个单元格，一个数据表，一个COM对象等，它可以使被动的对象进入运行状态；

COM实现了一组名字对象，如，文件名字对象等等；

IMoniker接口，P140；

COM系统名字对象：文件、复合、单项、反-、指针和类 名字对象，以及不断更新的，如URL名字对象，广泛用于整个操作系统，文件复合单项最常用；

文件名字对象：CreateFileMoniker P142；

复合名字对象：不同类型的名字对象组合在一起，就是复合名字对象，有通用和特定组合方式，IMoniker::ComposeWith 实现组合过程，CreateGenericComposite创建通用，P143；

单项名字对象：标识一个包含在其他对象中的对象，容器对象决定名字空间，如：”C:\Data\MyReport.doc ! embedobj1” 标识文件中的embedobj1对象 CreateItemMoniker P144；

反、指针 P144；

类名字对象：更灵活创建类对象 P144；

以上都是同步名字对象（绑定过程中，必须等到所有数据和对象准备就绪后，才返回结果），网络中的是都是异步绑定的，异步实现IMoniker和IAsyncMoniker；

**使用**

1，获得名字对象

MkParseDisplayName通用，CreateFileMoniker特定，其它接口传递，数据传输机制得到（剪贴板或拖放）等

2，绑定

IMoniker::BindToObject 绑定，返回成功则获得对象；

#### 统一数据传输

概念

数据对象 IDataObject接口，使数据交换同传输协议分离，最基本的传输协议有 剪贴板 和OLE拖放；

机制：所有传输协议只需交换一个IDataObject接口指针，一个实现数据对象，一个获取数据，不管使用什么协议；

数据格式和传输介质

FORMATETC 和 STGMEDIUM P147；

IDataObject 接口 P51；

MFC对数据传输的支持，COleDataSource，COleDataObject P152；

剪贴板 传输 P154；

OLE拖放技术 P159；

#### COM应用

##### 导读

实际开发中，很少直接用COM API 和 在COM接口上进行设计，我们直接用COM技术，包括：自动化技术，OLE文档，ActiveX文档，ActiveX控件等；

VC编译器提供了一些COM支持，如

\_bstr\_t，封装了BSTR类型，\_variant\_t 封装了VARIANT类型，\_com\_ptr\_t 封装了COM接口指针，它会自动调用AddRef，Release和QueryInterface；

##### OLE/ActiveX文档

OLE文档包容器 和 OLE文档服务器

包容器是可以将OLE文档服务器以嵌入或链接方式合并的程序，服务器则是 创建OLE文档对象的程序，如word是包容器，word被嵌入就是服务器程序，大多数OLE服务器是独立的程序，叫full-servers，有的只能在包容器中运行，mini-servers，包容器对象不需要知道服务器的实现，只要是OLE组件，遵循COM规范即可；

站点对象：服务器对象通过它与包容器通信，每个OLE对象都有自己的站点对象；

ole技术用ActiveX商标销售，原来的链接、嵌入和现场激活技术转移到ole容器文档中；

嵌入与链接技术

链接，只是保存一个路径，如，双击会打开excel并调入原电子表格文档；

嵌入对象实现IOleObject P297，包容器实现IOleClientSite P299；

现场激活技术in-place

支持在包容器内部编辑， 除必要的OLE接口外，IOleInPlaceObject，还必须提供一个实现了IOleInPlaceActiveObject的对象， P299，站点对象也要实现一些接口 P300， 包容器框架对象 IOleInPlaceFrame；

MFC对OLE的支持， P301；

ActiveX文档，P168， P328；

在OLE文档基础上，可以从web服务器下载的特殊文档类型，浏览器遇到ActiveX文档时，自动从硬盘装入ActiveX文档服务程序（word，excel等），并接管整个浏览器窗口，可编辑，但不能传回服务器；

开发ActiveX文档， P329；

##### ActiveX控件

应用最广泛的COM组件，图形图像处理，数据库。。。能快速建立起功能全面的应用系统；

一个真正有用的ActiveX控件，有：

属性和方法， 事件机制，用户界面特性，永久性，通常支持的接口，P340；

库存属性、方法和事件， P340；

IOleControl，IOleControlSite， P342；

属性页技术， P344

当ActiveX控件处于设计模式时，用属性页修改属性比用接口现实；

MFC开发 ActiveX控件， P345；

ATL开发, P373；

##### Internet与ActiveX控件

安全性 ， 权限，签名 P392， 安全设置， P393； 使用许可， P398；

在ActiveX中访问DOM， P409；

IIS（Internet信息服务器）提供服务器功能， IE浏览器提供客户机功能；

ASP（活动服务器页面），是脚本文件，它在服务器上被分析处理，JavaScript，Perl等开发，当客户机（浏览器）请求ASP页面时，IIS发现是.asp，于是开始分析处理，发送http响应给客户机 P414；

ASP内建对象

不用创建，直接访问，可以获取http请求，存取对话信息，响应http等， P414；

在C++COM中访问ASP内建对象， P415；

##### 企业应用模型

多层软件应用模型：

界面交互，html，vc。。。

业务逻辑，COM，IIS（Internet Information Server），MTS（Microsoft Transaction Server）。。。

数据服务，db。。。

##### 在VC中使用ActiveX控件

创建一个支持ActiveX控件的MFC包容器程序；

通过创建包含控件接口的类 把控件下载到项目中；

确定控件的属性和方法，并设置设计属性；

建立一个代表控件包装类的对象的成员变量；

MFC向导，基于对话框，支持ActiveX；

#include <afxdisp.h> 和 AfxEnableControlContainer，这样就创建了一个“支持ActiveX控件的MFC包容器程序”;

##### 在VC中使用代码组件

三种方法：

1，直接用CoCreateInstance访问COM对象的方法和属性, P185；

2，使用MFC OLE向导，从对象的类型库中创建组件的包装类，VC程序通过包装类调用, ；

3，#import指令引入类型库，编译器从类型库中读出信息并创建包装类；

使用OLE/COM Object Viewer查看组件信息

系统自带，C:\Program Files (x86)\Windows Kits\8.0\bin\x86\oleview.exe

#### COM+,DCOM,MTS

COM的新版本，更注重于分布式网络应用的设计和实现，继承COM，优化COM；

MTS运行时环境为开发人员提供了许多系统服务：分布式事务，图形化管理工具等，使只关注业务逻辑的解决；

COM+把COM,DCOM,MTS统一， 形成适用于企业的组件技术；

#### 用vc开发com

MFC部分， P214；

ATL

随着Internet的发展，COM要在网络上传输，由于mfc的种种限制，微软推出了全新COM开发工具ATL；

模板类 和 多继承是主要技术；

不要同时使用MFC和ATL；

几个重要的类 P237

CComModule

执着部分功能，引用技术，在本地服务器内类厂注册以及管理自动注册。。。一些COM 核心功能，类似CWinApp；

CComObjectRoot/ CComObjectRoot**Ex**

ATL COM组件都 继承的，类似CCmdTarget；

CComCoClass

组件都 继承的，负责定义类厂；

CComObject

实现IUnKnown接口，聚合等；

### 书签：P73 Visual

P183 COM本质论

P425 VC++ 实践 COM和COM+