COM实现

潘爱民

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

内容

- 复习：COM接口与COM对象
- 注册表
- 类厂
- COM库
- 总结和例子

组件接口

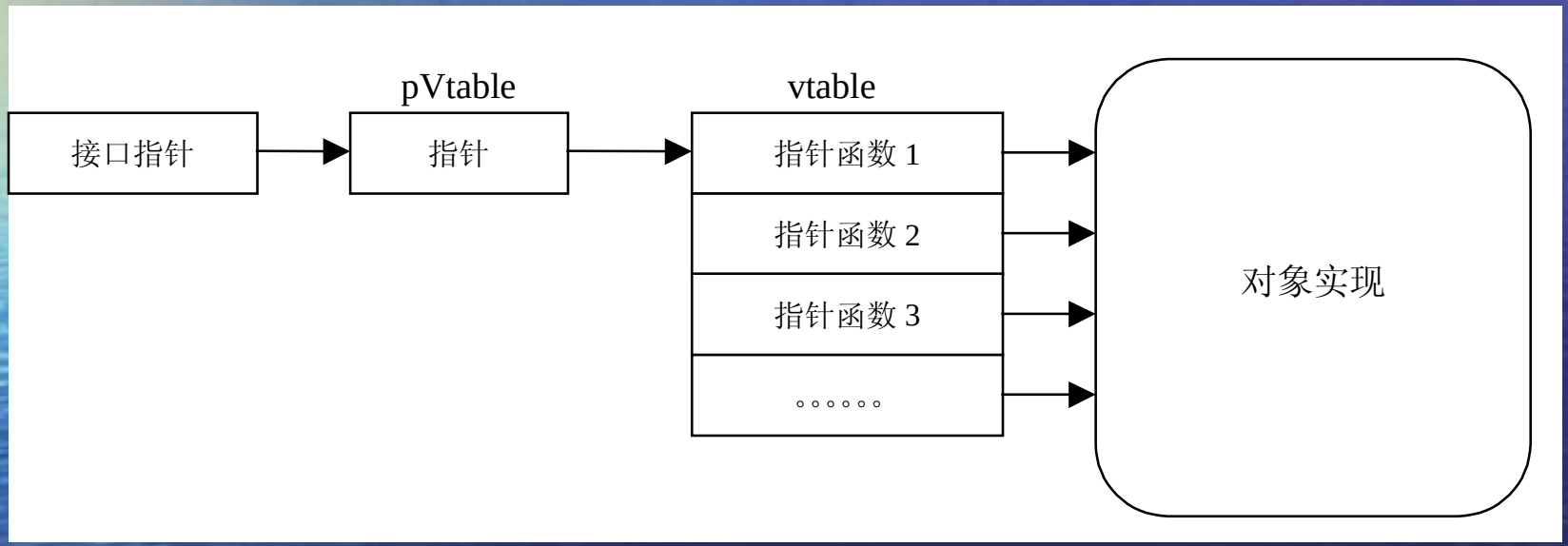
- 第一个里程碑
  - 用vtable作为接口
  - 解决了名字冲突和二进制结构兼容问题
- 第二个里程碑
  - 接口转换：Dynamic_cast
- 第三个里程碑
  - 管理对象生命周期：引用计数

COM接口

- 接口标识：IID
- IUnknown接口：

```
class IUnknown
{
public:
    virtual HRESULT__stdcall  QueryInterface(
                        const IID& iid, void **ppv) = 0 ;
    virtual ULONG   __stdcall  AddRef() = 0;
    virtual ULONG   __stdcall  Release() = 0;
};
```

COM接口结构



COM接口引用计数

- 引用计数的含义
  - Outstanding reference
- 对象实现引用计数
  - AddRef、Release
- 客户显式地操纵引用计数
  - 客户有责任维护好引用计数
- 引用计数规则

COM接口QueryInterface

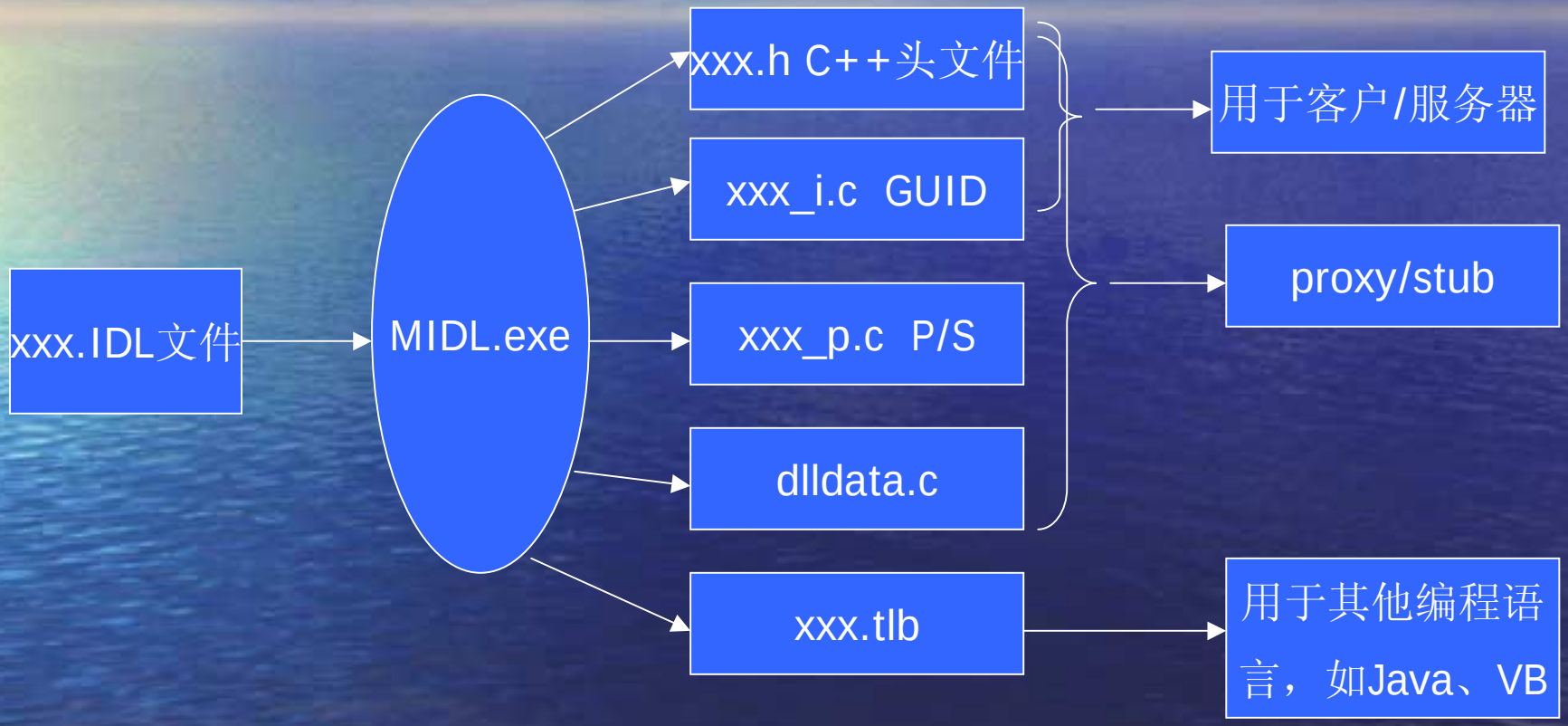
```
HRESULT __stdcall QueryInterface(
                    const IID& iid, void **ppv)
```

- 一个COM对象可以实现多个接口
  - QueryInterface是技术保证
- QueryInterface实现
  - 多继承情况：使用static_cast向上转换
  - 对象身份：IUnknown必须唯一
- 客户通过QueryInterface使用对象的接口
  - 注意：QueryInterface内含AddRef

IDL

- 类C的语言，但不是编程语言
  - 跨语言的中间语言
- 编译器：MIDL.exe产生C++头文件定义
  - 产生C++头文件定义，相当于C++定义
- MIDL.exe产生TLB类型库
  - 许多编程语言利用COM对TLB的支持
- 所有的标准接口都可以在SDK中找到IDL描述
  - 支持import/importlib语法

编译IDL


xxx.IDL文件
MIDL.exe
xxx.h C++头文件
xxx_i.c GUID
xxx_p.c P/S
dlldata.c
xxx.tlb
用于客户/服务器
proxy/stub
用于其他编程语言，如Java、VB

COM对象

- 对象标识：CLSID
- 封装：属性和方法，或者状态和操作
- 对象与接口的联系
  - 如同C++对象与成员函数之间的联系
- COM对象的身份
  - 对象同一性判断准则

实现形式

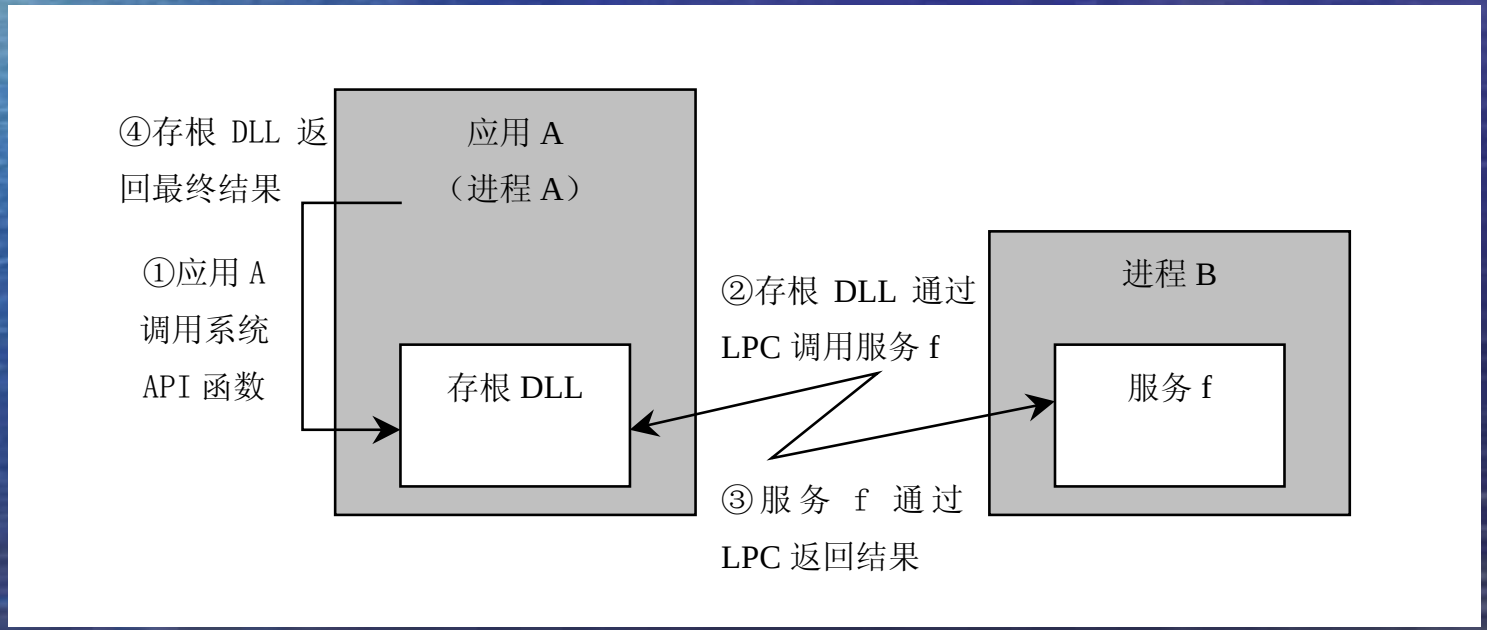
- 进程内组件
  - in process component
- 进程外组件
  - out of process component

进程内组件

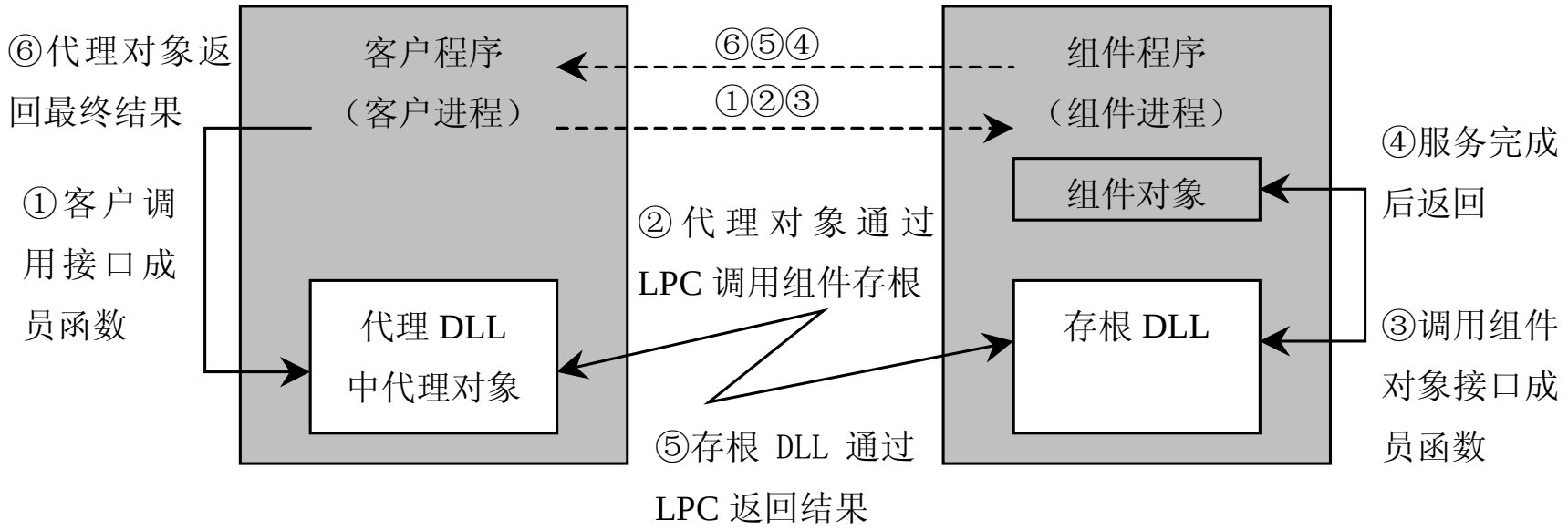
- 组件：做成DLL——引出函数
- 客户：用到的API函数，LoadLibrary、GetProcAddress、FreeLibrary
- 说明：
  - 1. 也可以引出全局变量
  - 2. DumpBin检查组件的引出函数和变量

进程外组件

- 实现形式：EXE
- IPC：DDE、消息机制、共享内存、RPC/LPC等等
- 例：应用调用系统服务



进程外组件(续)


⑥代理对象返回最终结果
客户程序
（客户进程）
⑥⑤④
①②③
组件程序
（组件进程）
④服务完成后返回
①客户调用接口成员函数
组件对象
②代理对象通过LPC 调用组件存根
代理 DLL
中代理对象
存根 DLL
③调用组件对象接口成员函数
⑤存根 DLL 通过LPC 返回结果

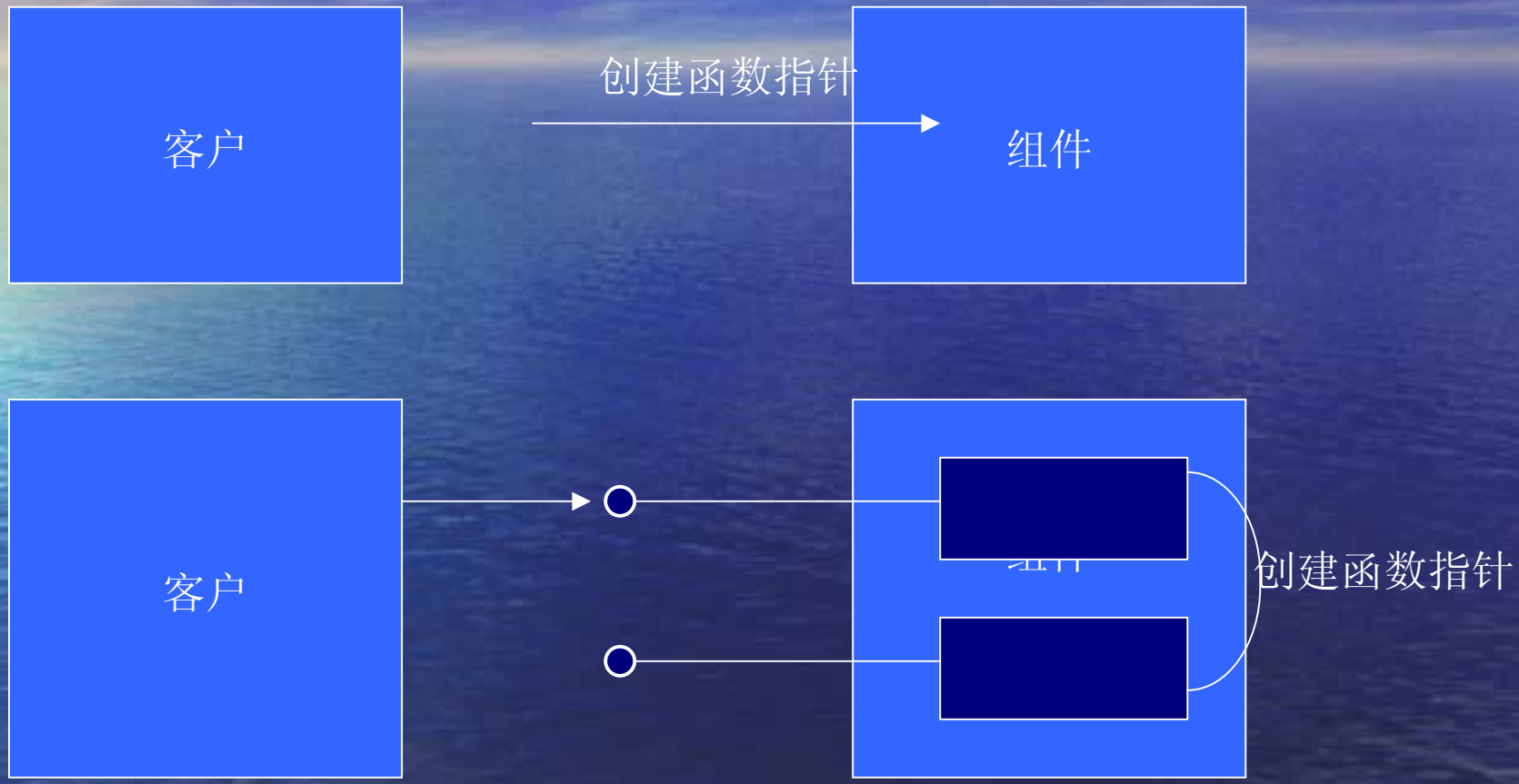
回顾：对象与客户之间的连接

- 客户通过vtable与对象进行通信
- 客户如何获得第一个接口指针？
  - CreateString引出函数
- 如何创建(create)？激活(activate)？
- 创建工作一定是由组件中的一个函数来完成：创建函数CreateObject
- 客户如何访问这个函数？

创建函数

- 方案1
  - 直接引出创建函数
  - 优点：对于DLL非常方便
- 方案2
  - 把创建函数封装到一个对象中，通过vtable调用
  - 优点：灵活，客户以一致的方式调用创建函数

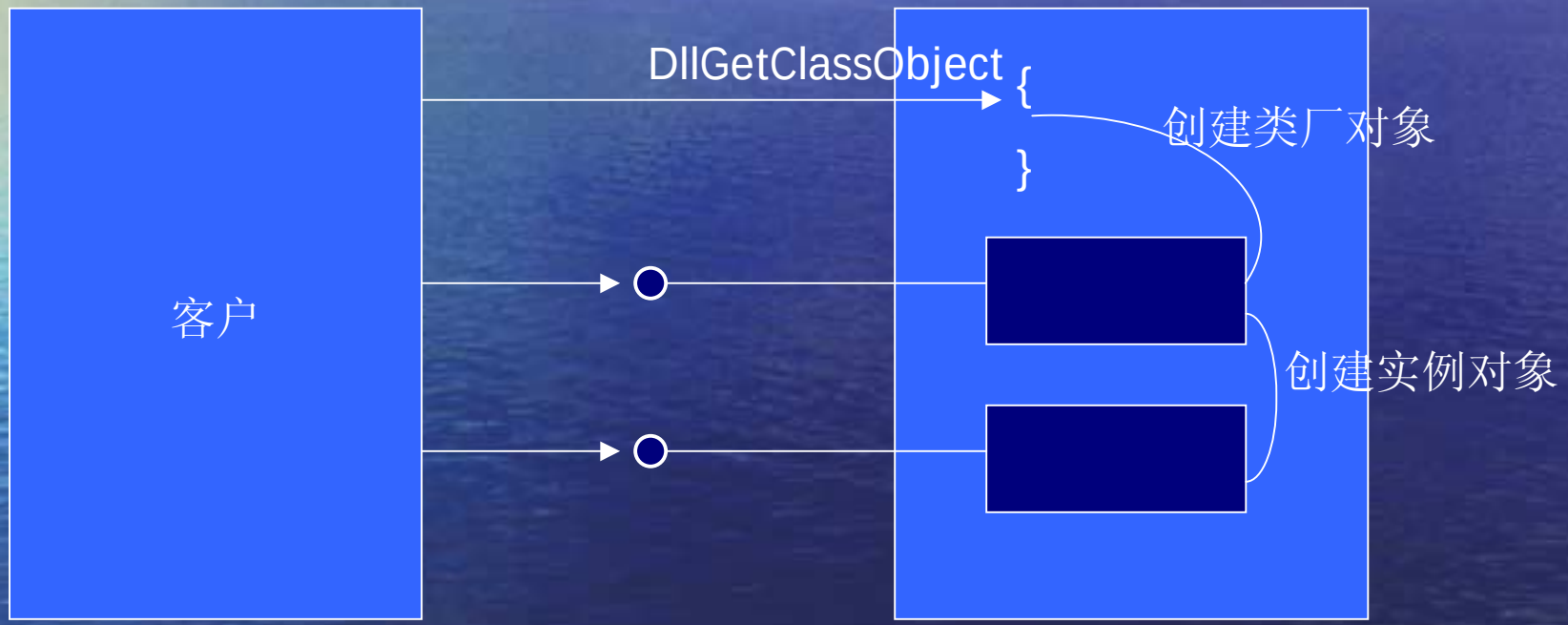
创建函数(续)



创建函数所在的对象

- 该对象被称为类对象，也称为类厂
- 现在问题是：如何创建类厂对象？
  - 对于DLL通过引出函数
  - 对于EXE，EXE的引出函数？
- 客户-〉引出函数-〉类厂对象-〉用户对象
- 引出函数的名字固定：DllGetClassObject
- 增加了一层间接性，带来灵活性

创建对象结构示意图

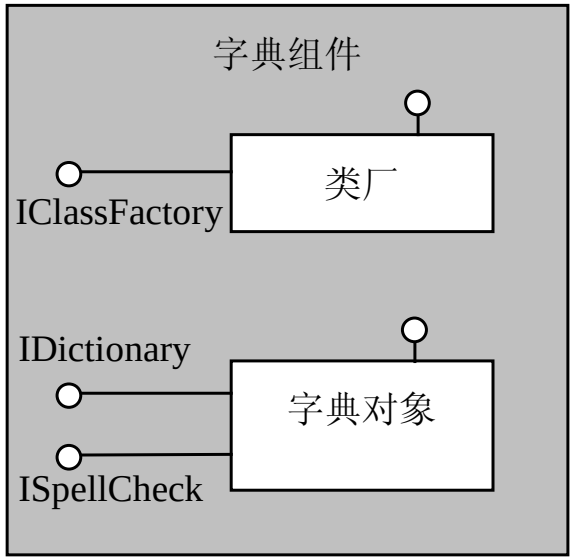


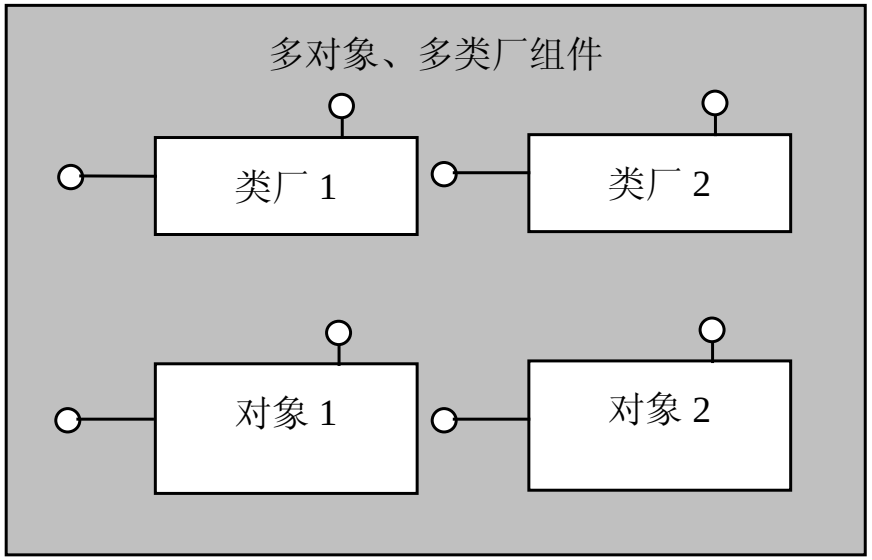
类厂(Class Factory)

- 类厂：用于创建COM对象的COM对象
- 目标：完成COM对象的创建过程，更好地把客户与对象隔离开来。
- 特殊性：
  - 实现一个或多个创建接口，缺省的接口为IClassFactory
  - 类厂本身没有CLSID

类厂(续)

- 类厂与COM对象有一一对应关系





创建类厂对象

- DllGetClassObject创建类厂对象
  - 创建类厂对象需要哪些信息？

- DllGetClassObject原型：

```
HRESULT DllGetClassObject(
            const CLSID& clsid,
            const IID& iid,
            (void **)ppv
);
```

创建函数需要哪些信息？

- clsid
  - 与类厂绑在一起
- iid
  - 客户提供
- 结果接口指针
  - 类型取决于iid

IClassFactory接口

```
class IClassFactory : public IUnknown
{
    virtual HRESULT __stdcall CreateInstance(
                        IUnknown *pUnknownOuter,
                        const IID& iid,
                        void **ppv) = 0;

    virtual HRESULT __stdcall LockServer(
                        BOOL bLock) = 0;
};
```

小结：客户创建对象过程

- 客户提供信息
  - 组件位置、clsid、iid、结果接口指针地址ppv
- 过程：
  - 根据组件位置，LoadLibrary
  - GetProcAddress，获取DllGetClassObject
  - 用clsid和IID_IClassFactory获得类厂对象接口指针pFactory
  - 用iid、ppv调用pFactory->CreateInstance

创建过程的位置透明性

- 位置透明性可以极大地方便客户程序
- 如何做到位置透明性？
  - 在当前环境下，每个clsid必定与某个组件相联系
  - 如何从clsid映射到组件位置？
- 解决方案：
  - 维护clsid与组件位置的映射关系
  - 在客户与组件之间插入中介

COM方案

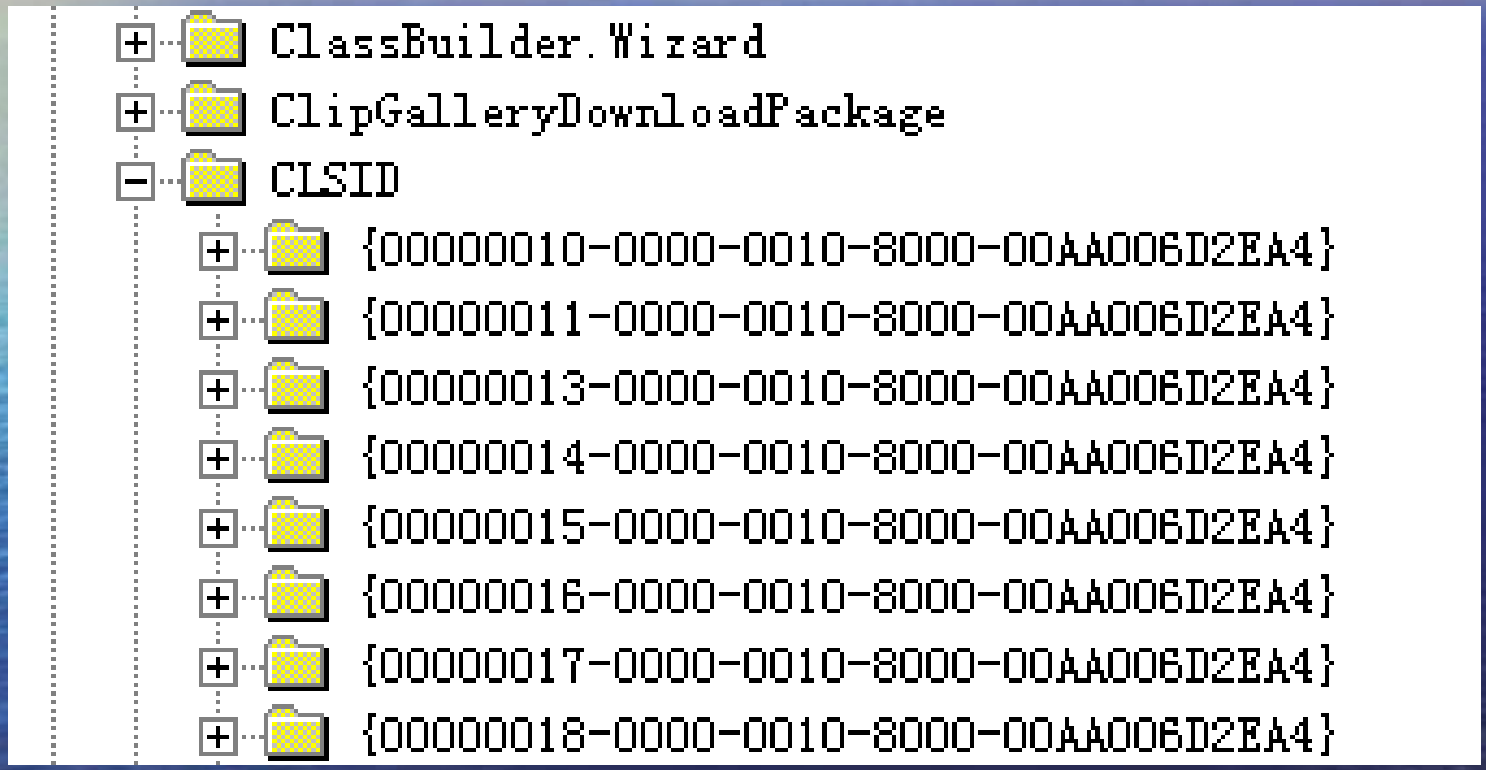
- 在Windows平台上，使用系统注册表保存映射关系，所以，从clsid可以找到对应组件的位置
- 在客户与组件之间插入COM库，由COM库完成创建的细节工作

Windows系统注册表

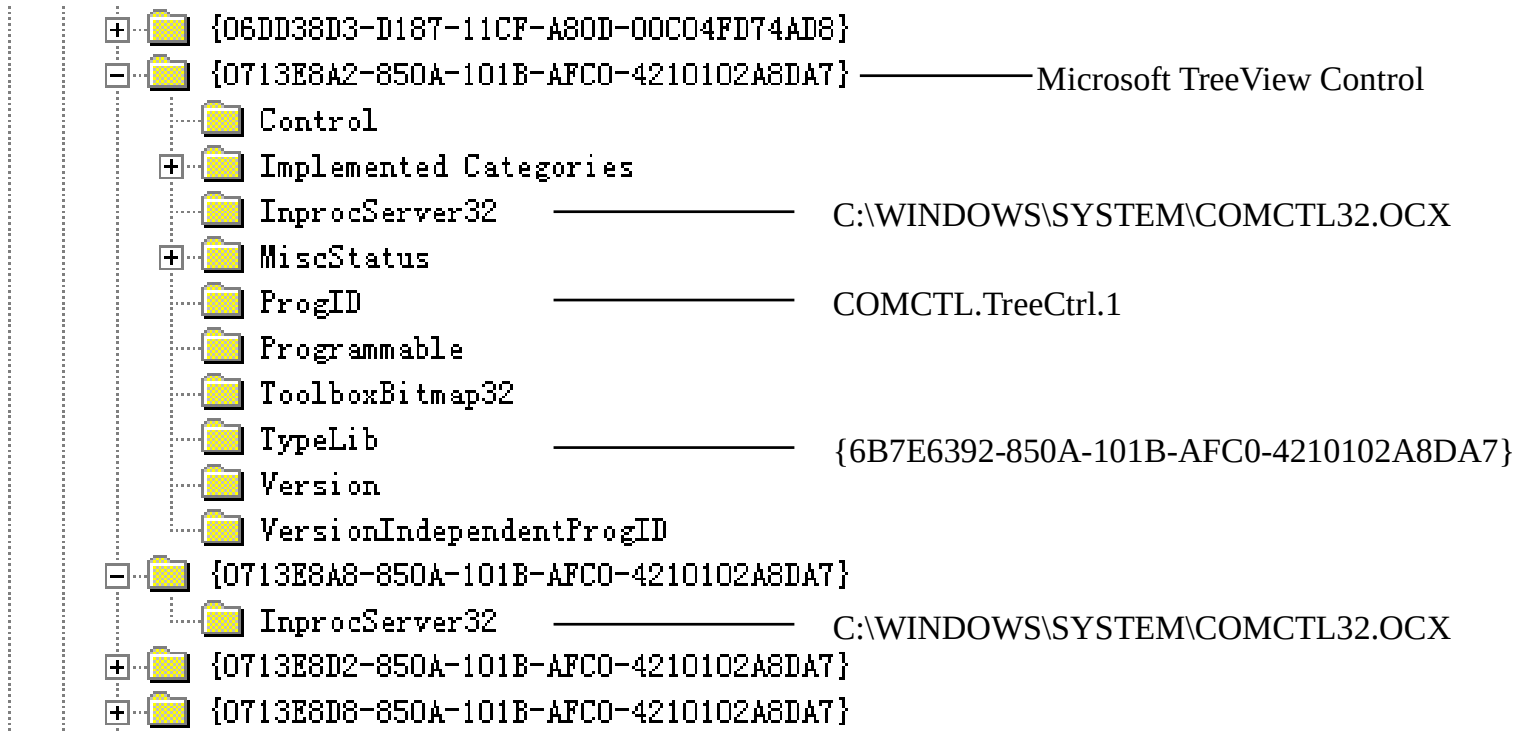
- 树状结构
  - 根是“My Computer”
  - 预定义的5个子节点
  - HKEY_CLASSES_ROOT
    - 为HKEY_LOCAL_MACHINE的一个子节点
  - HKEY_CURRENT_USER
    - 为HKEY_USERS的一个子节点
  - HKEY_LOCAL_MACHINE
  - HKEY_USERS
  - HKEY_CURRENT_CONFIG

通过注册表管理COM对象

- HKEY_CLASSES_ROOT\CLSID



TreeView组件的注册信息

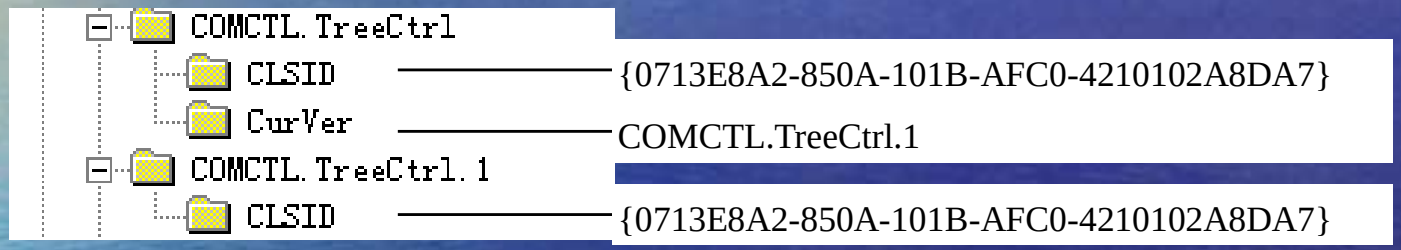


回顾：COM对象的标识

- CLSID，两种形式
  - 128位整数，随机数，不需要运算功能，但是需要比较和查找功能
  - 字符串形式

    例如：{72d3edc2-a4c4-11d0-8533-00c04fd8d503}
- ProgID：友好名，字符串形式
  - 有可能重名，用一种约定来避免重名
  - 例如：Word.Document
  - 包含版本：Word.Document.8

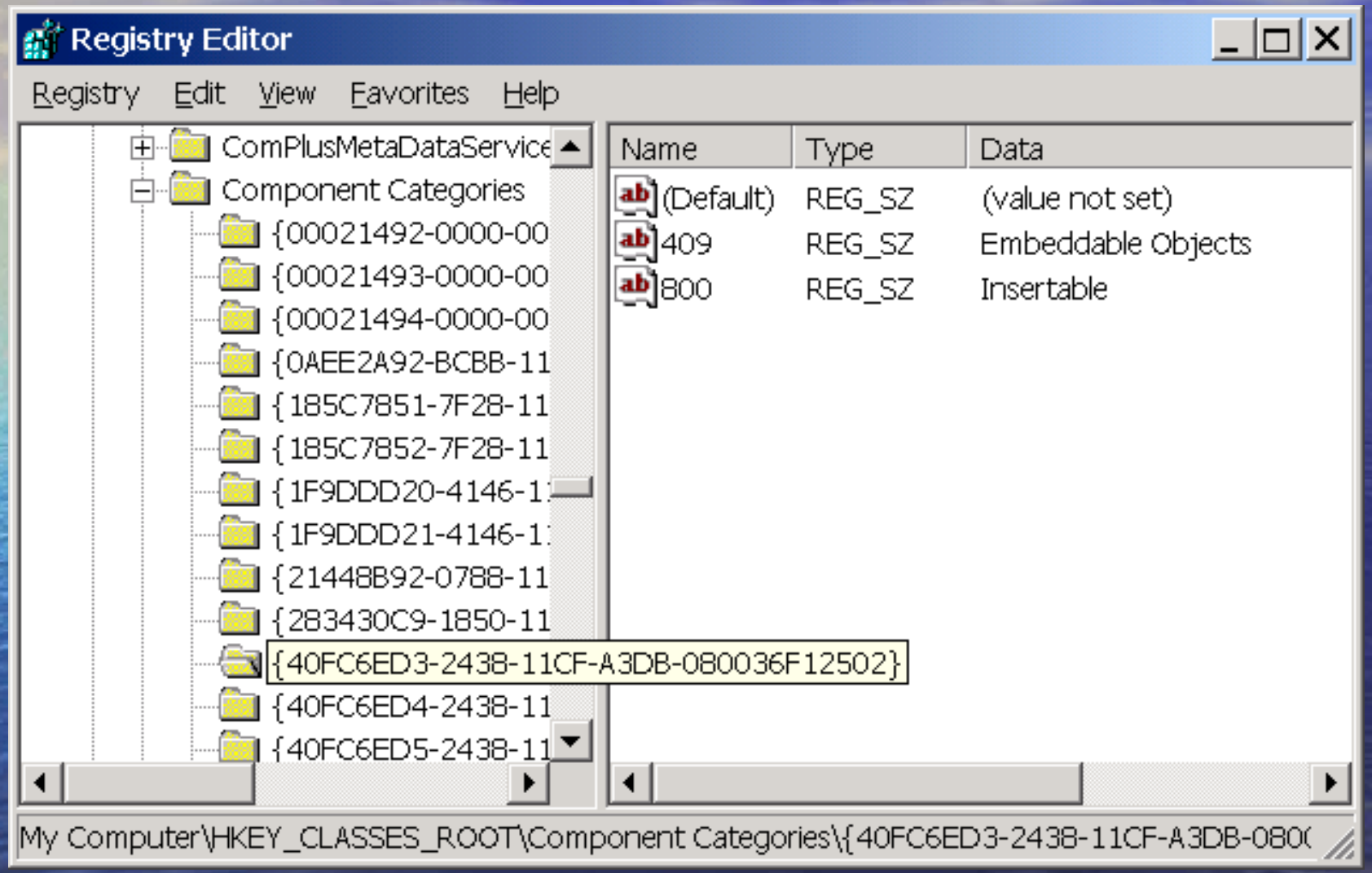
TreeControl的ProgID信息



注册表其他事项

- 系统全局的注册信息、公共信息仓库
- 工具RegEdit.exe、Regedt32、OLEView
- 程序访问途径：Win32 API
- Component Categories（组件类别）

组件类别


Registry Editor
Registry Edit View Favorites Help
ComPlusMetaDataService
Component Categories
{00021492-0000-00
{00021493-0000-00
{00021494-0000-00
{0AEE2A92-BCBB-11
{185C7851-7F28-11
{185C7852-7F28-11
{1F9DDD20-4146-1
{1F9DDD21-4146-1
{21448B92-0788-11
{283430C9-1850-11
{40FC6ED3-2438-11CF-A3DB-080036F12502}
{40FC6ED4-2438-11
{40FC6ED5-2438-11
Name Type Data
(Default) REG_SZ (value not set)
409 REG_SZ Embeddable Objects
800 REG_SZ Insertable
My Computer\HKEY_CLASSES_ROOT\Component Categories\{40FC6ED3-2438-11CF-A3DB-080(

COM组件的注册

- 进程内组件
  - 两个引出函数DllRegisterServer和DllUnregisterServer

    注册工具：RegSvr32.exe

    例如：RegSvr32 c:\DictComp\DictComp.dll

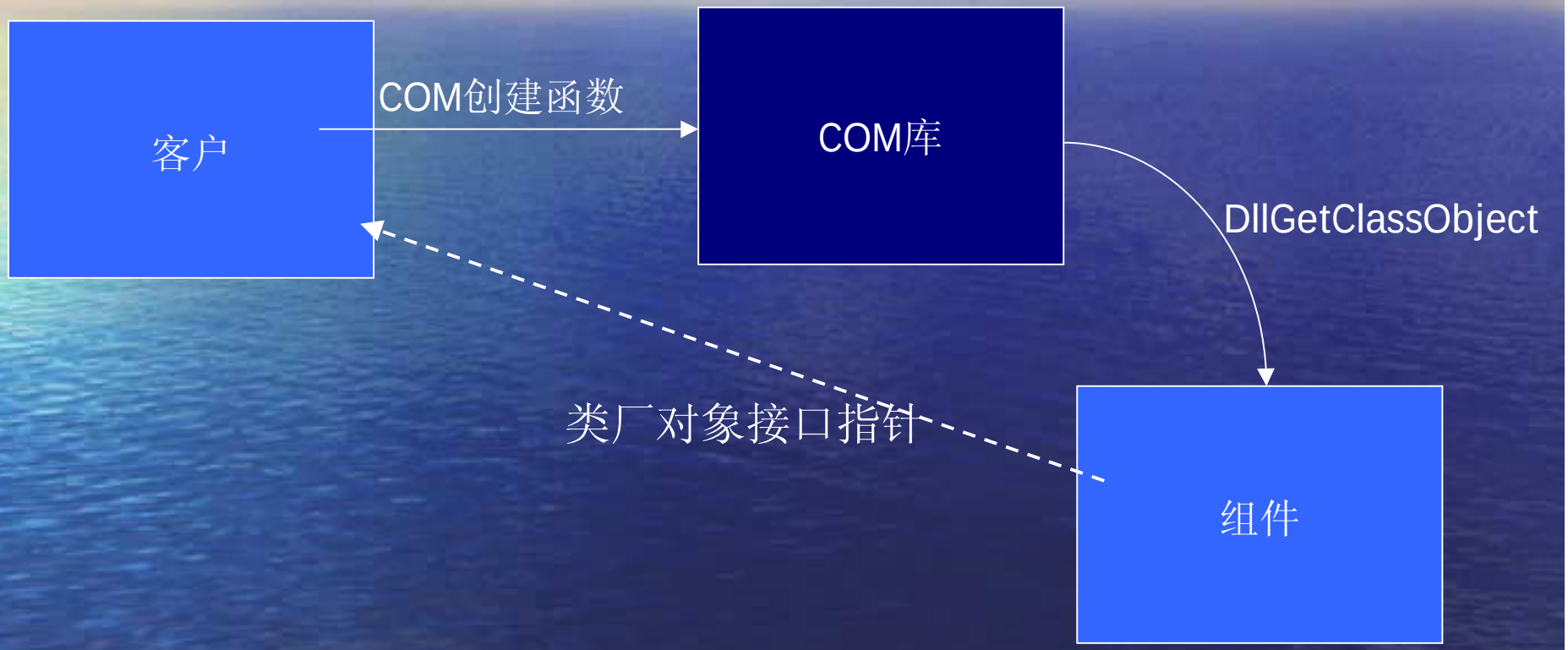
    RegSvr32 /u c:\DictComp\DictComp.dll

- 进程外组件
  - 命令行参数/RegServer和/UnregServer

COM库

- 创建过程
  - COM库处于COM组件和客户中间
- 调用过程
  - 对于进程内组件， COM库不再参与处理

COM对象创建过程



COM创建函数

- COM库中三个用于创建组件的函数：
  CoGetClassObject
  CoCreateInstance
  CoCreateInstanceEx

CoGetClassObject

- 创建一个类厂

```
HRESULT CoGetClassObject(
    const CLSID& clsid,
    DWORD dwClsContext,
    COSERVERINFO   *pServerInfo,
    const IID& iid,
    (void **)ppv
);
```

CoCreateInstance

```
HRESULT CoCreateInstance(
    const CLSID& clsid,
    IUnknown *pUnknownOuter,
    DWORD dwClsContext,
    const IID& iid,
    (void **)ppv
);
```

CoCreateInstance实现伪码

```
HRESULT CoCreateInstance(const CLSID& clsid, IUnknown  *pUnknownOuter,
                         DWORD dwClsContext, const IID& iid, void  *ppv)
{
    IClassFactory  *pCF;
    HRESULT    hr;

    hr = CoGetClassObject(clsid, dwClsContext, NULL,
                          IID_IClassFactory, (void *)pCF);

    if  (FAILED(hr))
            return hr;

    hr = pCF->CreateInstance(pUnkOuter, iid, (void *)ppv);
    pCF->Release();

    return hr;
}
```

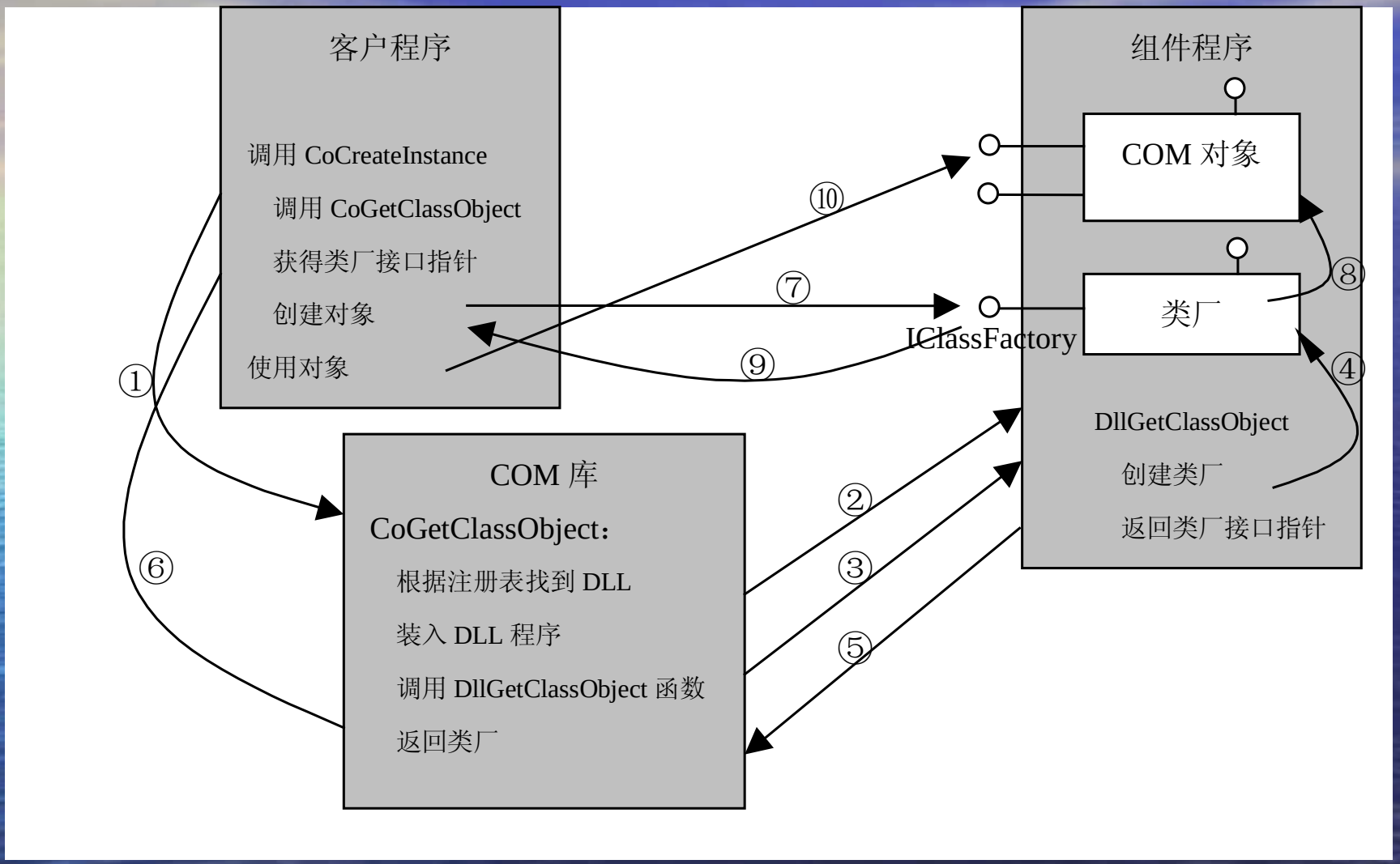
CoCreateInstanceEx

```
HRESULT CoCreateInstanceEx(
    const CLSID& clsid,
    IUnknown *pUnknownOuter,
    DWORD dwClsContext,
    COSERVERINFO  *pServerInfo,
    DWORD dwCount,
    MULTI_QI *rgMultiQI
);
```

三个创建函数选用原则

- 如果客户创建远程对象或者希望一次获取对象的多个接口指针，则选用CoCreateInstanceEx函数；
- 如果客户希望获取类厂对象或者要调用类厂的某些成员函数，则选用CoGetClassObject函数；
- 在其他情况下，使用CoCreateInstance函数创建对象，这是最常用的方法。

创建过程示意图


客户程序
调用 CoCreateInstance
调用 CoGetClassObject
获得类厂接口指针
创建对象
使用对象
组件程序
COM 对象
类厂
IClassFactory
DllGetClassObject
创建类厂
返回类厂接口指针
COM 库
CoGetClassObject：
根据注册表找到 DLL
装入 DLL 程序
调用 DllGetClassObject 函数
返回类厂
①
②
③
④
⑤
⑥
⑦
⑧
⑨
⑩

类厂的实现

```
class CDictionaryFactory : public IClassFactory
{
  protected:
    ULONG          m_Ref;
  public:
    CDictionaryFactory (void);
    ~ CDictionaryFactory (void);

    //IUnknown members
    HRESULT QueryInterface(const IID& iid, void **ppv);
    ULONG AddRef();
    ULONG Release();

    //IClassFactory members
    HRESULT CreateInstance(IUnknown *, const IID& iid, void **ppv)
    HRESULT LockServer(BOOL);
};
```

CreateInstance函数的实现

```
HRESULT CDictionaryFactory::CreateInstance(IUnknown
    *pUnknownOuter, const IID& iid, void **ppv)
{
    CDictionary * pObj;
    HRESULT hr;
    *ppv=NULL;
    hr=E_OUTOFMEMORY;
    if (pUnknownOuter != NULL)
        return CLASS_E_NOAGGREGATION;

    pObj=new CDictionary();
    if (pObj== NULL) return hr;

    //待续
```

CreateInstance函数的实现(续)

```
    //续上页
    //Obtain the first interface pointer (which does an AddRef)
    hr=pObj->QueryInterface(iid, ppv);

    if (hr != S_OK)
    {
        g_DictionaryNumber --;
        delete pObj;
    }
    return hr;
}
```

DllGetClassObject的实现

```
extern "C" HRESULT __stdcall DllGetClassObject(const CLSID&
    clsid, const IID& iid, void **ppv)
{
    if (clsid == CLSID_Dictionary )  {
        CDictionaryFactory *pFactory = new CDictionaryFactory;

        if (pFactory == NULL)   {
            return E_OUTOFMEMORY ;
        }

        HRESULT result = pFactory->QueryInterface(iid, ppv);
        return result;
    } else  {
        return CLASS_E_CLASSNOTAVAILABLE;
    }
}
```

类厂对组件生存期的控制

- 组件引用计数不计类厂
- IClassFactory::LockServer函数

COM库

- COM库的初始化
- COM库的内存管理
- 组件程序的装载和卸载
- 常用函数和HRESULT

COM库的组成

- 用于创建过程的SCM(Service Control Manager)
  - rpcss.exe
  - ole32.dll
- 其他
  - 提供COM环境
  - 管理server、组件等
  - ……

COM库的组成(续)


COM应用
(COM client)
COM应用
(COM server)
OLE32.DLL
OLE32.DLL
Service Control Manager
RPCSS.EXE

COM库的初始化

- 基本的初始化函数：
  - HRESULT CoInitialize(void *pReserved);
- 初始化之前唯一可以调用的函数：
  - DWORD CoBuildVersion();
- 另一个初始化函数：
  - CoInitializeEx
- COM库的终止函数：
  - void CoUninitialize(void);

有关CLSID和ProgID的COM函数

- IsEqualGUID、IsEqualIID、IsEqualCLSID
- CLSIDFromProgID、ProgIDFromCLSID
- StringFromCLSID、CLSIDFromString
- StringFromIID、 IIDFromString
- StringFromGUID2
  - 内存由调用者分配
- 注意：COM库函数的字符串使用OLECHAR类型

COM库的内存管理

- COM库提供了内存管理器以及内存管理器的标准

```
HRESULT CoGetMalloc(DWORD dwMemContext, IMalloc **ppMalloc);

class IMalloc : public IUnknown
{
    void *      Alloc(ULONG cb) = 0;
    void *      Realloc( void * pv,  ULONG cb) = 0;
    void        Free(void* pv) = 0;
    ULONG       GetSize( void * pv) = 0;
    int         DidAlloc(void * pv) = 0;
    void        HeapMinimize()= 0;
};
```

COM库内存管理用法（一）

```
DWORD     length = MAX_LENGTH;
IMalloc * pIMalloc;
HRESULT   hr;

hr=CoGetMalloc(MEMCTX_TASK, &pIMalloc);
if (hr != S_OK)
    // return failure

psz=pIMalloc->Alloc(length);
pIMalloc->Release();

if (NULL==psz)
    // return failure
......
pszText = psz;
```

COM库内存管理用法（二）

- 三个封装函数：

```
void * CoTaskMemAlloc(ULONG cb);
void CoTaskMemFree(void *pv);
void CoTaskMemRealloc(void *pv, ULONG cb);
```

COM库内存管理用法（三）

```
DWORD      length = MAX_LENGTH;
IMalloc * pIMalloc;
HRESULT    hr;

psz=CoTaskMemAlloc (length);

if (NULL==psz)
    // return failure
......
pszText = psz;
```

COM库内存管理用法（四）

```
WCHAR *pwProgID;
char pszProgID[128];
hResult = ::ProgIDFromCLSID(CLSID_Dictionary,
                            &pwProgID);
if  (hResult != S_OK)      {
      ……
}
wcstombs(pszProgID, pwProgID, 128) ;
CoTaskMemFree(pwProgID);
```

组件程序的装载和卸载

- 进程内组件的装载
  - DllGetClassObject
- 进程外组件的装载
  - “/Embedding”命令行参数
- 进程内组件的卸载
  - CoFreeUnusedLibraries
- 进程外组件的卸载
  - main或者WinMain函数退出

进程内组件的卸载

- 组件不能自己卸载
- 客户调用COM库函数CoFreeUnusedLibraries
- COM库调用DLL组件的引出函数
  - HRESULT DllCanUnloadNow();
  - 若DllCanUnloadNow返回S_OK，则同意卸载
  - 若DllCanUnloadNow返回S_FALSE，则不同意卸载
- DllCanUnloadNow实现：对象计数+锁计数

COM库中一些常用函数

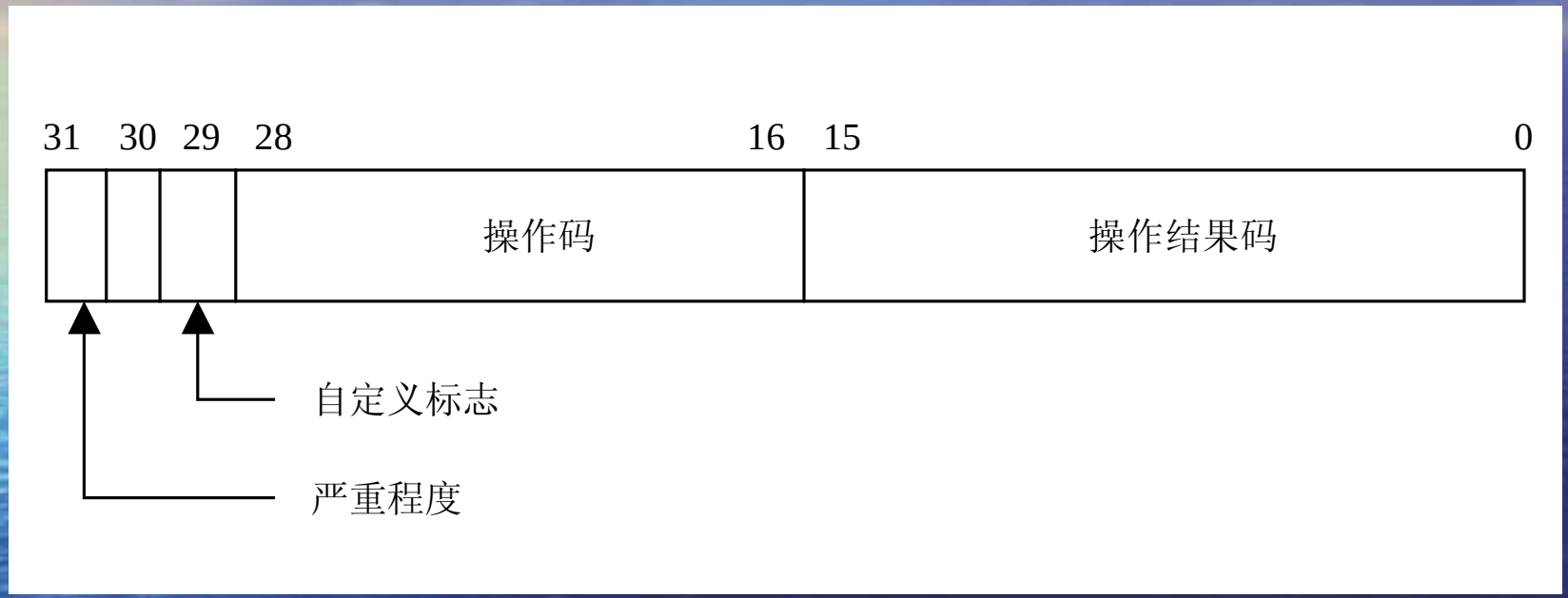
- 初始化函数
- GUID有关的函数
- 对象创建函数
- 内存管理函数

COM 库中一些常用函数

类别	函数	功能
初始化函数	CoBuildVersion	获取 COM 库的版本号
	CoUnitialize	COM 库的初始化
	CoUninitialize	COM 库功能服务终止
	CoFreeUnusedLibraries	释放进程中所有不再使用的组件程序
GUID 有关的函数	IsEqualGUID	判断两个 GUID 是否相等
	IsEqualIID	判断两个 IID 是否相等
	IsEqualCLSID	判断两个 CLSID 是否相等
	CLSIDFromProgID	把字符串形式的对象标识转化为 CLSID 结构形式
	StringFromCLSID	把 CLSID 结构形式转化为字符串形式
	IIDFromString	把字符串形式的 IID 转化为 IID 结构形式
	StringFromIID	把 IID 结构形式转化为字符串形式
	StringFromGUID2	把 GUID 结构形式转化为字符串形式
对象创建函数	CoGetClassObject	获取对象的类厂
	CoCreateInstance	创建 COM 对象
	CoCreateInstanceEx	创建 COM 对象，可指定多个接口或远程对象
	CoRegisterClassObject	登记一个对象，以便其它应用可以连接到该对象
	CoRevokeClassObject	取消对象的登记操作
	CoDisconnectObject	断开其它应用与对象的连接
内存管理函数	CoTaskMemAlloc	内存分配函数
	CoTaskMemRealloc	内存重新分配函数
	CoTaskMemFree	内存释放函数
	CoGetMalloc	获取 COM 库的内存管理器接口

HRESULT数据结构

- 表达方法的操作结果，32位整数



- 类别码：反映了函数调用结果的基本情况
- 操作码：标识了结果操作来源

HRESULT

- 类别码
  - 00 - 表示函数调用成功
  - 01 - 包含了一些信息
  - 10 - 警告
  - 11 - 错误
- 操作码

```
#define FACILITY_WINDOWS     8
#define FACILITY_STORAGE     3
#define FACILITY_RPC         1
#define FACILITY_SSPI        9
#define FACILITY_WIN32       7
#define FACILITY_CONTROL     10
#define FACILITY_NULL        0
#define FACILITY_INTERNET    12
#define FACILITY_ITF         4
#define FACILITY_DISPATCH    2
#define FACILITY_CERT        11
```

- Win32 SDK的头文件WinError.h

HRESULT(续)

- FormatMessage函数
- SUCCEEDED和FAILED宏
- 常用定义

最常用的 HRESULT 宏定义

返回值	说明
S_OK	函数执行成功，其值为 0。*
S_FALSE	函数执行成功，返回布尔 FALSE，其值为 1。*
E_FAIL	函数执行失败，失败原因不确定。
E_OUTOFMEMORY	函数执行失败，失败原因为内存分配不成功。
E_NOTIMPL	函数执行失败，成员函数没有被实现。
E_NOTINTERFACE	函数执行失败，组件没有实现指定的接口，用于 QueryInterface 成员函数的返回值。

COM实现过程

- 进程内组件与客户的协作过程
- 进程外组件与客户的协作过程

实现一个进程内COM组件的步骤

- 定义必要的CLSID和IID
- 实现COM对象
  - 通过QueryInterface暴露其接口
  - 管理引用计数，注意对全局引用计数的维护
- 实现类厂对象
  - 对象的引用计数不记在全局对象引用计数内
  - 维护锁计数
- 实现DllGetClassObject、DllCanUnloadNow
- (可选)实现两个注册函数

进程内组件与客户的协作过程

客户程序	COM 库	组件程序(DLL)
CLSID clsid IClassFactory *pClf; IUnknown *pUnknown; CoInitialize(NULL); CLSIDFromProgID("Dictionary.Object", &clsid);		
	COM 在注册表中查找字典 CLSID	
CoGetClassObject(clsid, CLSCTX_INPROC_SERVER, NULL, IID_IClassFactory, (void **)&pClf);		

	COM 库在内存中查找 clsid 组件
	if (DictComp.dll 还没被装入内存)
	{
	从注册表中获取组件程序
	全路径名 ...\DictComp.dll
	CoLoadLibrary(...)
	}
	调用 DLL 的 DllGetClassObject	

		创建类厂对象 CDictionaryFactory 并返回 IClassFactory 接口
	COM 库返回 IClassFactory 接口给客户	

```
pClf->CreateInstance(NULL,
    IID_IUnknown,
    (void **)&pUnknown);
```

类厂对象的 CreateInstance 函数被调用(通过组件的 vtable 被客户直接调用)

用 new 操作符构造字典组件对象

```
new CDictionary;
```

返回 IUnknown 接口指针

客户使用字典组件，通过其接口进行各种操作.....

```
pClf->Release();
pUnknown->Release();
```

组件对象的 Release 函数被调用(也是通过组件的 vtable)

```
if (m_Ref == 0)
{
    delete this;
    return 0;
}
```

```
CoFreeUnusedLibraries()
```

COM 库调用字典组件的引出函数 DllCanUnloadNow

DllCanUnloadNow 函数中：

```
if (不存在字典对象
  && 锁计数器为 0)
    return TRUE;
else
    return FALSE;
```

```
if (字典组件 DllCanUnloadNow 函数返回 TRUE)
    CoFreeLibrary(...);
```

```
CoUninitialize()
```

COM 库释放资源

客户程序退出

进程外组件与客户的协作过程

客户程序	COM 库	组件程序(EXE)
CLSID clsid IClassFactory *pClf; IUnknown *pUnknown; CoInitialize(NULL); CLSIDFromProgID("...", &clsid);		
	COM 在注册表中组件对象 CLSID	
CoGetClassObject(clsid, CLSCTX_LOCAL_SERVER, NULL, IID_IClassFactory, (void **)&pClf);		

	COM 库在内存中查找 clsid 组件
	if (组件 EXE 还没有启动
	|| COM 需要另一个实例)
	{
	从注册表中获取 EXE 组件名
	创建组件进程
	}
	等待	

调用 CoInitialize

创建组件支持的各种类厂对象

调用 COM 函数注册所有类厂对象：CoRegisterClassObject

等组件注册完成后，COM 库把 IClassFactory 接口返回给客户

```
pClf->CreateInstance(NULL,
    IID_IUnknown,
    (void **)&pUnknown);
```

类厂对象的 CreateInstance 函数被调用(通过进程中代理对象被间接调用)

构造组件对象

返回 IUnknown 接口指针

客户使用字典组件，通过其接口(间接)进行各种操作.....

```
pClf->Release();
pUnknown->Release();
```

组件对象的 Release 函数被调用

```
if (m_Ref == 0)
{
    delete this;
    return 0;
}
```

如果满足退出条件则正常退出

```
CoUninitialize ()
```

COM 库对所有该客户没有成功释放的对象调用 Release 函数

CoUninitialize ()

组件程序退出

COM 库释放资源

客户程序退出

自由作业

- 读程序
  - “COM原理与应用”中
    - 第二章的“字典组件和字典客户”
    - 两个工程：DictComp和DictCtrl
  - 或者“Inside COM”
    - 第七章的例子程序
  - 目标：

    了解COM组件的创建过程

    进一步理解COM组件的位置透明性