COM细节

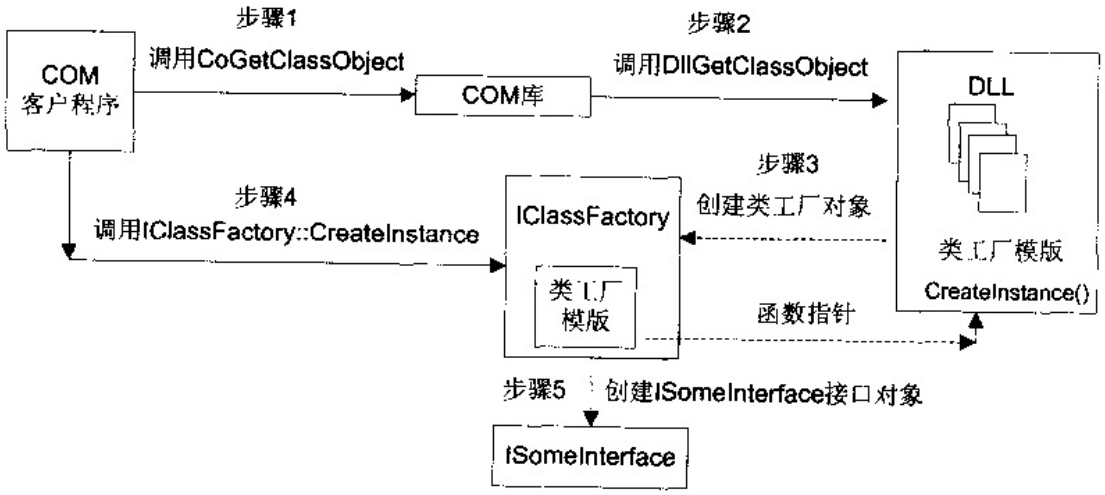
在注册表*计算机\HKEY_CLASSES_ROOT\CLSID{xxx}\InprocServer32*中保存了插件的路径，供加载插件时使用

在高版本中，写入的注册表项会重定位，以32位为例，会重定位到**计算机\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\WOW6432Node\CLSID{xxx}\InprocServer32**

导出实现以下函数

```
LIBRARY
EXPORTS
    DllGetClassObject    PRIVATE
    DllCanUnloadNow      PRIVATE
    DllRegisterServer    PRIVATE
    DllUnregisterServer  PRIVATE
```

过程



DllGetClassObject

从DLL对象处理程序或对象应用程序检索类对象。

```
HRESULT DllGetClassObject(
  REFCLSID rclsid,      // 类的GUID
  REFIID   riid,        // 接口的GUID
  LPVOID   *ppv         // 请求接口指针的地址
);
```

示例

```
HRESULT_export CALLBACK DllGetClassObject
    (REFCLSID rclsid, REFIID riid, LPVOID * ppvObj)
{
    HRESULT hr = E_OUTOFMEMORY;
    *ppvObj = NULL;

    CClassFactory *pClassFactory = new CClassFactory(rclsid);
    if (pClassFactory != NULL)   {
        hr = pClassFactory->QueryInterface(riid, ppvObj);
        pClassFactory->Release();
    }
    return hr;
}
```

DllCanUnloadNow

确定实现此函数的DLL是否正在使用。如果没有，调用者可以从内存中卸载DLL。

```
HRESULT DllCanUnloadNow();
```

DllRegisterServer

在注册表中注册模块

```
HRESULT DllRegisterServer();
```

示例

```
STDAPI DllRegisterServer(void)
{
    char szPath[MAX_PATH];

    for (int i = 0; i < sizeof(szReg) / sizeof(szReg[0]); i++)
    {
        const char* pSubKey = szReg[i][0];
        const char* pValueName = szReg[i][1];
        const char* pValue = szReg[i][2];

        if (pValue == (const char*)-1)
        {
            //获取路径
            GetModuleFileName(g_hModule, szPath, sizeof(szPath));
            pValue = szPath;
        }

        HKEY hKey;
        RegCreateKey(HKEY_CLASSES_ROOT, pSubKey, &hKey);
        RegSetValueEx(hKey, pValueName, NULL, REG_SZ, (CONST BYTE*)pValue,
strlen(pValue));
        RegCloseKey(hKey);
```

```
    }
    return S_OK;
}
```

DllUnregisterServer

在注册表中卸载由`DllRegisterServer`注册的模块

```
HRESULT DllUnregisterServer();
```

示例

```
//导出卸载函数
STDAPI DllUnregisterServer(void)
{
    for (int i = sizeof(szReg) / sizeof(szReg[0]) - 1; i >= 0; i--)
    {
        const char* pSubKey = szReg[i][0];
        RegDeleteKey(HKEY_CLASSES_ROOT, pSubKey);
    }

    return S_OK;
}
```

使用COM

```
int main()
{
    HRESULT hr;
    ISuperMath* pObject = NULL;
    // 初始化
    CoInitialize(NULL);
    // 创建COM组件对象
    hr = CoCreateInstance(CLSID_CSuperMath, NULL, CLSCTX_INPROC_SERVER,
IID_ISuperMath, (void**)&pObject);
    if (FAILED(hr))
        return 0;
    long ret;
    pObject->Add(1, 2, &ret);
    printf("1+2=%d\n", ret);
    pObject->Release();
    // 反初始化
    CoUninitialize();
    return 0;
}
```

BSTR

BSTR 字符串结合了C-style 字符串和Pascal-style字符串。它在前4字节存储了字符串长度（字符串的字节数，但不包括Null结束符），在字符串结尾以字符0识别。和pascal-style 字符串不同，指向BSTR字符串的指针指向第一个字符，而非开首的字符串长度。故此适用于读取C-style 字符串的程序库同样适用于BSTR字符串（但写入则另作别论）。 字符是按照Unicode编码保存。允许在BSTR串中间嵌入NULL字符。

> Windows提供了BSTR相关函数：
>
> - 分配空间并初始化BSTR，这也相当于char*转换成BSTR：例如，BSTR bstrText = ::SysAllocString(L"Test");
> - BSTR SysAllocString(const OLECHAR * psz);
> - INT SysReAllocString(BSTR* pbstr,const OLECHAR* psz);
> - BSTR SysAllocStringLen(const OLECHAR * strIn, UINT ui);
> - INT SysReAllocStringLen(BSTR* pbstr,const OLECHAR* psz,unsigned int len);
> - void SysFreeString(BSTR bstrString);
> - UINT SysStringLen(BSTR); //长度是指字符串中字符个数，而非字节数
> - UINT SysStringByteLen(BSTR bstr); 长度是指字符串中字节数
> - BSTR SysAllocStringByteLen(LPCSTR psz,UINT len);
> - 使用*variant_t把char*转换成BSTR：例如，*variant_t strVar("This is a test"); BSTR bstrText = strVar.bstrVal;
> - 使用_bstr_t函数，例如： BSTR bstrText = _bstr_t("This is a test");
> - 使用ATL中专门用于操作BSTR字符的CComBSTR类：例如BSTR bstrText = CComBSTR("This is a test");
> - 使用ConvertStringToBSTR。例如： char* lpszText = "Test"; BSTR bstrText = _com_util::ConvertStringToBSTR(lpszText);
> - 使用ConvertBSTRToString：例如，char* lpszText2 = _com_util::ConvertBSTRToString(bstrText);
> - 使用_bstr_t的类型转换运算符重载把BSTR转换成char*：例如， char* lpszText2 = bstrText;
> - CString转换成BSTR通常是通过使用CStringT::AllocSysString来实现。例如：CString str("This is a test");BSTR bstrText = str.AllocSysString();
> - BSTR转换成CString，例如：CStringA str=bstrText;

IDL

接口描述语言（Interface description language，缩写IDL），是用来描述软件组件界面的一种计算机语言。IDL通过一种中立的方式来描述接口，使得在不同平台上运行的对象和用不同语言编写的程序可以相互通信交流。

语法说明

1. 包含其他文件

```
; 包含unknwn.idl
import "unknwn.idl";
```

2. 接口

```
; 属性列表
[
    object,
    uuid(2729A764-828A-46c9-9E53-842EEAFB323D)
]
; 接口类
interface IMyString : IUnknown
{
    ; helpstring为给其他语言的注释
    [helpstring("初始化字符串")]
    ; 方法 [attributes]HRESULT name(param_list)
    HRESULT InitData([in]BSTR bstrInit);

    [helpstring("获取字符串")]
    HRESULT GetData([out, retval]BSTR *pbstrResult);

    [helpstring("获取字符串长度")]
    HRESULT GetLength([out, retval]long *plLen);
};
```

- 属性:
  - `object`：是指定接口是COM接口的方法。没有object属性，接口被认为DCE RPC(分布式计算环境远程过程调用)微软因此增加了object属性，作为为了支持COM。所有你定义的COM接口将有object标志。即所有的COM接口必须要有object属性。
  - `uuid`：指定了接口的GUID,使接口被唯一的标示。
  - `helpstring`：最大长度255字节
- 参数:
  - `in`：传入参数
  - `out`：传出参数
  - `retval`：将参数改为返回值

3. 类型库

```
; 属性列表
[
    uuid(39A61464-0E8B-4f66-941A-9057229D65C5),
    version(1.0)
]
; 库
library MyStringLib
{
    importlib("stdole.tlb");

    ; 属性
    [
        uuid(9A064DDA-6453-4b97-9EFA-2372F3255987),
        helpstring("字符串处理")
    ]
    ; 对象
    coclass MyString
    {
        interface IMyString;
```

```
        interface IMyMath;
    };
};
```

COM在逆向中的应用

1. 对于直接继承自`IUnknown`接口的COM对象而言，其功能函数在虚表的第四项

```
interface IUnknown {
  virtual HRESULT QueryInterface (REFIID riid, void **ppvObject) = 0;
  virtual ULONG   AddRef () = 0;
  virtual ULONG   Release () = 0;
};
```

2. 对于继承自`IDispatch`接口的COM对象而言，其功能函数在虚表的第八项

```
interface IDispatch : public IUnknown {
  virtual HRESULT GetTypeInfoCount(unsigned int * pctinfo) = 0;
  virtual HRESULT GetTypeInfo(unsigned int iTInfo, LCID lcid, ITypeInfo **
ppTInfo) = 0;
  virtual HRESULT GetIDsOfNames(REFIID riid, OLECHAR ** rgszNames, unsigned
int cNames, LCID lcid, DISPID * rgDispId) = 0;
  virtual HRESULT Invoke(DISPID dispIdMember, REFIID riid, LCID lcid, WORD
wFlags, DISPPARAMS * pDispParams, VARIANT * pVarResult, EXCEPINFO *
pExcepInfo, unsigned int * puArgErr) = 0;
};
```