2020/12/18_COM_第4课_插件的设计(注册表写入与封装)

笔记本:	COM
创建时间:	2020/12/18 星期五 15:01
作者:	ileemi
标签:	插件的设计(注册表写入与封装)

接口中的全局变量应放置在独立的 .cpp 中。

组件跨语言问题

- int：各个编译器对类型的长度不固定（接口参数的类型：int，推荐使用新类型：int32_t（对应的头文件：stdint.h））老编译器编译的int为2个字节，现在一般的int为4字节。long -- 一般为4字节，通用性较强。
- 字符串格式：(1)通用性使用 wchar_t（Unicode），(2)使用微软定义的 **统一字符串格式（BSTR）**。 前面四个字节放字符串的长度，中间放字符数据，结尾以"/0" 结尾（len data /0）。API：SysAllocString、SysFreeString等

```
#include <stdio.h>
#include "MyInterface.h"
#include <oleauto.h>

int main() {

  //统一字符串格式  len data \0
  //字符串格式 'Hello\0'  5"hello"
  BSTR bstr1 = SysAllocString(OLESTR("hello"));

  HMODULE hDll = LoadLibrary("SuperMath.dll");
  if (hDll == NULL)
    return 0;
```

(3)使用 _bstr_t 类（包含头文件 comdef.h）
(4)使用 CComBSTR 类（包含头文件 atlbase.h），其定义的类对象可以当作参数使用，操作如下：

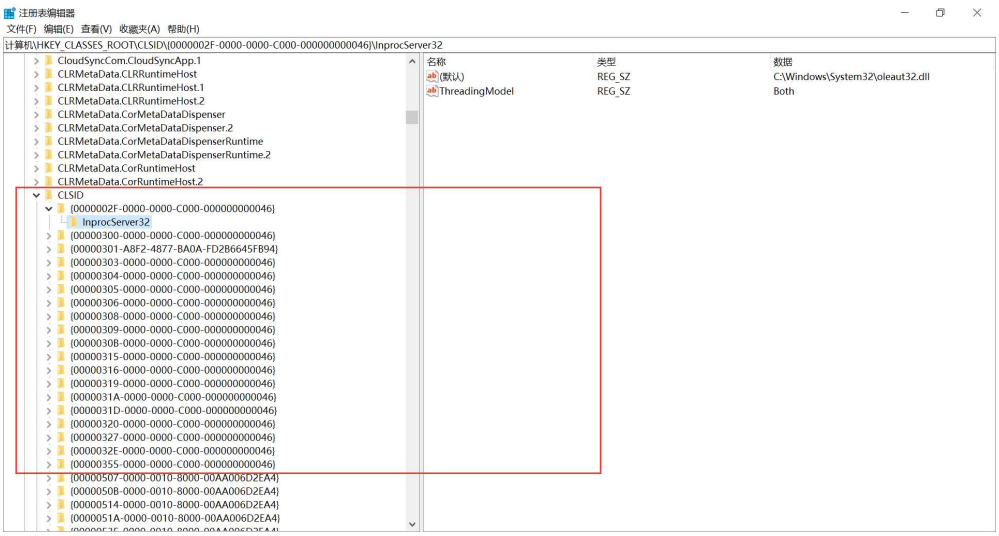
```
  //2.
  _bstr_t bstr2("hello");
  bstr1 = bstr2;

  //3.
  CComBSTR bstr3("hello");
  bstr1 = bstr3;

  ISuperString* pSuperString = NULL;
  pSuperString->Length(bstr3, NULL);  //转换运算符

  HMODULE hDll = LoadLibrary("SuperMath.dll");
  if (hDll == NULL)
    return 0;
```

- 插件路径问题：
  (1)路径定死
  (2)插件允许安装在任何位置：在注册表中记录插件的路径（由插件开发者将插件写入到注册表中），防止路径冲突使用GUID



regsvr32 -- 用来将插件写入到注册表中，必须要有一个写入注册表的导出函数，插件使用者可以调用这个插件。
标准插件需要有下面四个导出函数：
DllGetClassObject -- 获取类对象
DllCanUnloadNow -- 是否可以卸载
DllRedisterServer -- 安装
DllUnregisterServer -- 卸载

插件在注册表中的写入

做一个数组，数组内存放需要写于注册表中的内容。**有两种加载插件的方法。**

插件的使用代码封装

插件使用的步骤：

- 查询注册表
- 加载导出函数
- 产生类工厂
- 产生对象

每次使用插件前几步操作需要重复，所以可以对其进行封装，方便下次使用，代码实例：

```
// Use.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <stdio.h>
#include "MyInterface.h"
```

```
#include <oleauto.h>
#include <comdef.h>
#include <atlbase.h>

HRESULT __stdcall MyGetClassObject(const GUID& clsid, const GUID& IID,
void** ppObject) {
  char szKey[MAX_PATH];
  //{FE5955E8-B3C2-4a1d-8B46-B24CDF5F2808}
  wsprintf(szKey, "CLSID\\{%08X-%04X-%04X-%02X%02X-
%02X%02X%02X%02X%02X%02X%02X}\\InprocServer32",
    clsid.Data1, clsid.Data2, clsid.Data3,
    clsid.Data4[0], clsid.Data4[1], clsid.Data4[2], clsid.Data4[3],
clsid.Data4[4],
    clsid.Data4[5], clsid.Data4[6], clsid.Data4[7]);

  //1. 查询注册表
  char szPath[MAX_PATH];
  LONG nLen = sizeof(szPath);
  HKEY hKey;
  RegOpenKey(HKEY_CLASSES_ROOT, szKey, &hKey);
  RegQueryValue(hKey, NULL, szPath, &nLen);
  RegCloseKey(hKey);
  //2.加载
  HMODULE hDll = LoadLibrary(szPath);
  if (hDll == NULL)
    return S_FALSE;

  MY_GET_CLASS_OBJECT pfnGetClassObj =
(MY_GET_CLASS_OBJECT)GetProcAddress(hDll, "DllGetClassObject");
  if (pfnGetClassObj == NULL)
    return S_FALSE;

  return (*pfnGetClassObj)(clsid, IID, ppObject);
}

HRESULT __stdcall MyCreateObject(const GUID& clsid, const GUID& IID,
void** ppObject) {
  IMyFactory* pFactory = NULL;
  HRESULT hr = MyGetClassObject(clsid, IID_IMyFactory, (void**)&
pFactory);
  if (FAILED(hr)) {
    return hr;
  }

  hr = pFactory->CreateObj(IID, ppObject);

  pFactory->Release();
```

```
  return hr;
}

template<typename TYPE>
class CMyPtr {
public:
  CMyPtr(TYPE* pObject = NULL) {
    m_pObject = pObject;
  }
  ~CMyPtr() {
    if (m_pObject != NULL) {
      m_pObject->Release();
      m_pObject = NULL;
    }
  }
  TYPE* operator-> () {
    return m_pObject;
  }
private:
  TYPE* m_pObject;
};

int main() {

  //.TLB(typelib)   接口定义语言  ==>  编译 ==> TLB
  CMyPtr<ISuperMath> pSuperMath = NULL;
  HRESULT hr = MyCreateObject(CLSID_CSuperMath, IID_ISuperMath,
(void**)&pSuperMath);
  if (FAILED(hr)) {
    return 0;
  }

  long ret;
  pSuperMath->Add(1, 2, &ret);
  printf("1+2=%d\n", ret);

  //pSuperMath->Release();
  return 0;
}
```