您的潜力，我们的动力

Microsoft
微软（中国）有限公司

VC++ 2005：非托管互操作

李建忠

上海祝成科技 高级讲师

msdn

MSDN Webcasts

Agenda

- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

C++/CLI非托管互操作简介

- C++/CLI支持三个级别的代码重用，即：模块级的重用——P/Invoke，组件级的重用——COM互操作，源代码级的重用——C++ Interop。
- P/Invoke和COM互操作为.NET框架支持的两种互操作机制，可用于绝大多数.NET语言。
- C++ Interop为C++/CLI单独支持的一种更为灵活和强大的互操作机制。

Agenda

- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

P/Invoke 简介

- P/Invoke支持在.NET Managed代码中调用特定平台（如Windows)模块（DLL）中的Unmanaged函数。
- P/Invoke为.NET框架直接支持的平台互操作机制，为绝大多数.NET语言（包括C++/CLI，C#，Visual Basic .NET 等）所支持。
- P/Invoke适用于没有源代码而只有DLL文件的情况。

一个简单的P/Invoke示例


```
[DllImport("User32.dll")]
int MessageBoxA(int hWnd, String^ msg,
        String^ caption,  int type);

int main( )
{
   MessageBoxA(0,
          "Do you love C++/CLI?",
          "C++/CLI Interop",
          0);
}
```

使用P/Invoke 的几个步骤

- 使用DllImportAttribute特性指定unmanaged函数所在的模块，入口点、以及调用字符集等其他设置。
- 在DllImport特性之后指定unmanaged函数的签名，包括函数名称、返回值，参数。
- 由于许多unmanaged类型和managed类型的内存布局存在差异，因此在进行非托管互操作时，需要对这些类型进行转换，即Marshal。

基本类型的Marshal

非托管类型	托管类型	描述
void*	System.IntPtr	
unsigned char	System.Byte	
short	System.Int16	
unsigned short	System.UInt16	
int	System.Int32	
unsigned int	System.UInt32	
long	System.Int32	
unsigned long	System.UInt32	
char	System.Char	指定 ANSI
char*	System.String or System.StringBuilder	指定 ANSI
const char*	System.String or System.StringBuilder	指定 ANSI
wchar_t*	System.String or System.StringBuilder	指定 Unicode
const wchar_t*	System.String or System.StringBuilder	指定 Unicode
float	System.Single	
double	System.Double	

数组的Marshal


```
//dll中的非托管函数
extern "C“ __declspec(dllexport)
void myfunc(int list[], int length);


//managed 代码中声明的函数原型
[DllImport("myfunclib.dll")]
public void myfunc(
    [MarshalAs(UnmanagedType::LPArray)]array<int>^,
    int length);
```

自定义类型的Marshal

```
//dll中的自定义类型
typedef struct _MYSTRUCT {
   char* text;
   int data;
} MYSTRUCT;

//managed 代码中声明的自定义类型
[ StructLayout( LayoutKind::Sequential,
         CharSet=CharSet::Ansi )]
public value struct MyStruct {
   String^ text;
   int data;
};
```

代码演示（1）

P/Invoke互操作

Agenda

- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

COM互操作简介

- C++/CLI支持两种COM互操作：第一种为.NET 框架提供的Tlbimp.exe，适用于绝大多数.NET语言；第二种为C++/CLI特有的COM互操作机制。
- Tlbimp.exe需要将COM接口的所有成员导出到一个程序集wrapper中，其支持的数据类型也有限。
- C++/CLI所特有的COM互操作机制支持所有的数据类型，并且透明处理其中的Marshal。也无需导出所有成员。

使用Tlbimp.exe

- 执行Tlbimp.exe，并为其传递COM类型库文件，或者包含COM类型库的DLL文件。执行结果将得到一个托管的程序集Wrapper：
  - tlbimp myCOMLib1.tlb
  - tlbimp myCOMLib2.dll
- 引用Tlbimp.exe产生的托管程序集Wrapper，并使用其中的类型：
  - #using <myCOMLib1.dll>

使用C++/CLI COM Interop

- 使用C++/CLI COM Interop，可以将COM组件直接封装在C++/CLI类型内，进行无缝的访问，C++/CLI编译器会自动处理其中的转换工作。
- 这些封装类被称为CRCW（custom runtime callable wrappers），它们可以被其他.NET语言访问。
- CRCW不需要wrapper程序集，只需定义COM接口的头文件，这些头文件可用MIDL编译器生成。

代码演示（2）

COM互操作

Agenda

- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

C++ Interop简介

- 使用C++ Interop，可以将非托管的C++代码与托管的C++/CLI代码放在同一个文件中编译，互相之间进行无缝的访问。
- 生成文件为一个包含非托管机器指令和MSIL指令的混合程序集，它们之间的调用采用P/Invoke。
- C++/CLI Interop会透明地处理其中的类型Marshal，是最为灵活和高效的互操作方案。

C++ Interop机制解析

- 绝大多数非托管代码被编译为托管MSIL指令，并通过全局函数调用来完成互操作。
- C++ 本地类型被编译为value class，成员函数被编译为全局函数，数据成员则通过位操作来访问赋值。
- 通过MSIL和相关的元数据来实现C++本地对象模型，例如虚函数的调用就通过间接的函数指针来完成调用。

代码演示（3）

C++ Interop

Agenda

- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

讲座总结

- C++/CLI支持三个级别的重用，即：模块级的重用——P/Invoke，组件级的重用——COM互操作，源代码级的重用——C++ Interop。

- 互操作的关键在于类型的Marshal，P/Invoke，COM互操作，C++ Interop都需要类型Marshal。

- C++ Interop技术大大增强了.NET平台和其他技术的互操作能力，尤其支持源代码级的重用，使得互操作技术更为灵活和透明。

Agenda

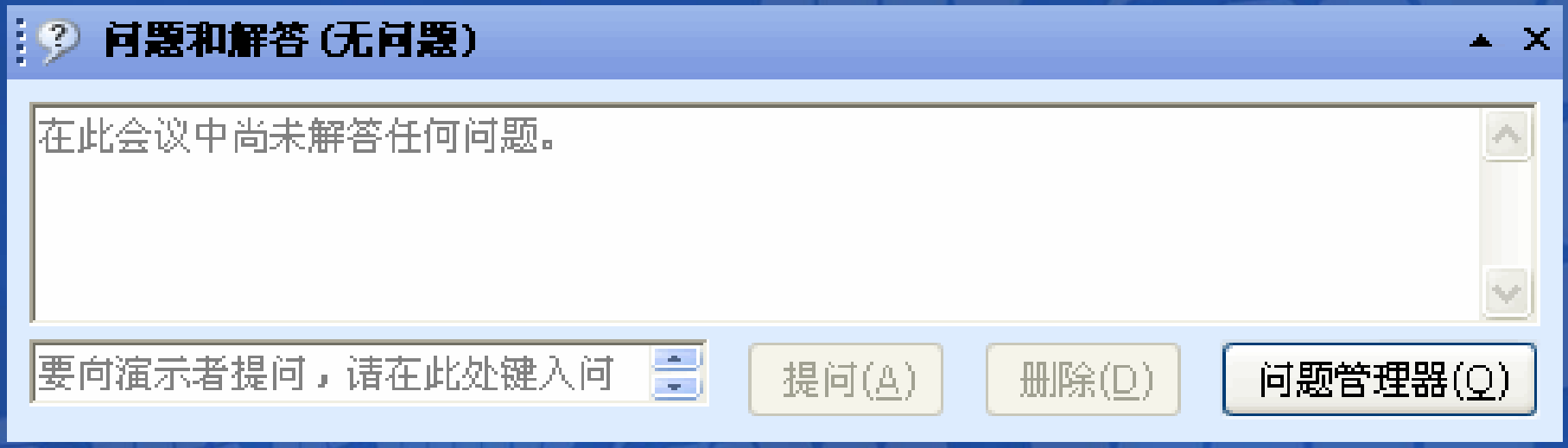
- C++/CLI非托管互操作简介
- 模块级的重用——P/Invoke
- 组件级的重用——COM互操作
- 源代码级的重用——C++ Interop
- 讲座总结
- Q&A

您的潜力，我们的动力

Microsoft
微软（中国）有限公司

Q&A

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。



msdn

MSDN Webcasts


您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts