DEV349
Visual C++ 2005
无缝集成，无限潜力

李建忠
微软特约讲师
上海祝成科技

VC++ 2005 集成潜力

- VC++ 2005的内核语言为C++/CLI，是目前.NET平台上能力最接近IL代码的系统级语言
- 无缝集成本地代码与托管代码，支持混合程序集
- 同时支持CLR泛型与ISO-C++模板
- 支持多种框架类库，Win32 , MFC , COM, STL, ATL, .NET 框架类库
- 高度优化的代码，是所有.NET语言中优化程度最高的代码
- C++/CLI不是ISO-C++与C#的简单相加，而是一门全新的语言！

C++ × CLR= Visual C++ 2005

C++ 技术特点：
- 静态化的对象模型
- 对象空间和生成文件的高度优化
- 确定性内存收集
- 特定平台目标编译
- 强大的静态模板
- 灵活的指针与引用
- 强大的STL, MFC, ATL

CLR 技术特点：
- 动态化的组件模型
- 丰富的元数据
- 自动垃圾收集
- JIT编译，跨平台
- 运行时泛型
- 安全的对象句柄，数组，委托（函数指针）
- 强大的.NET 框架类库

Visual C++ 2005

C++/CLI集成技术图谱

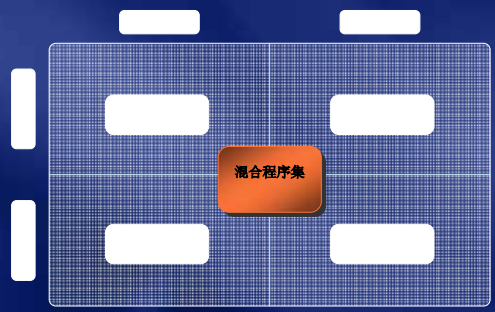
	技术	支持	特点
源代码集成	C++ Interop	只有C++/CLI支持	效率高，绝大部分本地代码被编译为IL代码
对象模型集成	C++ Interop	只有C++/CLI支持	效率高，绝大部分本地代码被编译为IL代码，但目前不够完善
函数库模块集成	P/Invoke	.NET框架支持，可用于所有.NET语言	非托管模块单独执行，需要类型Marshal
COM组件集成	COM互操作 C++ Interop	.NET框架支持，可用于所有.NET语言	COM组件单独执行，需要类型Marshal
泛型与模板集成	CLR泛型，C++模板	只有C++/CLI支持	泛型只能应用于托管类型，模板可应用于托管类型与本地类型，STL.NET

大型框架集成, STL.NET！ MFC.NET? Boost.NET ?

使用C++ Interop集成源代码

- 使用C++ Interop，可以将ISO-C++代码单独编译为托管代码，也可以将ISO-C++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。
- C++ Interop 技术保证了所有的ISO-C++代码经过cl/clr编译后行为保持不变。
- 绝大多数ISO-C++代码编译后将得到托管代码，即IL代码。部分不能编译为IL代码的采用P/Invoke调用实现，生成文件为一个包含非托管机器指令和IL指令的混合程序集。
- C++ 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。

混合程序集



demo

源代码集成Code Example

集成ISO-C++与CLI对象模型

- ISO-C++对象模型和CLI对象模型集成是C++/CLI集成技术中最为复杂，也最彰显潜力的地方。
- C++/CLI在对象模型集成过程中几个突出的问题
  - C++/CLI只支持对托管引用类型进行垃圾收集服务，不支持对ISO-C++本地类型的垃圾收集服务。
  - 垃圾收集导致了托管对象地址的不稳定，与ISO-C++本地对象稳定的地址形成鲜明对比。
  - C++/CLI中的托管对象的内存布局也和本地对象的内存布局有明显的不同。
  - C++/CLI中类型的多态机制（虚拟）也不同于本地类型的多态机制。

对象模型结构的集成（1）

在托管对象中包含本地对象的指针

```
ref class ManagedClass{
    string* pText;
};
```

在本地对象中包含托管对象的指针

```
class NativeClass {
    gcroot<String^> pText;
};
```

- 在托管对象中包含本地对象的指针，或者在本地对象中包含托管对象的指针，从而可“连通”托管世界和本地世界，实现两个代码库的相互复用。
- 在这个集成过程中，CLR垃圾收集只负责托管对象的收集工作，本地对象的回收工作仍由程序员自己负责delete。

对象模型结构的集成（2）

× 在托管对象中包含本地对象

```
ref class ManagedClass{
    string text;
};
```

× 在本地对象中包含托管对象

```
class NativeClass{
    String text;
};
```

- 在托管对象中包含本地对象，或者在本地对象中包含托管对象，相对于“包含指针”有相当技术难度，但成效与“包含指针”相差不大。
- 这两种集成技术在Visual C++ 2005中都尚未实现，但在Visual C++的未来版本中有望通过代理来实现。

对象模型结构的集成（3）

× 将本地对象放在托管堆中

```
string^ hText=
        gcnew string;
```

× 将托管对象放在本地堆中

```
String* pText =
        new String;
```

- 将本地对象放在托管堆中，或者将托管对象放在本地堆中，从而实现C++之父Bjarne Stroustrup所推崇的“可选的”垃圾收集，意义重大，但实现难度相当大。
- 这两种集成技术在Visual C++ 2005中都尚未实现，在Visual C++开发组中仍处于研究状态。

demo

对象模型集成Code Example

P/Invoke 平台调用

```
[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);
}
```

- P/Invoke支持在托管代码中调用特定平台（如Windows)模块（DLL）中的非托管函数。
- P/Invoke为.NET框架直接支持的平台互操作机制。适用于没有源代码而只有DLL文件的情况。
- 由于许多本地类型和托管类型的内存布局存在差异，P/Invoke需要对这些类型进行Marshal。

COM互操作

- C++/CLI支持两种COM互操作：第一种为.NET框架提供的TIbimp.exe，适用绝大多数.NET语言；第二种为C++/CLI特有的COM互操作机制。
- TIbimp.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。
  - 可以将COM组件直接封装在C++/CLI类型内，进行无缝访问，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，CRCW不需要wrapper程序集，只需定义COM接口的头文件。

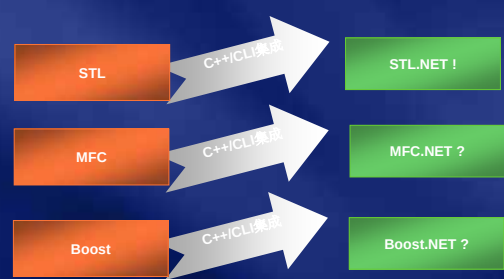
demo

COM互操作Code Example

泛型与模板集成

- C++/CLI支持三种泛型应用
  - 在ISO-C++本地类型上应用模板（编译时泛型）
  - 在CLI托管类型上应用模板（编译时泛型）
  - 在CLI托管类型上应用CLI泛型（运行时泛型）
- C++/CLI所支持的泛型程序库
  - 标准模板库 STL
  - CLI标准模板库 STL.NET
  - CLI泛型库 System::Collections::Generic
- C++模板采用基于“签名”的隐式约束，同时支持非类型参数、模板的模板参数、模板特化以及部分特化，具有极高的灵活性；而CLI泛型采用基于“基类+接口”的显式约束，丧失了很大的灵活性，难以实现一些高级的应用

C++/CLI集成展望：大型应用程序框架集成



社区资源

- 梦溪e谈：www.dreambrook.com 传说中的C++高端技术社区☺
- MSDN 中国： Visual C++ 开发中心
- blog.dreambrook.com/jzli 个人博客
- comp.lang.c++.moderated
- msdn.microsoft.com/visualc
- blogs.msdn.com/slippman
- pluralsight.com/blogs/hsutter

Microsoft®

您的潜力，我们的动力