

VC++ 2005：元数据与动态编程

李建忠

上海祝成科技 高级讲师

2005年6月8日

Agenda

- C++/CLI 动态编程概览
- CLI 元数据系统
- 动态编程(1)——反射 Reflection
- 动态编程(2)——特性 Attributes
- 讲座总结
- Q&A

动态编程简介

- 动态程序是指能够在运行时改变自身结构和行为的程序
- 静态——编译时，早约束，紧耦合
- 动态——运行时，迟约束，松耦合
- 动态 vs. 静态 效率vs.灵活性

ISO-C++与动态编程

- ISO-C++是一门静态编程语言，只具有非常有限的动态能力
 - 动态堆内存——虚函数的动态绑定
 - 有限的类型识别能力——RTTI
- 获得了极高的效率，却丧失了动态编程所具有的高灵活性
 - DLL动态链接库
 - COM 组件

C++/CLI与动态编程

- C++/CLI仍是一门静态语言，只不过相对于ISO-C++来讲，它具有了更强的动态编程特质
- 完备的元数据系统赋予了C++/CLI以相当的动态编程能力
- C++/CLI在付出较小性能代价的同时，获得了组件编程时代所需要的灵活性

基于元数据的动态编程

- C++/CLI 动态编程的两个主要突出表现
 - 反射 Reflection
 - 特性 Attributes
- 反射——动态地发现类型，操作类型，创建类型（查询元数据）
- 特性——允许程序定义新的元数据，从而在运行时动态地感知环境（创建元数据）

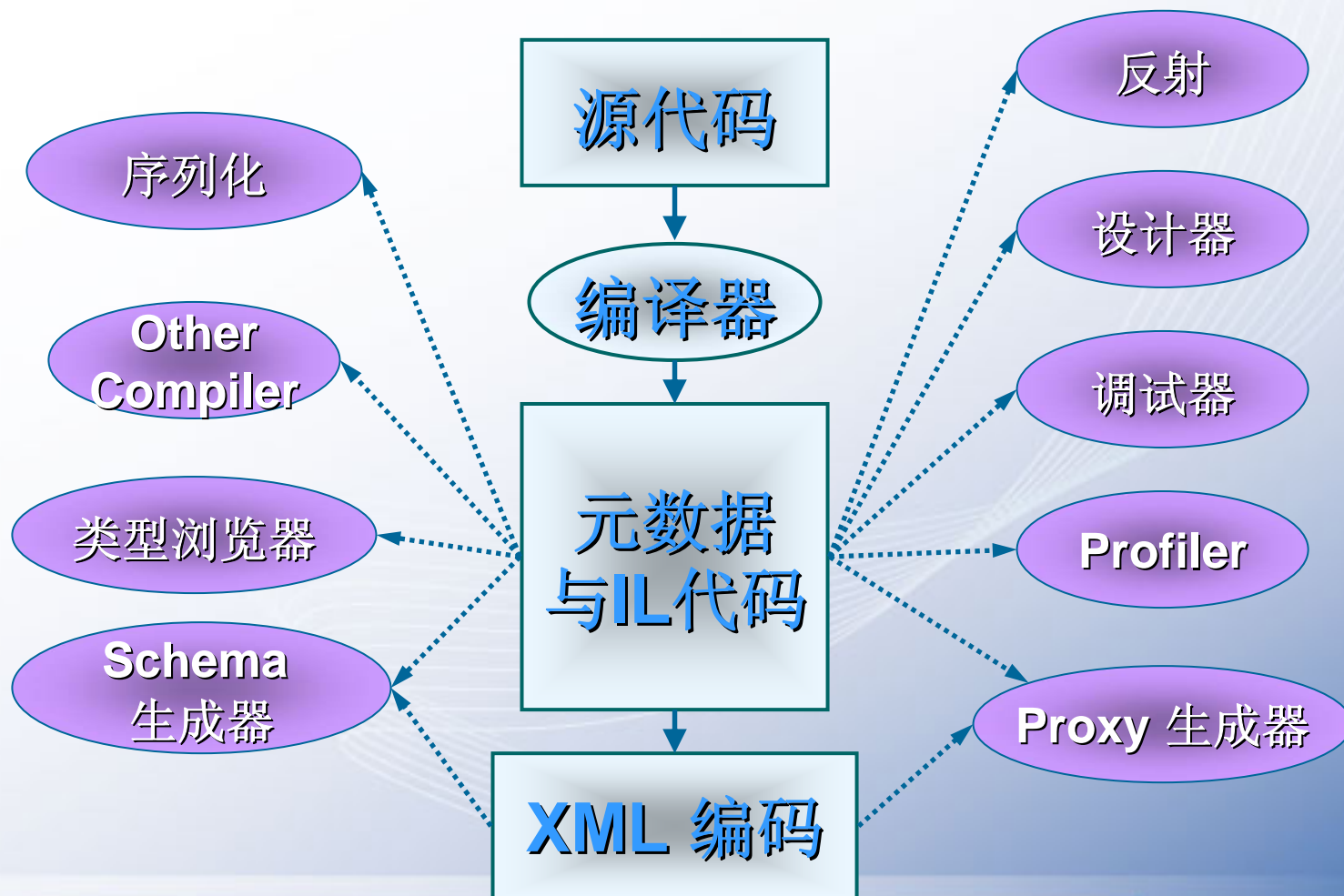
Agenda

- C++/CLI 动态编程概览
- CLI 元数据系统
- 动态编程(1)——反射 Reflection
- 动态编程(2)——特性 Attributes
- 讲座总结
- Q&A

什么是元数据

- 元数据（Metadata）是“数据的数据”
- 元数据是CLI组件合同的描述载体，组件平台的“粘合剂”
- CLI元数据分为三种：
 - 定义型元数据 ——描述代码中定义了什么
 - 引用型元数据 ——描述代码中引用了什么
 - 特性 Attributes——扩展定义新的元数据

元数据在CLI平台中的应用

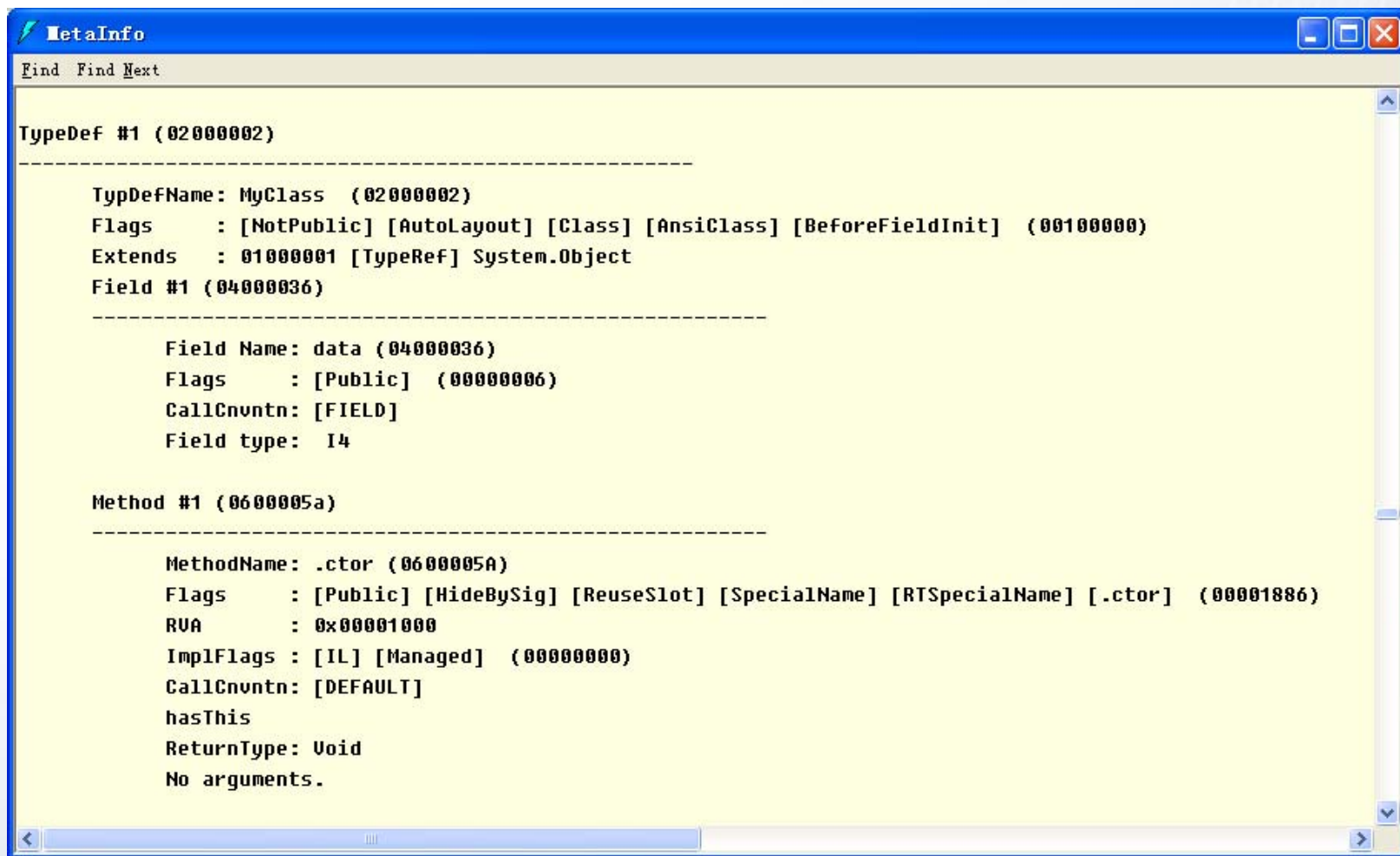


示例代码

```
ref class MyClass{  
public:  
    int data;  
};
```

```
int main() {  
    System::Console::WriteLine(  
        MyClass::typeid);  
}
```

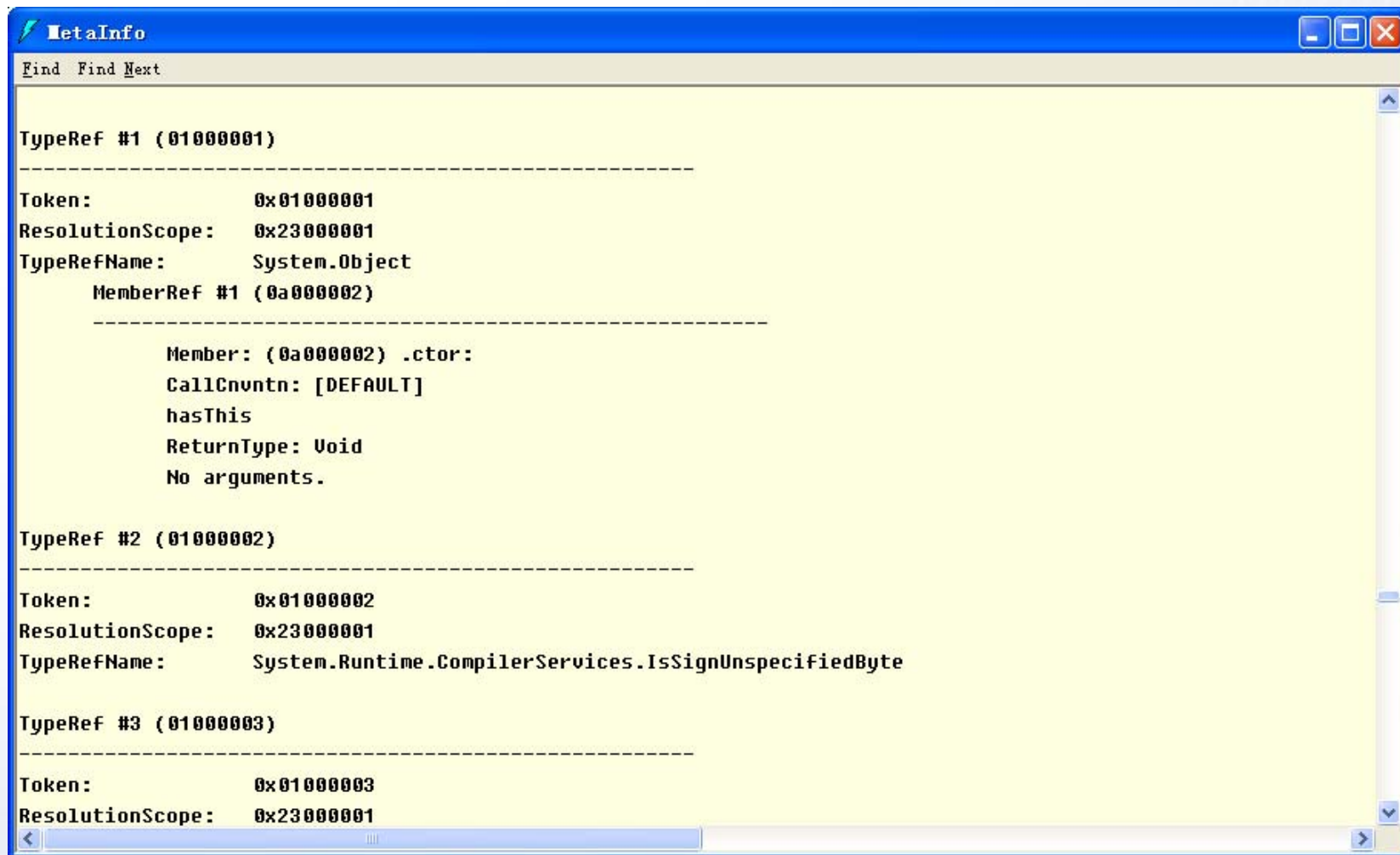
元数据示例 (1)



The screenshot shows a window titled "MetaInfo" with a menu bar containing "Find" and "Find Next". The main content area displays the following metadata:

```
TypeDef #1 (02000002)
-----
TypeDefName: MyClass (02000002)
Flags       : [NotPublic] [AutoLayout] [Class] [AnsiClass] [BeforeFieldInit] (00100000)
Extends     : 01000001 [TypeRef] System.Object
Field #1 (04000036)
-----
Field Name: data (04000036)
Flags      : [Public] (00000006)
CallConvntn: [FIELD]
Field type: I4
-----
Method #1 (0600005a)
-----
MethodName: .ctor (0600005a)
Flags      : [Public] [HideBySig] [ReuseSlot] [SpecialName] [RTSpecialName] [.ctor] (00001886)
RVA        : 0x00001000
ImplFlags  : [IL] [Managed] (00000000)
CallConvntn: [DEFAULT]
hasThis
ReturnType: Void
No arguments.
```

元数据示例 (2)



The screenshot shows a window titled "MetaInfo" with a menu bar containing "Find", "Find Next", and "Next". The main content area displays three type references, each separated by a dashed line. The first type reference is "TypeRef #1 (01000001)" and details a constructor for System.Object. The second is "TypeRef #2 (01000002)" for System.Runtime.CompilerServices.IsSignUnspecifiedByte. The third is "TypeRef #3 (01000003)".

```
MetaInfo
Find Find Next

TypeRef #1 (01000001)
-----
Token:           0x01000001
ResolutionScope: 0x23000001
TypeRefName:     System.Object
  MemberRef #1 (0a000002)
  -----
    Member: (0a000002) .ctor:
    CallConvntn: [DEFAULT]
    hasThis
    ReturnType: Void
    No arguments.

TypeRef #2 (01000002)
-----
Token:           0x01000002
ResolutionScope: 0x23000001
TypeRefName:     System.Runtime.CompilerServices.IsSignUnspecifiedByte

TypeRef #3 (01000003)
-----
Token:           0x01000003
ResolutionScope: 0x23000001
```

定义型元数据

- 模块定义—— ModuleDef
- 类型定义—— TypeDef
- 方法定义—— MethodDef
- 字段定义—— FieldDef
- 参数定义—— ParamDef
- 属性定义—— PropertyDef
- 事件定义—— EventDef

引用型元数据

- 程序集引用—— AssemblyRef
- 模块引用—— ModuleRef
- 类型引用—— TypeRef
- 成员引用—— MemberRef

Agenda

- C++/CLI 动态编程概览
- CLI 元数据系统
- 动态编程(1)——反射 Reflection
- 动态编程(2)——特性 Attributes
- 讲座总结
- Q&A

类型发现 (1)

```
Assembly^ a = Assembly::Load(args[1]);  
array<Type^>^ types = a->GetTypes();  
IEnumerator^ typeIter =  
    types->GetEnumerator();  
  
while (typeIter->MoveNext()){  
    Type^ t =  
        dynamic_cast<Type^>(typeIter->Current);  
    Console::WriteLine(" {0}",  
        t->ToString());  
}
```

类型发现 (2)

```
array<MemberInfo^>^ members =
```

```
    t->GetMembers();
```

```
IEnumerator^ memberIter =
```

```
    members->GetEnumerator();
```

```
while (memberIter->MoveNext()) {
```

```
    MemberInfo^ mi =
```

```
        dynamic_cast<MemberInfo^>
```

```
            (memberIter->Current);
```

```
    Console::Write("{0}", mi->ToString());
```

类型操作 (1)

```
ref struct A{  
    void Method(){  
        Console.WriteLine("A::Method()");}  
};
```

```
A^ objA = gcnew A;  
Type^ typeA = objA->GetType();  
MethodInfo^ methodA =  
    typeA->GetMethod( "Method" );  
methodA->Invoke( objA, nullptr );
```

类型操作 (2)

```
array<Type^>^ types =  
    plugAssembly->GetTypes();  
Type^ formType = Form::typeid;  
for (int i=0; i<types->Length; i++){  
    if (formType->IsAssignableFrom(types[i])){  
        Form^ f = dynamic_cast<Form^>  
            (Activator::CreateInstance(types[i]));  
        if (f) { f->Show(); }  
    }  
}
```

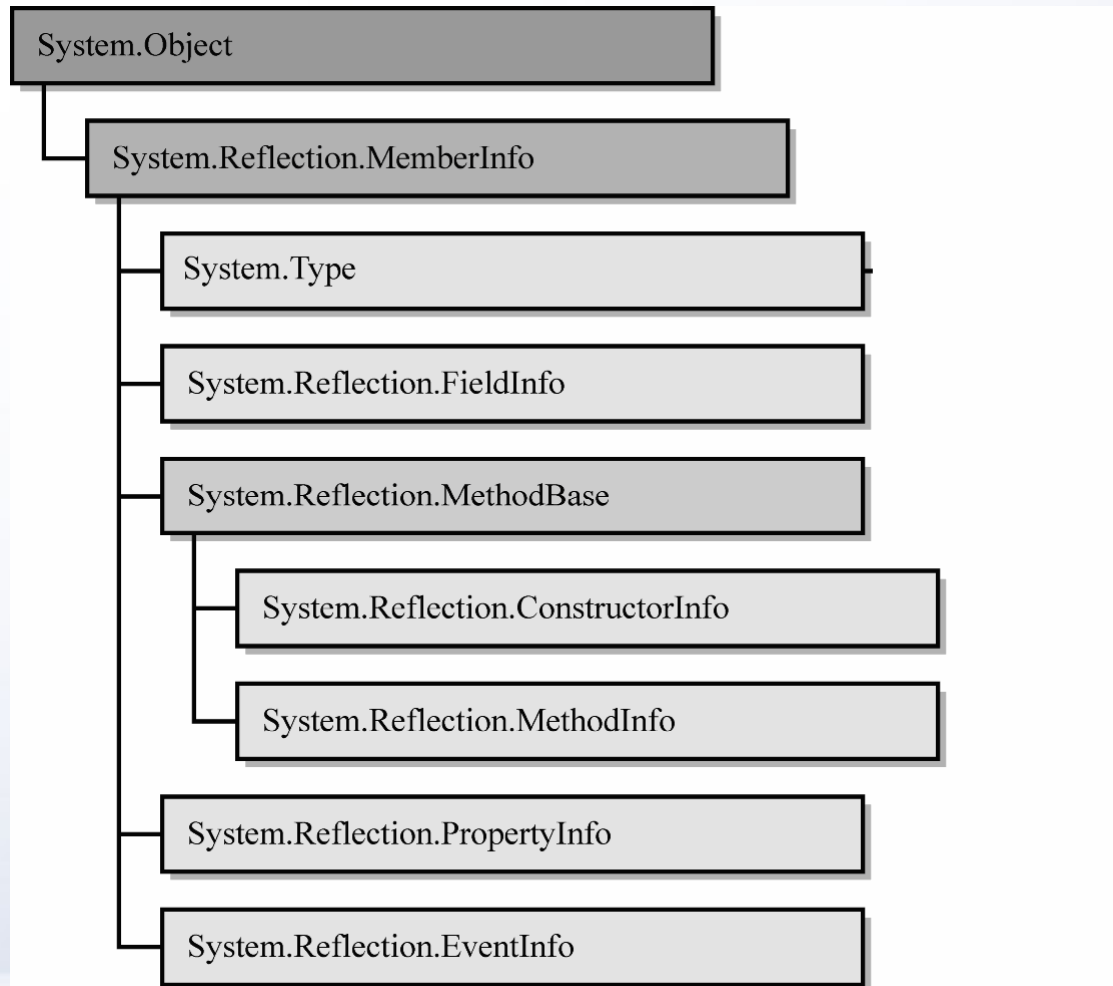
类型创建 (1)

```
TypeBuilder^ typeBuilder =  
    moduleBuilder->DefineType("MyType",  
        TypeAttributes::Public |  
        TypeAttributes::Class,  
        Object::typeid,  
        interfaceTypes);  
  
typeBuilder->AddInterfaceImplementation(  
    IMyInterface::typeid );
```

类型创建 (2)

```
ILGenerator^ ilGenerator =  
    methodbuilder->GetILGenerator();  
ilGenerator->Emit(OpCodes::Ldarg_1);  
...  
MethodInfo^ methodInfo =  
    typeid<IMyInterface^>-  
        >GetMethod("MyMethod");  
typeBuilder->DefineMethodOverride(  
    methodbuilder,  
    methodInfo);
```

反射类型层级结构



Agenda

- C++/CLI 动态编程概览
- CLI 元数据系统
- 动态编程(1)——反射 Reflection
- 动态编程(2)——特性 Attributes
- 讲座总结
- Q&A

特性 Attributes-定义

```
[AttributeUsage(AttributeTargets::Class,  
    AllowMultiple = true)]
```

```
public ref class AuthorAttribute : Attribute {  
    String^ name;
```

```
public:
```

```
    AuthorAttribute(String^ name) : name(name)  
    { }
```

```
    property String^ Name { String^ get()  
    { return name;} }
```

```
};
```

特性 Attributes-使用

```
[Author("Brian Kernighan")]  
[Author("Dennis Ritchie")]  
ref class CLang{  
    ...  
};
```

编译时实例化，运行时查询

特性 Attributes的应用

- 特性极大地扩展了对组件的描述能力，将元数据发挥到了极致，可以加速许多丰富的、创新的应用
 - 产生式编程
 - 声明性编程
 - 面向方面编程AOP
 - 模型驱动体系架构MDA

Agenda

- C++/CLI 动态编程概览
- CLI 元数据系统
- 动态编程(1)——反射 Reflection
- 动态编程(2)——特性 Attributes
- 讲座总结
- Q&A

讲座总结

- C++/CLI仍是一门静态编程语言，仍然具有静态编程语言的高效率，只是具有了更强的动态特质
- 元数据系统决定了C++/CLI的动态编程能力，是CLI平台的“血液”
- 反射和特性是C++/CLI动态编程的具体体现，也是C++/CLI极具活力的地方

Agenda

- C++/CLI 动态编程概览
 - CLI 元数据系统
 - 动态编程(1)——反射 Reflection
 - 动态编程(2)——特性 Attributes
 - 讲座总结
- Q&A

Q & A

资源链接

- msdn.microsoft.com/visualc
- comp.lang.c++.moderated
- blogs.msdn.com/slippman
- pluralsight.com/blogs/hsutter
- blog.joycode.com/lijianzhong
- www.chinaitclub.org/forums/

Microsoft[®]