第32卷 第16期 计 算 机 工 程 2006年8月
Vol.32 № 16 Computer Engineering August 2006
·软件技术与数据库· 文章编号：1000—3428(2006)16—0095—02 文献标识码：A 中图分类号：TP311

C++反射机制的一种简单实现

鲍 亮，陈 平
（西安电子科技大学软件工程研究所，西安 710071）

摘 要：讨论了C++反射机制的实现问题，介绍了反射机制的概念和分类，比较了向C++添加反射机制的可能性和方式，提出并分析了一种基于宏定义、模板和泛型机制的C++反射机制实现手段——“简单C++反射机制（Simple C++ Reflection SCR）”。

关键词：反射；宏定义；模板；简单C++反射

A Simple Implementation of C++ Reflection

BAO Liang, CHEN Ping
(Software Engineering Institute, Xidian University, Xi'an 710071)

【Abstract】 This paper is focus on the implementation of C++ reflection mechanism. The definition and classification of reflection is simply introduced first. Then, the possibility and methods on how to add reflection mechanism to C++ programming language are discussed. Finally, simple C++ reflection (SCR), a macro definition, template and generic programming based solution, is discussed.

【Key words】 Reflection; Marco definition; Template; Simple C++ reflection(SCR)

1 反射技术简介

1.1 反射的概念

反射概念来自人工智能。反射结构分为结构反射和计算反射两类。结构反射的概念澄清了元类（Metaclass）与类之间的关系，计算反射则在计算层次上对与应用领域有关的目标计算和系统自身有关的反射计算进行了划分，分别由对象和元对象（Metaobject）的行为来体现。

本文采用的反射定义，主要侧重于在执行过程中对表示程序状态数据的操纵能力：“反射是程序在其自身执行过程中操纵那些表示程序状态的数据的能力。这种操纵表现在两个方面：内省（Introspection）和调解（Intercession）。内省是程序观察进而解释自身状态的能力。调解是程序修改其执行状态或修改自身含义的能力。操纵这两个方面都需要有一种将执行状态编码为数据的机制，实现这种编码的过程叫做具化(Reification)[1]。”

1.2 反射模型

针对不同的反射定义和关心侧面，存在多种不同的反射结构，形成了多种计算反射模型。现有的反射模型分为结构反射（Structural Reflection）模型和计算反射（Computational Reflection）模型两大类[2]。本文主要从实现层面来讨论结构反射模型：实现对类型信息的查询和对方法的动态调用。

1.3 为什么需要反射机制

最近几年，由于在Java和.NET的成功应用，反射技术以其明确分离描述系统自身结构、行为的信息与系统所处理的信息，建立可动态操纵的因果关联以动态调整系统行为的良好特征，已经从理论和技术研究走向实用化，使得动态获取和调整系统行为具备了坚实的基础。

当需要编写扩展性较强的代码、处理在程序设计时并不确定的对象时，反射机制会展示其威力，这样的场合主要有：(1)序列化（Serialization）和数据绑定（Data Binding）。(2)远程方法调用（Remote Method Invocation RMI）。(3)对象/关系数据映射（E/R Mapping）。

当前许多流行的框架和工具，例如Castor(基于Java的数据绑定工具)、Hibernate(基于Java的对象/关系映射框架)等，其核心都是使用了反射机制来动态获得类型信息。因此，能够动态获取并操纵类型信息，已经成为现代软件的标志之一。

2 C++中添加反射机制的可能方式

针对C++的语言特点，可以考虑从表1中的几个方面入手添加反射机制。

表1 C++中添加反射机制的可能方式

添加方式	优点	缺点
解析编译器生成的调试信息	(1)程序不需要改变 (2)能够从程序中提取完整的类型信息	(1)程序必须以调试的形式进行编译，在发布时带有调试信息 (2)调试信息的格式与具体的编译器相关
预处理方式：解析C++程序，创建类描述信息	程序不需要改变	(1)不同的编译器会以不同的方式处理类成员变量的对齐、偏移，虚函数表的格式以及命名规则等问题。不可能实现统一的预处理器，直接获取类型信息 (2)解析C++代码比较复杂。除此之外，这种工具还必须完全能够处理C++预编译指令，并能够被普通的C++编译器调用
构造支持反射机制的C++编译器	(1)不需要在编译时提供额外的类型信息 (2)编译器能够针对反射机制提供优化	(1)实现一个C++编译器是一个十分复杂的庞大工程 (2)现代的C++编译器一般都与集成开发环境绑定在一起，说服用户放弃使用已经习惯的开发环境比较困难
由程序员在代码中添加需要的信息	(1)适用于任何C++编译器，移植性最强 (2)实现方式最简单 (3)根据用户需要提供额外的类型信息	(1)程序员需要修改代码 (2)在手工添加类型信息时，可能会出现错误

3 简单C++反射的设计和实现

3.1 简介

由于实现一个C++预编译器或支持反射机制的C++编译

基金项目：“十五”军事电子预研重点课题项目
作者简介：鲍 亮(1981—)，男，硕士生，主研方向：面向对象技术；陈 平，教授、博导
收稿日期：2005-11-20 **E-mail**：liangbaobox@hotmail.com

器是十分庞大和复杂的工程。即使能够完成，这种方法也不能解决所有的问题。鉴于此，作者提出了一种基于手工添加类型信息描述的解决方案："简单 C++反射（Simple C++ Reflection，SCR)"。

SCR 的核心是定义一组描述类型信息的辅助类，在程序中手工添加一系列宏定义，用来描述类型信息，实例化相关的辅助描述类，在运行时就可以使用辅助描述类来实现反射的功能。

假设用户定义了一个二维点的类型，重载了"+"操作符：

```
Class Point2D{
public:
 int x;
 int y;
 Point2D()
void SetPoint(int _x, int _y);
 Point2D& operator +(const Point2D& pt);
TYPE_INFO(Point2D,
FIELD_INFO(x, SCR_PUBLIC),
FIELD_INFO(y, SCR_PUBLIC),
METHOD_INFO(Point2D, SCR_PUBLIC),
METHOD_INFO(SetPoint, SCR_PUBLIC ),
METHOD_INFO(+,SCR_PUBLIC)))
};
```

可以看到，上面这个类定义，除了正常的字段和方法定义外，添加了一些复杂的宏定义，正是通过这些宏定义，手工将所关心的类信息、类型信息和方法信息保存下来。

3.2 描述类信息

为了保存类型信息，首先需要定义描述类型信息的一组类。在这里，作者参考了.NET 中与反射机制相关的类定义结构，设计了如图 1 所示的类结构。

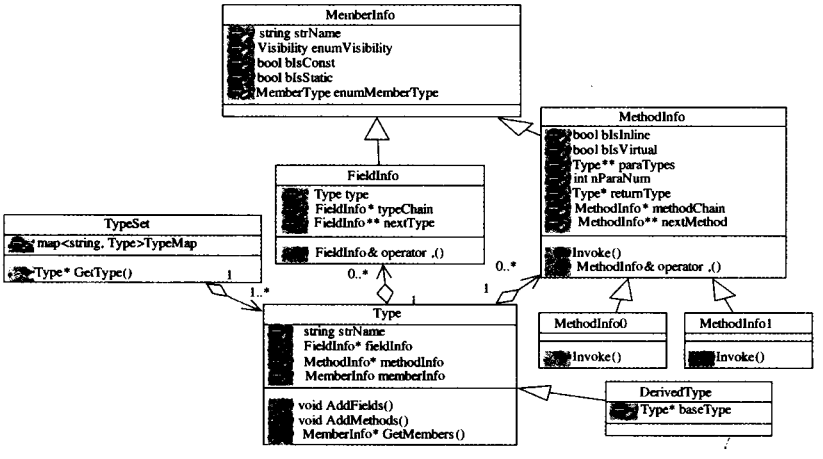

图 1 SCR 的类体系结构

各个类的简单介绍：

MemberInfo：类成员信息类型的抽象基类。FieldInfo：类字段信息的描述类。MethodInfo：类方法信息的描述类。MethodInfoX：MethodInfo 的子类，X 表示参数个数。Type：类型的描述类，反射操作的根。DerivedType：Type 类的子类，继承其他类的子类描述类。TypeSet：Type 类集合，保存所有类型信息。

3.3 信息收集和使用

C++是静态编译语言，所有的信息收集工作必须在编译时完成。可以利用编译预处理功能，使用宏定义在程序的合适位置插入描述类型信息的代码。这些宏定义分为以下几种：

3.3.1 字段信息描述 FIELD_INFO(字段名称，标志位)

```
#define FIELD_INFO(fieldname,flag)*new FieldInfo(#fieldname,
(void*)&fieldname-(void*)&this, sizeof(fieldname), flag, new Type
(typeid(fieldname).name()), 1)
```

宏定义展开后实际上是 FieldInfo 类的构造函数的调用，通过该函数创建一个字段的名称、地址(与起始位置的偏移)、大小、类型和字段性质的标志。

字段性质的标志包括能见度、是否是静态成员和是否是常量等信息。

宏定义的最后一项采用了 C++内建的 RTTI 机制[3]，获得了一个对象的类型名称，并以类型名称为参数，创建了该字段的类型信息。

3.3.2 方法信息描述 METHOD_INFO（方法名称，标志位）

```
#define METHOD_INFO(methodname, flag) *new MethodInfo
(#methodname, flag, MethodType(&NewType::methodname));
```

方法信息的描述涉及到方法名称、参数和返回值的描述。由于参数类型和返回值不确定，必须使用 C++的模板和泛型机制来完成这项工作[4]。

(1)使用仿函数(Fuctor)定义函数原型

为了满足方法参数和返回值的抽象性要求，需要利用 C++函数模板的机制，预先定义一组接收不同参数的函数原型模板，并使用 MethodInfo 的子类进行包装，举例说明如下：

具有一个参数的函数原型定义

```
template<class RT, classType, class ParaType1> class MethodInfo1:
public MethodInfo{
    typedef RT (C::*fptr)( ParaType1); //定义函数原型，参数类型是
ParaType1
    fptr method;
    MethodInfo1(fptr method){
        this->method = method; //得到函数指针
        methodType = TypeSet::GetType(typeid(C).
name());//方法所属类型
        returnType = TypeSet::GetType(typeid(RT).
name()); //返回值类型
        nParaNum = 1; //参数个数：1
        paraTypes = new Type*[1]; //分配空间
        paraTypes[0] = new Type(typeid(ParaType1).n
ame()); //参数类型：ParaType1 }
    };
```

使用类似的方式，可以定义具有任意多个参数的函数原型。在使用时，定义并重载 MethodType 函数，让编译器自动匹配合适的函数原型定义：

```
template<class C, class P1> inline MethodInfo*
MethodType(void (C::*f)(P1)){
    return new MethodInfo1<C, P1>(f);  }
```

编译时，对函数模板进行特化(specification)处理，自动匹配合适的函数定义，生成方法描述信息。

(2)重载 Invoke 函数实现动态调用

MethodInfo 定义了统一的调用函数 Invoke：

```
void Invoke(void result, void* obj, void*paras[])
```

参数 1 用来存放返回值。参数 2 是具体的调用对象，参数 3 是函数参数列表。

不同的 MethodInfo 子类分别对这个函数进行改写：

(下转第 99 页)

表 2 两个算法服务选择的比较

	S1	S2	S3	S4	S5	Total Time	Total Cost
时间最小算法	(3)	(2)	(1)	(1)	(1)	14	63
本文算法	(3)	(1)	(1)	(1)	(2)	14	51

上述例子是在最好情况下的一个例子，它的费用降低了12个单位，总的执行时间未变。这说明扩展关键活动算法能够保证时间尽可能短的情况下，给用户带来更低的费用。

3 结论

计算经济驱动的资源管理模型对工作流的调度提出了挑战。本文从工作流的执行特点出发，根据网格动态、自治的特性，提出了动态服务选择下基于扩展关键活动的费用和时间的调度算法，符合网格的实际情况，在保证工作流执行时间尽可能短的情况下优化了费用，能够更好地满足用户对于费用和时间要求。然而实验效果是在保证服务正常工作的情况下给出的，保证算法在动态、复杂的网格环境中成功执行，各种容错考虑是未来的工作[8]。

参考文献

1 Foster I, Kesselman K. The Anatomy of the Grid: Enabling Scalable Virtual Organizations[J]. International Journal of High Performance Computing Applications, 2001, 15(3): 200-222.

2 Foster I, Kesselman C. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration[EB/OL]. http://www.globus.org/reserch/papers/ogsa.pdf, 2002.

3 Deelman1 E, Blythe1 J. Mapping Abstract Complex Workflows onto Grid Environments[J]. Journal of Grid Computing, 2003, 1(1): 25-39.

4 Martino V D. Scheduling in Grid Computing Environment Using Genetic Algorithms[C]. Proc. of the 16th Int'l Parallel and Distributed Processing Symp., Florida, USA, 2002.

5 Abraham A, Buyya R. Nature's Heuristics for Scheduling Jobs on Computational Grids[C]. Proc. of the 8th Int'l Conf. on Advanced Computing and Communication, Cochin, India, 2000

6 Buyya R, Abramson D, Giddy J. An Economy Driven Resource Management Architecture for Global Computational Power Grids[C]. Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas, USA: CSREA Press, 2000: 517-525.

7 Buyya R, Giddy J, Abramson D. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications[C]. Proceedings of the 2nd International Workshop on Active Middleware Services. Pittsburgh, USA: Kluwer Academic Press, 2000: 221-230.

8 Sample N, Keyani P, Wiederhold G. Scheduling Under Uncertainty: Planning for the Ubiquitous Grid[C]. Proceedings of the 5th International Conference on Coordination Models and Languages, 2002.

9 Frey J, Tannenbaum T. Condor-G: A Computation Management Agent for Multi-institutional Grids[J]. Cluster Computing, 2002, 5(3): 237-246.

10 He Xiaoshan, Sun Xianhe. QoS Guided Min-Min Heuristic for Grid Task Scheduling[J]. Journal of Computer Science and Technical, 2003, 18(4): 442-451.

11 Singh G, Kesselman C, Deelman E. Optimizing Grid-based Workflow Execution[R]. CS 05-851, University of Southern California, 2005.

(上接第 96 页)

```
void MethodInfo1::Invoke(void* result, void* obj, void params[])
{
 *(RT*)result = ((C*)obj)->*method)(*(ParaType1*)params[0]);
//一个参数 }
```

这样，就实现了函数的动态调用。

3.3.3 类型信息描述 TYPE_INFO(类型名称，字段信息，方法信息)

```
#define TYPE_INFO(T, fields, methods) \
static Type* ptype = new Type(#T);\
ptype->AddFields(&fields);\
ptype->AddMethods(&method);\
TypeSet::AddType(ptype);\
typedef T NewType;
```

为了支持多个字段和方法的定义，需要分别在 FieldInfo 和 MethodInfo 中分别维护一个 FieldInfo 和 MethodInfo 的对象链表，并且重载“,”运算符，将所有信息收集起来。

3.3.4 继承信息描述 BASE_TYPE_INFO(类型名称，标志位)

```
#define BASE_TYPE_INFO(classname, flag) *new FieldInfo(#classname, (void*)(classname)this - (void*)this, sizeof(calssname), flag, new DerivedType(#classname));
```

根据 C++对象模型定义，将对象中基类对象的那部分描述为一个字段，并生成一个新继承类，设置其基类。

3.3.5 类型信息的使用

编译得到的代码，就可以实现动态调用。调用的主要方式有：

(1)由类型名称得到类型信息

```
Type *pType = TypeSet::("Point2D"); //得到类型描述
```

(2)由类型描述得到成员信息

```
MemberInfo* pMemberInfo = pType->GetMembers();//得到成员
//信息
pMemberInfo->GetMemberName();//得到成员名称
```

(3)得到字段描述信息

```
FieldInfo* pFieldInfox = pType->GetField("x");//得到字段信息
```

(4)得到方法描述信息

```
MethodInfo* pMethodPlus = pType->GetMethod("+");//得到方法
//信息
```

(5)方法的动态调用

```
//前面已经设置 p1 和 p2
Point2D result;
pMethodPlus->Invoke(&result, p1, p2);//result = p1->+(p2)。
```

3.4 小结

SCR 通过宏定义在程序中植入信息，并利用泛型机制，为 C++提供简单的反射机制。但是，这种方法还有局限性，主要体现在：(1)不支持枚举类型、联合类型和比特域。(2)支持的字段类型是有限的（包括类、结构、指针等）。(3)不能收集全局和静态变量、函数的信息。(4)不支持函数重载。

参考文献

1 Ducasse S. Reflective Programming and Open Implementations[D]. Bern: Paper University, 2001.

2 陈 平. 反射结构和对象标识的研究[D]. 西安: 西安电子科技大学, 1991.

3 Lippan S B, Lajoie J. C++ Primer[M]. Addison-Wesley, 2002.

4 Alexandrescu A. Moden C++ Design: Generic Programming and Design Pattern Applied[M]. Addison-Wesley, 2001.