C++语言 程序设计 面向目标

计算机科学1992Vol.19№.2

基于C++的良好OOP风格法则

15-19

韋远阳 杨芙清 方 裕 （100871北京大学计算机科学技术系） TP311.11

摘 要

From the property of good style OOP(Object-Oriented Programming), we propose the good style OOP law suitable to C++ language—Demeter/C++ law, and give out the proof of its transformation rules and show an example to specify it.

一、前 言

所谓“程序设计风格”，通常是指由编译器对程序的源代码进行静态分析时所能确认的特性。目前，关于OOP风格有两种理解：

1.采用非面向对象程序设计语言(OOPL)进行程序设计，使程序呈现出一定的OOP风格。

2.采用OOPL遵循一定的规范或法则进行OOP，使程序呈现出良好的OOP风格。

在本文我们关注后者，在Demeter法则[3]的基础上，以C++为背景，提出了一个采用C++进行良好风格OOP时所遵循的法则：Demeter/C++法则，并给出相应的证明和示例说明。

二、OOP风格

既然OOP被认为是八十年代的结构化程序设计，那么相应于结构化程序设计良好风格的特征之一——少用goto，什么是OOP良好风格的特征呢？

面向对象（OO）思想的两个核心概念是封装和继承，它们分别在“实例——类”和“子类——父类”两个层次上强调了模块（实例、类）的可重用性和可维护性，其关键是尽量降低模块之间的耦合度。我们认为OOPL支持OO思想，只是表明它仅提供了方便地（比较容易、安全和有效地）反映这一思想的机制，但并不表明用OOPL编制的程序就一定能充分地达到OO思想的上述目的，这不仅和语言设施有关，更与程序设计方法有关。研究表明：不加约束地使用OOPL中的OO机制时，生成的系统仍然不能满足OO方法的初衷，如继承使用不当就会破坏封装性[4]。因此，良好OOP风格的特征应当是以更好地降低模块间耦合度，增强模块的可重用性为目标。

在OO程序中，模块间的耦合反映在类对象之间方法的相互调用上，因此在程序设计时，应对方法中能访问的对象集合（通过发送消息调用其方法或变量）作一定的限制，并尽量使该集合为最小，使方法对周围环境对象的依赖性（耦合度）最小，相应降低其所属类对象和环境之间的耦合度，这一要求以程序设计法则的形式来表达，被称为Demeter法则[3]：**一个类的方法除了能访问自身类结构的直接子结构（即表层子结构）以外，不应以任何方式依赖于任何其它类的结构，并且每个方法只应对某个有限类集合中的相应对象发送消息。**

我们认为Demeter法则在下述几个方面是支持软件工程的有关原则的：

1.耦合度控制 耦合在OOP中的反映就是对象之间方法的调用/返回关系，Demeter

法则有效地减少了在某个方法内所能调用的方法数量，相应地增强了该方法的可重用性和软件的抽象层次。

2.信息隐藏 在Demeter法则中体现为类对象的结构信息隐藏。如果我们按"part-of"关系将复合结构对象描述成"组成树(composite tree)"，则Demeter法则不允许方法直接访问组成树中的深层子结构，只允许访问其表层子结构。

3.信息受控访问 在某些OOPL中（如C++[1]），可以将信息说明为私有的(private)，以保证信息的受控访问，而Demeter法则的着眼点是针对那些公有信息，在设计方法学上提供一组法则，使用户能按某种受控的方式访问它们，而不是在程序中通过语言设施说明将其隐藏起来。

4.信息局部化 Demeter法则要求一个方法仅应与其紧密联系的类对象有关（同时限制了方法中发送消息的消息名），而与系统的其它部分无关，实现了类信息和消息名信息的局部化。

5.减小模块界面 Demeter法则从程序设计方法的角度限制了方法所能访问的信息，从而减小了模块界面，这是因为，如果方法得到过多的信息，它就有可能会破坏这些信息。

6.Demeter法则强调一个实体的语义只应取决于其表层子结构 因此遵循这一法则的OOP程序有助于采用指称语义的合成定律（compositionality principle）进行结构化的推导验证。

三、Demeter法则

上述Demeter法则（称为L_0）只是根据OOP良好风格的特征描述的抽象、原则形式，具体运用时根据法则描述的层次（类、对象）不同，相应具有不同的表达形式，如图1所示。

图1 Demeter法则的不同表达形式

L_1——类表达形式：

对于类C中的任何方法M，M中发送任何消息的接收对象必须属于下述类之一：

1.类C本身。

2.方法M的参量类和返回对象类。

3.类C的实例变量所属的类（成员类）。

4.对类C显式说明是可见的类。

5.情形2中相应类的实例变量所属的类（成员类）。

6.方法M中能直接引用的实例变量只允许是上述情形1，2中类对象的实例变量。

L_2——对象表达形式：

对于类C中的任何方法M，M中发送任何消息的接收对象必须是下述对象之一：

1.对象self本身。

2.方法M的参量对象和返回对象。

3.全局变量中包含的、或者方法M中生成的，和情形1，2中相应对象是同类的对象。

4.情形1，2中相应对象所包含的实例变量对象(成员对象)。

5.全局变量中包含的、或者方法M中生成的，和对类C显式说明可见的类是同类的对象。

6.方法M中能直接引用的实例变量只允许是上述情形1，2，3中相应对象的实例变量。

就Demeter法则的两种表达形式，我们认为：法则的对象表达形式要求判定方法M中的消息接收者是否属于一个有限的对象集合，即要判定两个对象是否相等，其检测粒度是在对象层次；而法则的类表达形式要求判定方法M中的消息接收者是否属于一个有限的类集合，即要判定对象是否属于某个类，

其检测粒度是在类层次。既然“程序设计风格”是编译时的静态特性，而在编译时又难以判定对象的相等问题，这只能推迟到运行时刻解决，并且开销很大。因此，对象表达形式适用于无类型的OOPL（如Smalltalk），而对于强类型的OOPL（如C++），则类表达形式比对象表达形式更有效，因为这类语言能在编译时全部确认对象的类型信息。

正如结构化程序设计的良好风格法则并非对传统程序设计“包医百病”一样，Demeter法则也并不能解决OOP的所有“异常”，但对大部分OOP程序进行编译时风格的“规范化”依然是有效的。

四、Demeter/C++法则

1.Demeter/C++法则

上述法则是独立于具体OOPL的，我们将其进一步推广到C++[1]上，即给出Demeter法则的C++语言概念解释，以支持C++的良好风格OOP。考虑到C++的强类型性，因此我们选择法则的类表达形式，从而得到了Demeter/C++法则L_3：

对于类C中的任何成员函数M，M中能直接访问或引用的对象必须属于下述类之一：

1.类C本身。

2.成员函数M的参量类和返回对象类。

3.类C的数据成员所属的类（成员类）。

4.对类C显式说明是可见的类（可见度控制）。

5.情形2中相应类的数据成员所属的类（成员类）。

6.方法M中能直接引用的数据成员（无论其是否是类对象）只允许是上述情形1，2中相应类对象的数据成员。

下面我们通过构造性的定理证明来说明对C++程序进行良好风格化的转换过程。

2.证明

Demeter法则的实质是对方法所能访问的类模块范围进行限制，这种限制体现在：

1.若访问的是结构信息，则只允许调用相应类的表层子结构而非其深层子结构，但对结构信息的嵌套访问将会破坏这一限定。

2.若调用的是方法，则只允许对所限定的对象发送消息，当发生方法的嵌套调用时，这一限定常被破坏，因为如果不引入中间参量，则嵌套方法的运行返回结果一般不一定在法则所限定的范围内。

因此，按C++的概念说，破坏Demeter法则的根源在嵌套的成员表达式，而转换的实质就是将其“展开”成“平坦”的成员表达式。

定理 任何C++程序都能转换成遵循Demeter/C++法则的等价形式。

证明：设C++程序中出现的嵌套成员表达式为$a_0\rightarrow m_0\rightarrow m_1\rightarrow m_2\rightarrow m_3$（在此略去有关参量的表示），对象$a_0$的类是$A_0$，$a_0\rightarrow m_0$的返回结果类是$A_1$，$a_0\rightarrow m_0\rightarrow m_1$的返回结果类是$A_2$，$a_0\rightarrow m_0\rightarrow m_1\rightarrow m_2$的返回结果类是$A_3$，即：

class A_0{…; m_0; …};

class A_1{…; m_1; …};

class A_2{…; m_2; …};

class A_3{…; m_3; …};

1.设m是数据成员，则转换规则为：

class A_0{…; m_0; …}; ⟹class A_0{

…;

m_0;

m_0'{this→m_0→m_1'; }

…; };

class A_1{…; m_1; …}; ⟹class A_1{

…;

m_1;

m_1'{this→m_1→m_2'; }

…; };

class A_2{…; m_2; …}; ⟹class A_2{

…;

m_2;

m_2'{this→m_3→m_3; }

…; };

相应地将 $a_0\rightarrow m_0\rightarrow m_1\rightarrow m_2\rightarrow m_3$ ⟹a_0

$\rightarrow m_0'$，即在类A_i中引入方法m_i'，并只允许其直接访问类A_i的表层子结构m'_{i+1}，经代入替换后易知本转换是等价的。

2.设m_i是函数成员，则转换规则为：

```
class A_c{…; m_0; …}; ⟹class A_0{
    …;
    m_0;
    A_1  arg_1;
    m_0'{this→m_0''(this→m_0); }
    m_0''(A_1 arg_1)  {arg_1→m_1'; }
    …};
```

对类A_1，A_2都进行类似的转换，即在类A_i中引入中间参量arg_{i+1}、方法m_i'和m_i''，并只允许方法m_i''直接访问arg_{i+1}，从而将$a_0\rightarrow m_0\rightarrow m_1\rightarrow m_2\rightarrow m_3 \Longrightarrow a_0\rightarrow m_0'$，经过代换易知，本转换是等价的。

3.上述情形1，2是针对$a_0\rightarrow m_0\rightarrow m_1\rightarrow m_2\rightarrow m_3$的，对于$a_0\rightarrow m_0\rightarrow\cdots\rightarrow m_k$ $(k>3)$时，不断重复采用上述1，2中的转换规则进行归纳，即可将其“展开”成k个“平坦”的成员表达式this→m_i→m'_{i+1}或this→m_i''(this→m_i)以及$arg_{i+1}\rightarrow m'_{i+1}$，以遵循Demeter/C++法则的要求，因此本定理成立。□

3.示例说明

下面我们举例说明C++程序的非良好风格向相应良好风格形式的转换：

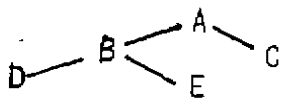

图2　复合结构类A

设类A是复合结构类，其组成树如图2所示：B、C、D、E为其嵌套成员类，并在有关类中均提供一成员函数search(X *x)，以在相应类对象中搜索信息x（x是类X的对象指针），如搜索成功则返回非零，搜索是以其组成树为目标，按深度优先方式进行的，相应的C++程序形式可以是：

```
#define OK 1
class A {
  public:
    B *b;
    C *c;
    int search(X *x){
        if (b→d→search(x)) return OK;
        else if (b→e→search(x)) return OK;
            else c→search(x);
        }
};
class B{
  public:
    E *e;
    D *d;
};
class C{
  public:
    int search(X *x){…}
};
class D{
  public:
    int search(X *x){…}
};
class E{
  public:
    int search(X *x){…}
};
class X{…};
```

可以看到成员函数A::search(X *x)中的嵌套成员表达式b→d→search(x)和b→e→search(x)破坏了Demeter/C++法则，因此必须将其进行转换。由于d为数据成员，则采用上述定理证明中情形1的转换规则，将该嵌套成员表达式“展开”，相应地将程序转换成：（由于此时嵌套成员表达式位于成员函数内部，则this可以省去）

```
class A{
  public:
    B *b;
    C *c;
    int search(X *x){
    if (b→search(x)) return OK;
    else c→search(x);
    }
```

```
;
class B{
  public;
    E *e;
    D *d;
    int search(X *x){
        if (d->search(x)) return OK;
        else e->search(x);
        }
};
```

五、小 结

综上所述，我们在指出OOP良好风格特征的基础上，基于Demeter法则，提出并证明了适用于C++的良好OOP风格法则——Demeter/C++法则及其转换方法。

关于Demeter/C++法则，可供进一步考虑的问题包括：

1.由于该法则中的类型可以在C++编译时静态确认，因此可以在C++编译器的类型检查部分嵌入对Demeter/C++法则的静态检查，在编译器中设置相应的任选项，供用户选择是否由编译器"强制地"进行C++的良好OOP风格检查。

2.开发一个将非良好OOP风格的C++程序转换到相应良好风格形式的转换工具。

3.对Demeter法则按其它OOPL(如Smalltalk)的概念进行解释，以得到适用于相应OOPL良好风格程序设计的Demeter法则。

参 考 文 献

[1] Bjarne Stroustrup, The C++ Programming Language, Addison-Wesley, 1986

[2] B. J. Cox著，杨芙清，章远阳，陈钟译：面向对象计算：一种循序渐进的途径，北京大学出版社，(待出版)

[3] K. Lieberherr, I. Holland & A. J. Riel, Object-Oriented Programming :An Objective Sense of Style, Proc. of OOPSLA'88, pp323-334

[4] A. Synder, Encapsulation and Inheritance in Object-Oriented Programming Languages,Proc. of OOPSLA'86, pp38-45

[5] Peter Wegner, Dimensions of Object Based Language Design, Proc. of OOPSLA'87, pp168-181

[6] 杨芙清，方裕，章远阳：分类机制与面向对象程序设计，计算机工程，(待发表)

下期主要内容预告

机器翻译与自然语言处理的现状与趋势
面向对象程序设计中的类型理论
程序设计语言中的值与对象
分布式数据库管理系统C-POREL之回顾
时态数据库述评
基于逻辑规则的递归查询之自底向上处理
面向对象数据库中的版本管理模型
人工智能与软件工程