行为抽象方法及其在设计模式中的应用*)

张志祥[1] 李庆华[1] 贲可荣[2]

(华中科技大学计算机学院 武汉430074)[1] (海军工程大学计算机系 武汉430033)[2]

摘 要 本文提出了一种新的面向对象设计方法,即行为抽象方法,分析了其原理、适用性和效果,给出了行为抽象的形式化定义,研究了行为抽象与众多设计模式之间的关系。

关键词 行为抽象,形式化定义,设计模式

Behaviour Abstraction Meshod and its Application in Design Paterns

ZHANG Zhi-Xiang[1] LI Qing-Hua[1] BEN Ke-Yong[2]

(College of Computer Science, Huazhong University of Science and Technology, Wuhan 430074)[1]

(Department of Computer Science, Navy Engineering University, Wuhan 430033)[2]

Abstract The paper proposes a new object-oriented design method, behaviour abstraction, analyzes its rationals, applicables and consequences. A formal definiton of behaviour abstraction is given, and the relationship between behaviour abstraction and many design paterns is studied.

Keywords Behaviour abstraction, Formal definition, Design paterns

1 概述

在面向对象的程序设计中继承是一种在类、子类以及对象之间自动地共享数据和方法的机制,它体现了抽象的概念。子类不仅可以继承父类的表示(实例变量),也可以继承行为(操作、方法等)。如果父类中的某些行为不能适用于子类,则程序设计人员可以重置这些方法,即重写方法或增加新的实现部分。继承性允许程序设计人员在设计新的类时只需考虑与已有的父类所不同的部分,而继承父类的内容作为自己的内容。

本文提出的行为抽象方法是一种特殊的继承的抽象方法,也是一种非常有效的面向对象设计方法。恰当地使用这种方法可以产生柔性的、易维护的设计。作为对常见设计问题的解决方案的设计模式[1]广泛地使用了行为抽象方法。

本文研究行为抽象的原理,分析了这种方法的适用性和效果,给出了对行为抽象的形式化定义,并研究了行为抽象与众多设计模式之间的关系。

2 行为抽象及其形式化描述

2.1 继承与代理

面向对象系统中功能重用的两种最常用的技术是类继承和对象组合[1]。类继承是根据其它的类实现(父类)定义一个新的类(子类),这种通过生成子类的重用方式又叫"白盒重用"。类继承是在编译时静态定义的,因此可直接使用,并方便地实现重用;缺点是无法在运行时改变继承于父类的实现,且在一定程度上破坏了封装性。

对象组合是指在一个对象中包含另一个对象或者对象指针,通过将请求委托给另一个对象来完成对请求的响应。因此这种方式也叫做代理。代理的主要优点是便于在运行时组合对象操作或者改变这些操作的组合方式。这种重用风格称为"黑盒重用"。

在面向对象程序设计中提倡尽量使用组合重用,这样可以带来如下好处:不破坏对象的封装性;运行时可以用一个对象代替另一个对象;实现上存在较少的依赖关系。

2.2 行为抽象

如果需求给定,通常我们可以很容易地获得一种直观的设计。但是,当需求经常发生变化时,设计出灵活性好的、柔性的面向对象程序结构就非常困难。一种解决方法是考虑系统在它的生命周期内能发生怎样的变化[2]。这需要做两件事情:

(1)将变化的概念进行封装。这里的封装不仅仅指数据隐藏。经常使用抽象类对其子类进行隐藏,这也是一种封装。抽象类和各个子类之间存在继承关系。

(2)客户代码中使用对抽象类的引用,可以隐藏各个子类的细节,以统一的方式处理各种变化的情况,客户代码与抽象类之间的关系是代理关系。

例如,假设我们要设计一个能显示不同格式的图形的软件。图形数据放在文件中,不同格式的图形对应于不同的显示方式。可以采用不同的方式处理这种行为的多样化(不同的显示方式):

1. 用一个数据成员作为标志区分图形文件的格式;

2. 从 *image* 类中派生出不同的子类,每种子类对应一种图形格式,见图1左。

第一种方法存在着紧耦合的问题,它大量使用标志进行条件判断,指导代码切换,会导致代码相当凌乱。而且当增加一个新的类型时会直接导致用户代码的改变。第二种方法则要求用户直接管理各个子类,用户面对了过多的细节。

第三种方法是在 *image* 中增加一个具有相应行为的对象(引用)*imp*,见图1(右)。*image* 中的 *draw*() 调用了 *imp* 的 *display*(),也就是说 *image* 将请求委托给了 *imp*。*imp* 实现的是 *image* 的行为,*imp* 指向不同的对象,那么 *image* 就具有不同的行为。

*)本文得到海军工程大学科研基金资助。张志祥 博士生。李庆华 博士生导师,教授。贲可荣 教授。

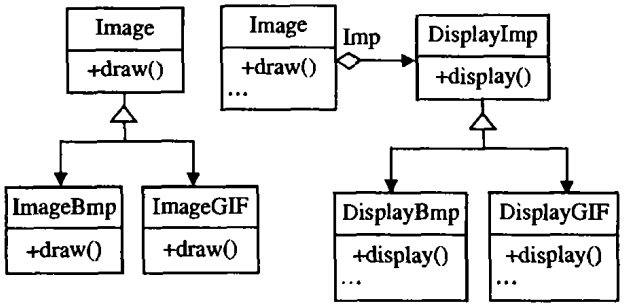

图1 对 image 的不同设计

我们称这种将不同的行为对象化(也就是具有相互独立的实现对象),可以在运行时交换这些对象以实现不同的行为的方法为"行为抽象"。

文[3]中提出了设计模式中适用的5种转换方法。其中有一种方法称为"泛化";为一组类增加一个父类,各子类可以对行为或表示进行有选择的特例化。如果父类定义子类的行为接口,而子类继承并实现父类的行为,并利用多态性实现动态联编,那么,这种泛化就是行为抽象。

行为抽象的典型结构如图2所示,使用行为抽象具有如下好处:

·封装性和模块化:行为被类型化,并封装在类中;

·可配置和可定制:客户可以在运行时选择具体的对象而改变或定制行为;

·可扩展,单点演化:要实现新的行为,只需增加新的类,而不影响已有的类;

·效率方面:尽管由于增加了新的继承层次而可能降低时间空间效率,但是客户可以动态地选择效率最高的类。

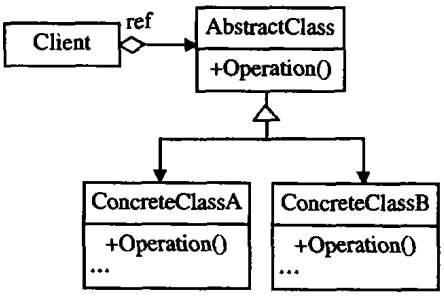

图2 行为抽象的结构

2.3 行为抽象的形式化定义

sigma 演算是一种创建面向对象语言的指称语义的方法[4]。这里采用 sigma 演算描述行为抽象。

在 sigma 演算中,用$<:$表示类之间的继承关系,例如:

$A<:B$ 表示 A 继承 B,它反映了 IS-A 的关系。

除了这种继承关系之外,定义行为抽象需要使用依赖性关系和抽象接口的概念。

定义1(依赖性关系$\ll_m$) 如果对象 a 的方法体 f 中存在一个对对象 b 的方法 g 的调用,则 $a.f \ll_m b.g$,表示 $a.f$ 依赖于 $b.g$。

依赖性关系反映了 USE-A 的关系。

定义2(抽象接口) 如果一个类 A 的方法 *Operation* 是抽象方法,也就是说,类 A 的子类必须实现 *Operation*,那么我们称(A,*Operation*)为抽象接口,记作 *AbstractInterface*(A,*operation*)。

定义3(行为抽象) 如果(*AbstractClass*,$Operation_i^{i\in 1\cdots n}$)是抽象接口,$ConcreteClass_j^{j\in 1\cdots m}$ 是 *AbstractClass* 子类,而 *Client* 类中含有对 *AbstractClass* 对对象的引用 *ref*,那么就称(*AbstractClass*,$ConcreteClass_j^{j\in 1\cdots m}$,*Client*)是行为抽象,记作 *Objectifier*(*AbstractClass*,$ConcreteClass_j^{j\in 1\cdots m}$,*Client*)。

下面按照文[5]中的方法以规则的形式给出 *Objectifier*(*AbstractClass*,$ConcreteClass_j^{j\in 1\cdots m}$,*Client*)的严格定义:

$$AbstractClass:[I_i:B_i^{i\in 1\cdots n}] \quad (1)$$

$$AbstractInterface(AbstractClass,I_i^{i\in 1\cdots n}) \quad (2)$$

$$ConcreteClass_j<:AbstractClass^{j\in 1\cdots m} \quad (3)$$

$$Client:[ref:AbstractClass] \quad (4)$$

$$Objectifier(AbstractClass,ConcreteClass_j^{j\in 1\cdots m},Client) \quad (5)$$

其中(1)表示类 *AbstractClass* 中有若干方法 I_i;(2)表示 *AbstractClass* 中的 I_i 是抽象接口,也就是说,*AbstractClass* 的子类需要实现 I_i 方法;(3)表示 $ConcreteClass_j^{j\in 1\cdots m}$ 是 *AbstractClass* 子类,这样按照(2)的约定,$ConcreteClass_j$ 中要实现具体的 I_i 方法;(4)表示 *Client* 类中含有对 *AbstractClass* 对对象的引用 *ref*。

图1(右)就是一个行为抽象的例子,下面给出用 C++ 描述的示例代码:

```
class DisplayImp{
Puclic:
  virtual display()=0
...
}
class Display BMP:public DisplayImp{
public:
virtual void display(){…};//显示 bmp 图形文件
...
}
class DisplayGIF:public DisplayImp{
public:
  virtual void display(){…};//显示 GIF 图形文件
  ...
}
class Image{
public:
  void draw(){imp->display()};
  void SetImpl(DisplayImp*dp):imp(dp){};
private:
  DisplayImp*imp;
  ...
}
main(){
  DisplayImp * dp1;dp2;
  Image pic;
  dp1=new displayBMP("c:\\test. bmp");
  pic. setImpl(dpI);
  pic. Draw();
  dp2=new DisplayGIF("c:\\test. bmp");
  pic. setImpI(dp2);
  pic. Draw();
  delete dp1;
  delete dp2;
  ...
}
```

3 设计模式中的行为抽象

许多设计模式都使用了行为抽象的方法,但它们的目的和需求更有针对性。例如,Bright 模式中使用了行为抽象,而抽象的行为是不同的实现方法;Builder 模式中抽象的是对不同对象的创建方法;Command 模式中抽象与各种命令有关的行为;Iterator 模式抽象对对象结构的遍历行为;Observer 模式中抽象的是与上下文有关的行为;State 模式中抽象的是与状态有关的行为;Strategy 模式中抽象的是复杂的算法;Visitor 模式中抽象的是与类型有关的行为(复合结构中单个对象的类型)。

(下转第151页)

代数系统中的安全隐患在复杂代数系统中应该同样存在，并且在复杂代数中，公钥体制的安全性仍较大地依赖于计算机的运算速度。

另一方面，我们从REESSE1和REESSE2体制中看到，对运算数进行组合是提高公钥体制安全性的另一条途径，这是一条数学与软件相结合的途径。这种途径要求公钥体制必须是序列密码体制。在REESSE1和REESSE2体制中，运算数的组合与排列是由杠杆函数 $f(i)$ 实现的，使得体制的安全性对计算机运算速度的依赖程度非常低。通过运算符组合来提高安全性有如对单个CPU提高速度，这种提高是有极限的。通过运算数组合来提高安全性有如对多个CPU并联提速，这种提高是无极限的。

5.4 易于硬件实现且运算速度快

REESSE2体制的加密算法与解密算法中大部分是模加运算和逻辑判断，所以，易于硬件，特别是简单硬件与CPU实现。

与流行的RSA体制、ECC(椭圆曲线上的ElGamal)体制相比，REESSE2也有速度上的优势(即使考虑到冗余加密和 $O(n^3)$ 时间复杂度解密等因素)。这是因为：(1)REESSE2的模数可以不超过192比特，而RSA至少为1024比特，ECC也常为192比特；(2)REESSE2的运算符主要是基本的带模加法、而RSA是带模乘法，ECC是复合运算符，由于乘法、复合运算符是由基本运算符(加法、减法等)构成的，因此，完成一个复合运算所需的时间要远远多于一个基本运算所需的时间。

结束语 在本文，我们虽然提出了REESSE2公钥体制的理论算法，但是在技术实现上，还应该对REESSE2的解密算法进行进一步优化，争取使其时间复杂度降至 $O(n^2)$ 的水平。还应在模数和序列长度确定的情况下，测定最少冗余加密次数。最后，我们期待同仁对REESSE2公钥体制做出更深入的安全性分析。

参考文献

1 卢开澄. 计算机密码学(第2版). 北京：清华大学出版社，1998

2 Schneier B(美). Applied Cryptography(应用密码学，吴世忠，祝世雄等译). 北京：机械工业出版社，2000

3 冯登国. 密码分析学. 北京：清华大学出版社，2000

4 冯克勤，李尚志. 近世代数引论. 合肥：中国科学技术大学出版社，2002

5 苏盛辉. REESSE1公开密钥密码体制. 计算机工程与科学，2003(5)

6 苏盛辉，李国华，王其文. 模运算的新性质及求模逆元的递归算法. 信息安全与通信保密，2001(11)

(上接第136页)

总之，行为抽象可用于以下几种场合：

·需要从类中分离出行为，以便可以交换、保存、修改、共享或者调用相互独立的行为对象；

·需要在运行时选择行为；

·一些类的区别仅仅在于一个或者几个方法的不同。这样，增加新的类，实现这些不同的行为，可以以统一的方式使用具有不同行为的类。

·在选择不同的行为时，需要使用大量的条件判断代码。

下面使用2中的方法，以bridge模式为例，说明设计模式与行为抽象的关系：

Objectifier (Implementor, ConcreteImplementor$_j^{j\in 1..m}$, Abstraction) (1)

AbstractInterface(Abstraction,Operation) (2)

r:Abstraction (3)

C:ConcreteImplementor$_j^{j\in 1..m}$ (4)

r.operation$\ll_m c$.operationImp (5)

Bridge (Abstraction, Implementor, concreteImplementor$_j^{j\in 1..m}$,operation,operationImp) (6)

可见，Bridge模式中使用了行为抽象方法，在此基础上在客户(这里是Abstraction)端增加了一个抽象接口。

总结 面向对象中封装性不仅仅体现在对象的状态和行为的封装，行为抽象中也体现了一种封装性：抽象类封装了各个具体子类。通过委托，客户不需了解子类的细节，就可以具有不同的行为。

行为抽象方法所产生的结构被认为是在众多设计模式中使用的基本设计模式。我们认为，在设计模式中存在着一些基本的设计方法，它们只使用类、对象、方法、方法调用和数据成员的概念，但能反映基本的OO设计原则。我们可以从这些简单的基本设计方法入手学习面向对象设计方法。另外，研究这些基本设计模式，可以更好地学习和掌握更加复杂的设计模式，发现设计模式之间的关系以及发掘新的设计模式。

参考文献

1 Gamma E,Helm R,Johnson R,Vlissides J. Design Patterns. Addison Wesley,1995

2 Shalloway A,Trott J R. Design Patterns Explained:A New perspective on Object-Oriented Design. Addison-Wesley,2002

3 熊兵舫，张志祥. 设计模式的软件度量分析. 青岛大学学报，2003，增刊

4 Abadi M,Cardelli L. A Theory of Objects. Springer-Verlag New York,Inc.,1996

5 Smith J M,Stotts D. Elemental design patterns:A link between architecture and object semantics:[Technical Report TR-02-011]. Univ. of North Carolina,2002

6 Zimmer W. Relationships between design patterns. In pattern Languages of Program Design. Addison-Wesley,1994